WYKORZYSTAJ JAVASCRIPT W NAJLEPSZY SPOSÓB!

Nowoczesny język

JavaScript

Larry Ullman

Helion

Tytuł oryginału: Modern JavaScript: Develop and Design

Tłumaczenie: Rafał Jońca

ISBN: 978-83-246-5148-1

Authorized translation from the English language edition, entitled: MODERN JAVASCRIPT: DEVELOP AND DESIGN; ISBN 0321812522; by Larry Ullman; published by Pearson Education, Inc, publishing as Peachpit Press. Copyright © 2012 by Larry Ullman.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/nojejs.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/nojejs
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Wprowadzenie **9**
Witamy w języku JavaScript **13**

1. (Ponowne) wprowadzenie do języka JavaScript **15**
Czym jest JavaScript? 16
Historia języka JavaScript 17
Czym JavaScript nie jest? 25
Porównanie JavaScript z… 26
Dlaczego JavaScript to dobra rzecz? 28
Wersje języka JavaScript i ich obsługa w przeglądarkach 29
Cele programowania w języku JavaScript 30
Podsumowanie 31

2. JavaScript w akcji **33**
Wybór typu dokumentu 34
Wprowadzenie do HTML5 36
Dodanie kodu JavaScript do kodu HTML 40
Podstawowe podejścia do programowania 42
Łączymy kod 45
Podkradnij ten kod JavaScript 52
Podsumowanie 53

3. Narzędzia dla programistów 55
Wielka debata — edytor tekstu czy IDE? 56
Przeglądarka internetowa — Twój przyjaciel i Twój wróg 62
Testowanie na wielu przeglądarkach 68
Testowanie kodu JavaScript 69
Błędy i testowanie 71
Zasoby dostępne online 79
Podsumowanie 80

4. Proste typy zmiennych 81
Podstawy działania zmiennych 82
Liczby 86
Praca z ciągami znaków 96
Przeprowadzanie konwersji typów 103
Powtórka i dodatkowa nauka 106
Podsumowanie 107

5. Struktury sterujące 109
Podstawy instrukcji warunkowych 110
Dodatkowe elementy instrukcji warunkowych 117
Bardziej wyrafinowane warunki 125
Podstawy pętli 131
Powtórka i dodatkowa nauka 135
Podsumowanie 136

6. Złożone typy zmiennych 137
Generowanie dat i czasu 138
Korzystanie z tablic 151
Korzystanie z obiektów 163
Tablice czy obiekty? 168
Powtórka i dodatkowa nauka 169
Podsumowanie 170

7. Tworzenie funkcji 171
Podstawy 172
Funkcje jako obiekty 186
Tematy zaawansowane 192
Powtórka i dodatkowa nauka 198
Podsumowanie 199

8. Obsługa zdarzeń 201
Obsługa zdarzeń — przypomnienie 202
Tworzenie procedur obsługi zdarzeń 202
Tworzenie biblioteki z funkcjami pomocniczymi 206
Rodzaje zdarzeń 208
Zdarzenia a dostępność witryny 215
Zdarzenia i stopniowe ulepszanie 215
Zaawansowana obsługa zdarzeń 217
Powtórka i dodatkowa nauka 226
Podsumowanie 228

9. JavaScript i przeglądarka internetowa 229
Okna dialogowe 230
Korzystanie z okien i obiektu window 232
Modyfikacja DOM 247
JavaScript i CSS 256
Korzystanie z ciasteczek 262
Wykorzystanie funkcji czasowych 269
Powtórka i dodatkowa nauka 271
Podsumowanie 273

10. Korzystanie z formularzy 275
Ogólne uwagi dotyczące formularzy 276
Pola i obszary tekstowe 281
Listy wyboru 283
Opcje wyboru 287
Przyciski opcji 289
Obsługa przesyłu plików 290
Wyrażenia regularne 292
Łączymy wszystko razem 301
Powtórka i dodatkowa nauka 304
Podsumowanie 306

11. Ajax 307
Podstawy Ajax 308
Korzystanie z innych rodzajów danych 319
Skrypt po stronie serwera 322
Przykłady użycia technologii Ajax 324
Powtórka i dodatkowa nauka 336
Podsumowanie 337

12. Zarządzanie błędami 339
Zgłaszanie i przechwytywanie błędów 340
Wykorzystanie asercji 343
Testy jednostkowe 344
Powtórka i dodatkowa nauka 349
Podsumowanie 350

13. Frameworki 351
Wybór odpowiedniego frameworka 352
Wprowadzenie do jQuery 353
Wprowadzenie do YUI 363
Biblioteki 373
Powtórka i dodatkowa nauka 374
Podsumowanie 375

14. Zaawansowany kod JavaScript 377
Definiowanie przestrzeni nazw 378
Tworzenie własnych obiektów 379
Prototypy i sposób ich działania 384
Korzystanie z domknięć 386
Inne sposoby rozpoznawania typu 389
Minifikacja kodu 390
Powtórka i dodatkowa nauka 392
Podsumowanie 393

15. Przykładowy projekt — JavaScript i PHP razem 395
Określenie celu 396
Tworzenie bazy danych 397
Konstrukcja witryny 398
Tworzenie wersji bez użycia kodu JavaScript 399
Tworzenie skryptów dla żądań Ajax 406
Dodanie kodu JavaScript 409
Dokończenie przykładu 419
Powtórka i dodatkowa nauka 420
Podsumowanie 421

Skorowidz 423

ROZDZIAŁ 2.

JAVASCRIPT W AKCJI

JavaScript, podobnie jak inne obiektowe języki programowania, jest czymś, co leniwy programista może wykorzystywać bez pełnego zrozumienia zasad jego działania. Ta cecha to zarówno zaleta, jak i poważne brzemię. Choć niniejsza książka uczy właściwego używania języka JavaScript, w tym rozdziale pojawią się przykłady wzięte z życia bez przeprowadzania nudnego, formalnego szkolenia. Z pewnością nie jest to ortodoksyjny sposób nauki, ale zakładam, iż Czytelnik miał wcześniej przynajmniej minimalny kontakt z językiem JavaScript. Co więcej, ten początkowy rozdział określa cele, które będą realizowane w kilku kolejnych rozdziałach. Dodatkowo w rozdziale tym poruszę podstawowe tematy związane z projektowaniem i tworzeniem witryn internetowych, w szczególności związane z wpływem `DOCTYPE` na niemal wszystko, co robisz.

WYBÓR TYPU DOKUMENTU

Gdy zaczynałem pracę nad witrynami internetowymi, nie zdawałem sobie sprawy z istotności deklaracji typu dokumentu, czyli tak zwanego DOCTYPE. W tamtym czasie wierzyłem, że stosowałem HTML 3.2, więc musiałem rozpoczynać strony WWW od następującego wpisu:

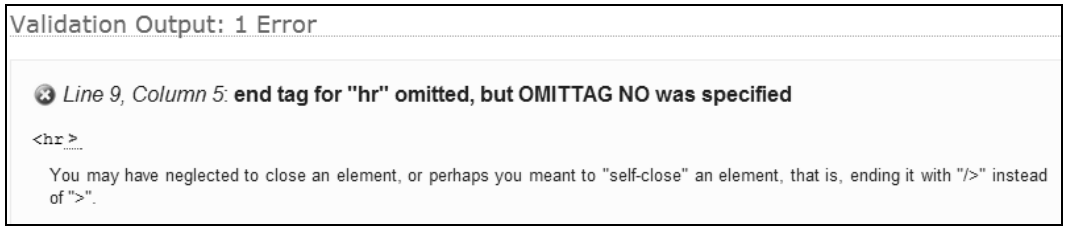
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
```

Wpis DOCTYPE do deklaracja wersji języka HTML wykorzystywanego przez stronę WWW, a każda nowa wersja języka wprowadza nowe funkcjonalności (na przykład nowe elementy formularzy). HTML 2.0 nie obsługiwał jeszcze tabel, a HTML 3.2 miał ograniczone wsparcie dla arkuszy stylów. Przez ostatnie lata najpopularniejszymi wpisami DOCTYPE były te, które dotyczyły HTML 4.01 i XHTML 1.0. XHTML to w zasadzie HTML, ale z bardziej restrykcyjnymi regułami zapewniającymi zgodność ze składnią XML (więcej na ten temat w następnym podrozdziale). Zarówno HTML 4.01, jak i XHTML 1.0 występują w trzech odmianach: Strict, Transitional i Frameset. Pierwszy jest najbardziej restrykcyjny i zapewnia obsługę mniejszej liczby elementów. Wersja Transitional to wersja Strict z dodatkową obsługą elementów wycofywanych z użycia. Wersja Frameset to Transitional z obsługą tak zwanych ramek.

Jeśli myślisz podobnie jak ja, wybierzesz HTML lub XHTML, a następnie prawdopodobnie wersję Transitional, ponieważ jest mniej restrykcyjna:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Być może masz nawyk sprawdzania utworzonych stron WWW i korzystasz przy tym z narzędzia W3C Markup Validator (*http://validator.w3.org/*). Jeśli tak, to wiesz, że narzędzia tego typu przeprowadzają walidację na podstawie rodzaju wpisu DOCTYPE. Jeżeli w kodzie strony pojawi się wycofany element lub ramka, a używa się trybu Strict, zostanie to wychwycone. To samo dotyczy niezastosowania się do składni XML w przypadku dokumentów XHTML (rysunek 2.1).



RYSUNEK 2.1. Narzędzie do walidacji sprawdza, czy dokument został sporządzony zgodnie z deklarowanym standardem

■ **UWAGA:** Wpis DOCTYPE musi być pierwszym elementem na stronie WWW. Przed nim nie może wystąpić nawet znak spacji.

Zapewne doskonale znasz wszystkie przedstawione tutaj informacje, ale jeśli nie lub nie interesowałeś się zbytnio samą walidacją, można to zrozumieć. Rzeczywistym celem nie jest zapewnienie, by walidator nie zgłosił błędów, ale by strony WWW prawidłowo funkcjonowały w przeglądarce internetowej. Tutaj wpis DOCTYPE również odgrywa istotną rolę: **przeglądarka internetowa wybierze jeden z dwóch trybów działania na podstawie wpisu DOCTYPE**. Jeśli wpis istnieje, przeglądarka włączy tak zwany tryb zgodności ze standardami (nazywany po prostu trybem standardów), czyli będzie obsługiwała kod HTML, CSS i JavaScript w sposób przewidziany w standardzie. Jeśli dokument nie zawiera wpisu DOCTYPE lub jest on nieprawidłowy, przeglądarka włączy tak zwany tryb quirks, który będzie traktował kod HTML, CSS i JavaScript podobnie jak starsze przeglądarki interne-

towe. Na przykład przełączenie przeglądarki Internet Explorer 8 w tryb quirks spowoduje wyświetlanie przez nią stron WWW w taki sam sposób, jak robiła to przeglądarka Internet Explorer 5.5 (IE 5.5 ma już grubo ponad 10 lat, więc wyobraź sobie, jak będzie w takiej sytuacji wyglądała nowoczesna, lśniąca witryna).

Czym jest DOM?

DOM, wspomniany w rozdziale 1. (skrót od *Document Object Model* — obiektowy model dokumentu), to sposób reprezentacji danych XML i poruszania się po nich, który obejmuje również HTML i XHTML. Standardem DOM zarządza World Wide Web Consortium (W3C). Obecnie obowiązującym standardem jest DOM Level 3, wydany w 2004 roku. Choć standard ten został wydany wiele lat temu, nadal nie jest w pełni lub jednolicie obsługiwany przez wszystkie przeglądarki internetowe. Trzeba mocno podkreślić, że DOM nie stanowi części języka JavaScript, ale kod JavaScript najczęściej używa DOM, by modyfikować elementy strony WWW lub otrzymywać od niej zdarzenia (tak zwana **modyfikacja DOM**).

Potwierdzenie trybu przeglądarki

Niektóre przeglądarki ułatwiają poznanie trybu, w którym wyświetlają wczytaną stronę WWW. Przeglądarka Firefox wyświetla tę informację w oknie *Informacje o stronie* otwieranym z poziomu menu *Narzędzia* (poszukaj wiersza *Tryb wyświetlania*). Przeglądarka Opera wyświetli stosowną informację po wybraniu *Widok/Narzędzia deweloperskie/Informacje o stronie* (poszukaj wiersza *Tryb wyświetlania*). Pozostałe przeglądarki nie wyświetlają tej informacji w tak przystępny sposób, ale w rozdziale 9. pokażę, jak ją uzyskać przy użyciu kodu JavaScript.

Jakby brakowało innych problemów, okazuje się, że w niektórych przeglądarkach nawet poprawny wpis `DOCTYPE` może powodować włączenie trybu quirks. Podobna sytuacja może wystąpić, gdy przeglądarka znajdzie na stronie niepoprawne elementy, choć wpis i pozostała część kodu są prawidłowe. Oznacza to, że poprawność wpisu `DOCTYPE` ma duży wpływ na jednolity wygląd i zachowanie stron WWW we wszystkich przeglądarkach. W książce, podobnie jak w życiu, trzeba podjąć decyzję dotyczącą wykorzystywanego wpisu `DOCTYPE`. Wybrałem następujący:

```
<!DOCTYPE html>
```

Ma on kilka zalet:

- łatwiej go napisać, więc istnieje niewielkie prawdopodobieństwo błędu;
- ma mniej znaków, więc przesyłane pliki HTML będą mniejsze i szybciej trafią do przeglądarki;
- jest obsługiwany przez wszystkie główne przeglądarki internetowe;
- powoduje automatyczne włączenie trybu standardów.

Jeśli jeszcze nie natknąłeś się na ten wpis `DOCTYPE`, wynika to zapewne z faktu, iż to nowy oficjalny wpis dla języka HTML5. Skoro jednak HTML5 nie jest jeszcze oficjalnym standardem i nadal trwają nad nim prace, to czy można z niego bezpiecznie skorzystać? Przyjrzyjmy się temu zagadnieniu dokładniej.

UWAGA: Nie wszystkie przeglądarki internetowe przełączają się między trybami w ten sam sposób. Przeglądarka Opera od lat domyślnie stosuje tryb standardów, a przeglądarki z rodziny Mozilla wykorzystują własny tryb „standardu z wyjątkami".

Wprowadzenie do HTML5

Gdy powstawała ta książka, był rok 2012 i HTML5 wydawał się ogromnym przedsięwzięciem. W takiej lub innej postaci tak naprawdę funkcjonuje on od wielu lat, ale dopiero oficjalne wstrzymanie prac nad XHTML 2.0 spowodowało, że HTML5 stał się de facto nowym standardem. Nie został jeszcze w pełni sformalizowany czy uznany za oficjalny standard, co oznacza, że przyszła wersja oficjalna może różnić się od omawianej tutaj. W tak rozproszonym środowisku, jakim jest internet, mądrze byłoby unikać korzystania z tej nowinki. Obecnie jednak najczęściej wybiera się model pośredni, czyli stosuje się niektóre funkcje HTML5, zapewniając w razie potrzeby wersje alternatywne. Przyjrzymy się ogólnemu szablonowi strony WWW w HTML5, a następnie omówię kilka nowych elementów formularzy HTML5.

■ **WSKAZÓWKA:** HTML5 nie jest jednym standardem. To raczej nazwa oznaczająca nowy standard HTML i zbiór towarzyszących mu technologii.

Szablon HTML5

Poniższy blok kodu przedstawia szablon HTML5, z którego będę korzystał we wszystkich przykładach prezentowanych w książce. Przyjrzyj się mu — nieco dalej omówię jego poszczególne części.

```
<!doctype html>
<html lang="pl">
<head>
   <meta charset="utf-8">
   <title>Szablon HTML5</title>
   <!--[if lt IE 9]>
   <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
   <![endif]-->
</head>
<body>
   <!-- szablon.html -->
</body>
</html>
```

Jak wcześniej wspomniałem, użycie pierwszego wiersza w takiej postaci spowoduje włączenie trybu standardów, który jest pierwszym pożądanym celem. Następnie pojawia się element `html` zawierający elementy `head` oraz `body`. Co ciekawe, HTML5 nie wymusza użycia elementu `head`, ale dla mnie takie wymuszenie byłoby bardzo dziwne. HTML5 nadal wymaga znacznika `title`, nawet jeśli zabraknie elementu `head`. Warto również wyrobić w sobie nawyk podawania sposobu kodowania znaków. Jak można zauważyć, również znacznik `meta` dotyczący kodowania znaków uległ uproszczeniu (wiersz 4.). Jeżeli zagadnienie kodowania znaków nie jest Ci znane, poszukaj informacji na ten temat. Na razie wystarczy wiedzieć, że `utf-8` spowoduje użycie kodowania UTF-8 obsługującego znaki większości języków świata. Dodałem również informację o języku strony WWW (wiersz 2.) jako atrybut `lang` znacznika `html`, choć nie jest to wymóg.

■ **UWAGA:** Ponieważ informacja o kodowaniu znaków musi pojawić się możliwie jak najwcześniej, zawsze umieszczaj ją tuż poniżej znacznika otwierającego `head` i przed znacznikiem `title`.

To podstawowa składnia dokumentu HTML5. W następnym podrozdziale omówię główny powód, dla którego używam HTML5 w niniejszej książce — znacznie rozbudowany zestaw elementów formularzy. Wróćmy jednak do dwóch dodatkowych elementów dotyczących szablonu

HTML5. W wielu przykładach będzie wykorzystywany zewnętrzny arkusz stylów; poprawna składnia jego osadzania wygląda następująco:

```
<link rel="stylesheet" href="css/styles.css">
```

Zauważ, że element `link` nie używa atrybutu `type`, ponieważ automatycznie zakłada się, iż w momencie ustawienia wartości atrybutu `rel` na `stylesheet` przyjmie on wartość `text/css`.

Ponadto HTML5 definiuje wiele nowych elementów semantycznych, takich jak `article`, `footer`, `header`, `nav` i `section`. Ich powstanie wiąże się z wynikami badań nad najczęściej stosowanymi na stronach WWW identyfikatorami i nazwami klas. Na przykład w HTML4 wielu projektantów używało elementu `div` z identyfikatorem `header`, by wskazać nagłówek strony WWW. Dzięki temu łatwiej było nadać mu odpowiedni styl CSS. W HTML5 wystarczy utworzyć element `header` i jemu przypisać odpowiedni styl. Większość starszych przeglądarek, które powstały przed HTML5, obsłuży nowe elementy i pozwoli nadać im style bez najmniejszych problemów. Niestety, przeglądarki Internet Explorer starsze niż wersja 9. nie potrafią nadawać stylów nieznanym elementom, co oznacza, że użytkownik IE8 (lub starszej wersji) domyślnie nie zobaczy odpowiedniego stylu. Rozwiązaniem jest funkcjonalny kawałek kodu JavaScript, nazywany HTML5 shiv, napisany przez bardzo sprytne osoby. Kod ten generuje nowe elementy za pomocą kodu JavaScript, co powoduje rozpoczęcie rozpoznawania ich przez przeglądarkę, a co za tym idzie, nadawanie im odpowiednich stylów. Biblioteka HTML5 shiv jest udostępniana na zasadzie otwartego kodu i znajduje się na serwerach Google Code. Jej użycie wymaga dodania następującego fragmentu kodu:

```
<!--[if lt IE 9]>
<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

Blok ten zaczyna się i kończy komentarzem warunkowym obsługiwanym tylko i wyłącznie przez przeglądarkę Internet Explorer. Warunek sprawdza, czy aktualna przeglądarka jest starsza niż IE9 (`lt`). Jeśli tak, do strony zostanie dołączony element `script`. Ten komentarz warunkowy jest odczytywany tylko i wyłącznie przez przeglądarkę IE; pozostałe przeglądarki nie podejmą nawet próby wczytywania skryptu.

Zauważ, że znacznik `script`, podobnie jak wcześniejszy znacznik `link`, nie używa atrybutu `type`, ponieważ `text/javascript` jest wartością domyślną.

W rozdziale 3. wymienię kilka narzędzi do walidacji kodu HTML. Na razie wspomnę, że poza standardowym walidatorem W3C można również korzystać z walidatora HTML5, dostępnego pod adresem *http://html5.validator.nu/*. Gdy powstawała ta książka, oba były uznawane za narzędzia eksperymentalne, ale wynikało to głównie z faktu, iż sam HTML5 nie był jeszcze oficjalnym standardem!

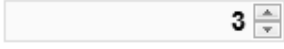
■ **UWAGA:** W niektórych prezentowanych przykładach są wykorzystane nowe znaczniki, przykłady te wymagają zatem dołączenia HTML5 shiv. Niemniej przedstawiony szablon stosuję konsekwentnie w całej książce.

ELEMENTY FORMULARZY HTML5

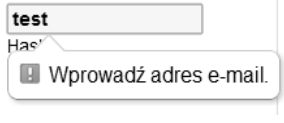
Istnieją dwa powody, dla których zdecydowałem się w niniejszej książce zastosować HTML5 pomimo tego, że nie stanowi on jeszcze oficjalnego standardu. Pierwszy powód jest oczywisty — HTML5 to z pewnością przyszłość witryn internetowych. Drugi dotyczy nowych elementów formularzy oferowanych przez HTML5, które czynią strony WWW znacznie przyjemniejszymi. W szczególności myślę o następujących typach pól tekstowych:

- adres e-mail,
- liczba,
- zakres liczb,
- wyszukiwanie,
- numer telefonu,
- adres URL.

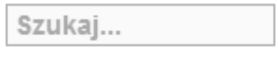
Dzięki tym elementom łatwiej wprowadzać adresy e-mail, liczby (za pomocą przycisków zmiany lub suwaka — rysunek 2.2), numery telefonów i adresy URL. W przeglądarkach, które obsługują te elementy, nastąpi ich automatyczne sprawdzenie i wyświetlenie stosownej informacji o błędzie. Na przykład pole wprowadzania adresu URL umożliwi wpisanie jedynie poprawnych adresów (gdy ten typ pola jest obsługiwany przez przeglądarkę; rysunek 2.3). Kilka typów pól ma dodatkowe zalety. Gdy użytkownik wprowadza wartość w pole adresu e-mail na urządzeniu przenośnym, takim jak iPhone, telefon automatycznie użyje wersji klawiatury przystosowanej do adresów e-mail. Pole tekstowe wyszukiwania (typ `search`) może mieć kilka dodatkowych efektów i funkcjonalności, a nawet inny wygląd (rysunek 2.4; tekst *Szukaj…* nie jest częścią domyślnego zachowania).



RYSUNEK 2.2. Nowe pole tekstowe do wpisania liczb z HTML5



RYSUNEK 2.3. Elementy formularzy HTML5 mogą zawierać automatyczne sprawdzanie poprawności danych, na przykład podanego adresu URL



RYSUNEK 2.4. W przeglądarce Safari nowe pole tekstowe wyszukiwania przyjmuje inny wygląd

Co istotne, gdy przeglądarka nie obsługuje nowych typów pól tekstowych, wyświetli użytkownikowi standardowe pole tekstowe. Przeglądarki domyślnie wyświetlają nieznane im elementy w trybie `inline`, co oznacza, że nowe pola nie spowodują katastrofy w układzie graficznym witryny!

Formularze HTML5 otrzymały również kilka nowych atrybutów wartych rozważenia. Pierwszym z nich jest `autofocus`, oznaczający element, który powinien być domyślnie wybrany po włączeniu formularza:

```
<input type="text" name="nazwa" autofocus>
```

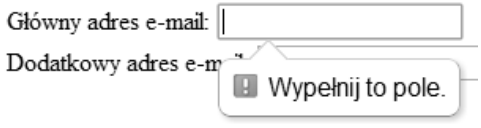
UWAGA: Gdy powstawała ta książka, nowe rodzaje pól formularzy w najszerszym zakresie obsługiwała przeglądarka Opera.

Drugim jest atrybut `placeholder`, który określa domyślny tekst pola, jeśli nie wprowadzono nowej wartości (rysunek 2.4):

```
<input type="search" placeholder="Szukaj...">
```

HTML5 wprowadza również atrybut `required` powiązany z automatycznym sprawdzaniem poprawności formularzy. Gdy zostanie zastosowany do pola tekstowego, użytkownik musi wprowadzić w tym polu wartość, aby proces walidacji mógł się zakończyć sukcesem. Jeżeli formularz wymaga adresu e-mail, użytkownik musi wprowadzić poprawny adres e-mail. Jeśli element nie jest wymagany, brak danych uznawany jest za sytuację prawidłową i tylko podanie błędnego adresu e-mail spowoduje zgłoszenie błędu (rysunki 2.5 i 2.6):

```
Główny adres e-mail: <input type="email" name="email1" required>
Dodatkowy adres e-mail: <input type="email" name="email2">
```



RYSUNEK 2.5. Włączenie wymagania wartości powoduje zgłoszenie błędu. Podana wartość musi być zgodna z typem pola (rysunek 2.6)

RYSUNEK 2.6. Jeżeli w polu opcjonalnym wprowadzono wartość, musi być ona poprawna

By ograniczyć długość tekstu wprowadzanego w polu tekstowym, użyj atrybutu `maxlength`. Sam atrybut istnieje od wielu lat, ale teraz możliwe jest wyświetlenie użytkownikowi dodatkowej informacji (w sposób zależny od przeglądarki), a nawet zastosowanie tego atrybutu do obszarów tekstu (znaczników `textarea`):

```
<textarea name="komentarz" rows="8" cols="40" maxlength="300"></textarea>
```

Aby wyłączyć automatyczne sprawdzanie formularza, dodaj atrybut `novalidate` w znaczniku otwierającym `form`:

```
<form action="strona.php" method="get" novalidate>
```

Warto w tym momencie ostrzec o możliwych komplikacjach, jeśli niektóre z przykładów z dalszych rozdziałów będą testowane w przeglądarkach obsługujących walidację formularzy z HTML5. Ponieważ w testowanych przykładach jest stosowana walidacja po stronie kodu JavaScript, musisz dodać atrybut `novalidate`. W przeciwnym razie przeglądarka nigdy nie prześle do kodu JavaScript błędnej wartości.

Znasz już podstawy HTML5, zatem możemy nareszcie powrócić do języka JavaScript!

■ **WSKAZÓWKA:** HTML5 wprowadził również nowy atrybut `pattern`, który sprawdza poprawność wartości elementu przy użyciu wyrażenia regularnego.

HTML5 KONTRA XHTML

XHTML wymaga bardzo restrykcyjnej składni wynikającej z XML, więc osobiście zawsze preferowałem tę wersję zamiast zwykłego HTML-u (dodatkowe restrykcje pomagają uniknąć błędów lub szybciej je wykryć). XHTML ma kilka reguł, które nie dotyczą zwykłego kodu HTML. W szczególności:

- ✓ elementy bez znacznika zamykającego, takie jak `img`, `input` lub `br`, muszą zostać zamknięte znakiem ukośnika w znaczniku otwierającym, na przykład:

```
<img src="plik.png" alt="obraz" />
```

- ✓ atrybuty muszą być umieszczone w cudzysłowach, jak w przykładzie powyżej;
- ✓ atrybuty zawsze muszą posiadać wartość, na przykład:

```
<option value="tak" selected="selected">Tak</option>
```

HTML5, podobnie jak wcześniejsze wersje HTML, nie wymaga stosowania restrykcyjnej składni XML. Ma to kilka implikacji, włącznie z tą, iż nie trzeba stosować żadnej z powyższych zasad. Dwa powyższe fragmenty w HTML5 można zapisać następująco i nadal będą poprawne:

```
<img src=plik.png alt=obraz />
<option value=tak selected>Tak</option>
```

Jestem w stanie opuścić ukośniki i wartości atrybutów (jeśli to możliwe), bo taka składnia jest czytelniejsza. Nadal jednak zalecam stosowanie cudzysłowów dla wartości atrybutów. Dzięki temu łatwiej określić, co jest wartością, a co już nowym atrybutem. Co więcej, istnieją sytuacje, w których użycie cudzysłowów jest obowiązkowe, na przykład gdy wartość zawiera znak spacji:

```
<img src="plik.png" alt="Moje wakacje">
```

Ponieważ niektóre atrybuty będą musiały mieć cudzysłowy z powodu stosowanych wartości, najlepiej zapewnić jednolitość kodu i stosować cudzysłowy w każdej sytuacji.

DODANIE KODU JAVASCRIPT DO KODU HTML

W tym rozdziale pojawi się kilka przykładów praktycznego kodu JavaScript, który zostanie wyjaśniony dopiero w drugiej części, dotyczącej podstaw języka JavaScript. Pewne elementarne informacje muszą jednak pojawić się już w tym rozdziale, na przykład sposób wstawiania kodu JavaScript na stronie WWW, aczkolwiek podejrzewam, że doskonale znasz te informacje.

Do osadzania kodu JavaScript na stronie HTML służy znacznik `script`:

```
<script></script>
```

We wcześniejszych wersjach języka HTML wymagano atrybutu `type`, który musiał mieć wartość `text/javascript`. W HTML5 wprowadzono drobną zmianę i brak atrybutu oznacza użycie wspomnianej wartości jako domyślnej. Jeżeli wykorzystujesz starszą wersję HTML, stosuj atrybut.

Kod JavaScript powinien znajdować się między znacznikami otwierającym i zamykającym. Przeglądarka internetowa w trakcie wczytywania strony WWW wykona znajdujący się tam kod.

Alternatywne rozwiązanie polega na użyciu zewnętrznego pliku zawierającego kod JavaScript i poinformowaniu przeglądarki o jego lokalizacji za pomocą atrybutu `src` znacznika `script`:

```
<script src="ścieżka/do/pliku.js"></script>
```

Ścieżka musi być poprawna, ale można podawać ją w wersji **względnej** (czyli względem adresu strony WWW) lub **pełnej** (patrz ramka).

Nadal bardzo popularne jest tworzenie małych fragmentów kodu bezpośrednio na stronie HTML, a nie w osobnym pliku. Gdy kod zacznie być bardziej złożony lub będzie stosowany na wielu stronach, wykorzystanie osobnych plików dla kodu zaczyna mieć większy sens, ponieważ takim kodem łatwiej zarządzać. W zewnętrznym pliku kod nie zawiera znacznika `script`, ponieważ kod ten nie znajduje się w kodzie HTML. Ponadto pliki z zewnętrznym kodem JavaScript używają rozszerzenia pliku *.js*.

Dodatkową zaletą zewnętrznych plików JavaScript jest to, iż mogą być łatwo umieszczone w pamięci podręcznej przeglądarki i użyte ponownie. Oznacza to, że jeśli kilka stron WWW korzysta z tego samego zewnętrznego pliku JavaScript, zostanie on pobrany przez przeglądarkę tylko raz.

Istnieje jeszcze pięć kwestii dotyczących znacznika `script`, o których warto pamiętać. Po pierwsze, podobnie jak w przypadku wielu innych elementów HTML, znacznik ten może pojawić się w kodzie strony WWW wielokrotnie. Do takiej sytuacji dochodzi w większości przypadków.

Po drugie, pojedynczy element `script` może zawierać kod JavaScript w sobie lub stosować zewnętrzny kod JavaScript, ale **nie jednocześnie**. Jeśli strona WWW korzysta z obu rozwiązań, musisz użyć dwóch niezależnych elementów.

Gdy jest stosowana wersja XHTML, cały kod JavaScript trzeba dodatkowo otoczyć znacznikami CTAGS, co prowadzi do powstania nieco dziwnej i mało przyjaznej składni:

```
<script>//<![CDATA[
// Właściwy kod JavaScript!
//]]></script>
```

Ścieżki pełne i względne

Typowym źródłem frustracji, szczególnie dla początkujących projektantów i programistów witryn internetowych, jest odpowiedni sposób odnoszenia się do innych plików lub folderów. Istnieją dwa główne rozwiązania: ścieżki **pełne** lub **względne**. Ścieżka pełna rozpoczyna się od ściśle określonego i niezmiennego punktu, na przykład głównego adresu URL witryny. W HTML ścieżki pełne zawsze rozpoczynają się od *http://domena* lub samego */* (zastąp ciąg *domena* rzeczywistą nazwą domeny, czyli na przykład *www.przyklad.pl*). W takiej sytuacji pełna ścieżka do pliku indeksu w korzeniu witryny ma postać *http://domena/index.html* lub po prostu */index.html*. Ścieżka pełna do pliku *plik.js* w folderze *js*, w korzeniu witryny, będzie miała postać *http://domena/js/plik.js* lub */js/plik.js*. Zaletą ścieżki pełnej jest to, że zawsze będzie prawidłowa, niezależnie od miejsca jej użycia w kodzie, ponieważ działa ona identycznie w pliku *index.html* w głównym folderze witryny i w pliku *strona.html* w jednym z podkatalogów.

Ścieżka względna jest zawsze wyliczana względem strony HTML przeprowadzającej wyliczenie i nie zaczyna się od *http://* czy */*. Ścieżkę tę najczęściej rozpoczyna się od nazwy pliku lub folderu. Na przykład *inna.html* jest ścieżką relatywną do pliku *inna.html* znajdującego się w tym samym folderze co aktualny plik. Aby utworzyć ścieżkę względną do pliku znajdującego się w podkatalogu, najpierw wpisuje się nazwę podkatalogu, a następnie nazwę pliku (lub innych podkatalogów), na przykład *js/plik.js*. Niektórzy preferują oznaczanie ścieżek względnych przez rozpoczynanie ich od znaków kropki i ukośnika, które oznaczają aktualny katalog. Innymi słowy, *./inny.html* jest równoznaczne z *inny.html*, a *./js/plik.js* jest równoznaczne z *js/plik.js*. Aby przejść do katalogu powyżej, użyj dwóch znaków kropki. Jeśli *strona.html* znajduje się w podkatalogu i ma odnieść się do pliku *plik.js* w folderze *js*, poprawna ścieżka względna będzie miała postać *../js/plik.js*, czyli najpierw przejdzie jeden folder wyżej, a następnie przejdzie do folderu *js*.

Ścieżki względne trudno jest zapisać poprawnie, ale najczęściej działają one prawidłowo również po przeniesieniu witryny w inne miejsce, a nawet do innej domeny (o ile tylko zostaną zachowane wzajemne relacje plików).

Techniczny powód jest dosyć złożony, ale wynika ze sposobu przetwarzania danych umieszczonych w elemencie `script` przez XHTML. Ponieważ jednak `[CDATA[]]` jest identyfikatorem dla parsera, a nie kodem JavaScript, zarówno znacznik otwierający `<![CDATA[`, jak i zamykający `]]>` muszą zostać poprzedzone komentarzem JavaScript (`//`). Przedstawione rozwiązanie dotyczy tylko i wyłącznie sytuacji, w których korzysta się z wersji XHTML i umieszcza kod JavaScript wewnątrz elementów `script`. Jest to rozwiązanie niezalecane dla HTML (włączając HTML5), a także w sytuacji, gdy kod JavaScript znajduje się w zewnętrznym pliku. Wspominam o tym, byś nie był zdziwiony, gdy zobaczysz tego rodzaju fragment w kodzie innych osób.

Po trzecie, element `script` bardzo często umieszcza się w elemencie `head` HTML, ale nie jest to wymóg. Co więcej, obecnie wielu programistów zaleca wstawianie elementów `script` pod koniec kodu strony HTML. Yahoo! zaleca umieszczanie ich tuż przed znacznikiem zamykającym `body`. Głównym argumentem jest chęć wywołania **wrażenia** szybszego wczytywania strony WWW. Wynika to z tego, iż przeglądarka internetowa po napotkaniu znacznika `script` od razu przystępuje do pobierania kodu (jeśli nie znajduje się on w jej pamięci podręcznej). Przeglądarka zawiesi dalsze przetwarzanie kodu HTML aż do momentu pobrania i wykonania wskazanego skryptu.

Po czwarte, staraj się nie używać na jednej stronie WWW zbyt wielu zewnętrznie wczytywanych skryptów, bo zaszkodzi to wydajności witryny.

PODSTAWOWE PODEJŚCIA DO PROGRAMOWANIA

Zanim przejdziemy do właściwego kodu, warto przyjrzeć się w szczegółach trzem podejściom do programowania witryn internetowych. To, które podejście wybierzesz — a można wybrać więcej niż jedno — wpływa nie tylko na sposób pisania kodu, ale także na komfort internautów odwiedzających witrynę.

PRZYJAZNA DEGRADACJA

Poza znacznikiem `script`, który dodaje kod JavaScript do strony HTML, istnieje znacznik stanowiący jego przeciwieństwo — `noscript`. Służy on do przedstawienia stosownego komunikatu lub użycia alternatywnej treści, jeśli przeglądarka nie obsługuje kodu JavaScript:

```
<noscript>Twoja przeglądarka nie obsługuje kodu JavaScript!</noscript>
```

Wszystko, co zostanie umieszczone między znacznikami, nie powinno stosować kodu JavaScript, czyli dopuszczalny jest jedynie tekst lub kod HTML.

Poszczególne statystyki różnią się pod tym względem, ale mniej więcej 1 – 3 procent wszystkich użytkowników korzystających z witryn internetowych nie ma włączonej z takich lub innych powodów obsługi języka JavaScript. Dotyczy to osób, które:

- celowo wyłączyły obsługę języka JavaScript w przeglądarce internetowej,
- korzystają z rozszerzeń typu NoScript (*http://noscript.net/*), które domyślnie wyłączają obsługę JavaScriptu na witrynie, chyba że użytkownik dołączy witrynę do tak zwanej białej listy,
- korzystają z odczytywania tekstu stron WWW na głos (osoby niedowidzące),
- korzystają ze starszych przeglądarek w urządzeniach przenośnych lub konsolach do gier,
- używają narzędzi wiersza poleceń — narzędzia te nie analizują pobranego kodu JavaScript (użytkownicy ci wpisują w wierszu poleceń na przykład polecenia `wget` lub `curl`),

Ponadto możemy mieć do czynienia nie z ludźmi, ale z robotami, na przykład robotami wyszukiwarek. To niewielki odsetek całego rynku, ale sam musisz się zastanowić, w jaki sposób najlepiej rozwiązać problem. Istnieją trzy podejścia.

1. Można przyjąć, że nie istnieją użytkownicy mający wyłączoną obsługę JavaScriptu.
2. Można zastosować **przyjazną degradację**.
3. Można wprowadzić **stopniowe ulepszanie**.

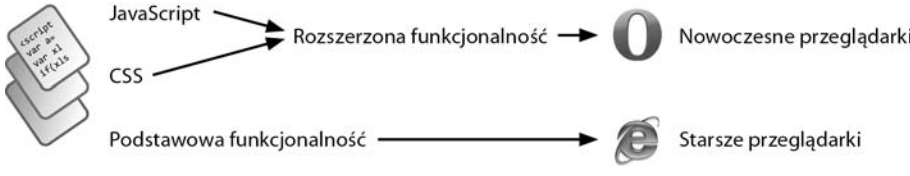
Choć oczywiście sam będziesz wybierał podejście, to jednak pierwsze rozwiązanie, w szczególności przy sporym wzroście liczby użytkowników korzystających z urządzeń przenośnych, może nie być najszczęśliwsze. Mimo to istnieje spora grupa programistów, którym nie mieści się w głowie, że ktoś mógłby nie posiadać włączonej obsługi języka JavaScript. W efekcie witryny tego typu nie działają prawidłowo, a odwiedzający nie wie nawet, co jest tego przyczyną. Oczywiście istnieją słuszne powody, dla których witryna może wymagać obsługi języka JavaScript, ale odwiedzający powinni zostać o tym fakcie poinformowani. Brak tego rodzaju informacji zniechęca odwiedzających do witryny i źle świadczy o programiście (lub firmie, do której należy witryna).

Przez lata najpopularniejszym rozwiązaniem było podejście drugie i nadal pojawia się stosunkowo często. **Przyjazna degradacja** to taktyka, w której wykonuje się wersję z pełną funkcjonalnością, a następnie zapewnia się alternatywny interfejs lub tylko komunikat informujący o potrzebie użycia kodu JavaScript w przypadku wykrycia urządzenia, które nie może skorzystać z pełnej wersji witryny. Brzmi to znajomo? Tak, to efekt użycia znacznika `noscript`. Przyjazna degradacja to znaczący postęp w stosunku do sytuacji, w której po prostu ignoruje się problem. Główna różnica polega na tym, że informuje się odwiedzającego o istnieniu problemu i wskazuje sposób jego rozwiązania (na przykład włączenie obsługi języka JavaScript lub użycie innej przeglądarki).

Jeszcze lepszym rozwiązaniem okazuje się **stopniowe ulepszanie**.

STOPNIOWE ULEPSZANIE

Stopniowe ulepszanie to termin, który został wprowadzony w 2003 roku, ale jego upowszechnianie trwa do dnia dzisiejszego. Stopniowe ulepszanie to podejście będące odwrotnością przyjaznej degradacji. Przyjazna degradacja zaczyna od pełnej funkcjonalności witryny i oferuje alternatywę, jeśli pełna wersja nie jest obsługiwana, natomiast stopniowe ulepszanie rozpoczyna od podstawowej funkcjonalności, a następnie rozbudowuje ją, dodając wygodniejsze sposoby obsługi do tych systemów, które je obsługują (rysunek 2.7). Stopniowe ulepszanie nie tylko zapewnia, że wszyscy klienci będą w stanie korzystać z witryny — osobiście uważam nawet, że takie podejście ułatwia programowanie.



RYSUNEK 2.7. Stopniowe ulepszanie dodaje dynamiczne elementy do podstawowej funkcjonalności

Stopniowe ulepszanie dotyczy nie tylko kodu JavaScript, ale również stylów CSS. Tematowi temu poświęcono nawet całe książki (*Tworzenie stron metodą stopniowego ulepszania. Witryny dostępne dla każdego*, Helion 2010) i nie mogę poświęcić mu tutaj zbyt dużo miejsca, ale cały proces jest znacznie prostszy, niż mogłoby się wydawać.

Zacznijmy od tego, że należy napisać zgodny ze standardami, dobrze ustrukturyzowany i poprawny semantycznie kod HTML. **Semantyczny HTML** wykorzystuje znaczniki HTML, by jawnie wskazać **znaczenie** treści, a nie sposób, w jaki ma zostać ona przedstawiona. Prosty przykład — zamiast znacznika `i` oznaczającego kursywę używaj znacznika `em` wskazującego na istotny element tekstu. Choć różnica wydaje się minimalna, w przypadku `em` przeglądarka internetowa nie musi

stosować dodatkowych stylów (choć domyślnie tak właśnie jest). Skoro już jesteśmy przy semantycznym HTML, cała definicja warstwy prezentacyjnej trafia do CSS, gdzie powinna się znaleźć. W sytuacji, gdy nie istnieją znaczniki wskazujące znaczenie elementu, używa się klas. Trzeba powiedzieć, że właśnie z powodu semantycznych klas w HTML5 wprowadzono nowe elementy, takie jak `footer`, `header` i `nav`.

Wykonaną poprawnie semantycznie stronę HTML warto poddać walidacji, by sprawdzić, czy nie zawiera błędów i czy nie spowoduje włączenia przez przeglądarkę trybu *quirks*. Upewnij się również, że kod HTML i podstawowe style CSS zapewniają poprawny wygląd strony we wszystkich docelowych przeglądarkach. Dopiero teraz można rozpocząć dodawanie ulepszeń, które mogą nie być dostępne dla wszystkich odwiedzających witrynę. Przykład takiego podejścia został zaprezentowany w formularzu rejestracyjnym z rozdziału 1.

Podstawowa funkcjonalność tego formularza, a także wszystkich innych, wykorzystuje przesyłanie danych bezpośrednio do skryptu na serwerze. Serwer dokonuje sprawdzenia danych i odpowiednio reaguje. W przypadku formularza rejestracyjnego brak błędów oznacza dodanie użytkownika do bazy danych, a wystąpienie błędów — ponowne wyświetlenie formularza z prośbą o ich poprawienie (rysunek 1.3). Pierwszym krokiem stopniowego ulepszania byłoby wprowadzenie dodatkowych stylów CSS i rozbudowanie formularza o semantyczne pola tekstowe bez użycia kodu JavaScript. Wydaje mi się, że to podejście jest prostsze i wygodniejsze — przed przystąpieniem do bardziej rozbudowanych rozwiązań (Ajax), które trudniej testować, najpierw wykorzystujemy i sprawdzamy rozwiązanie podstawowe.

Ostatni krok polega na dodaniu kodu JavaScript i CSS, który wprowadzi dodatkowe funkcjonalności, ale tylko wtedy, gdy obsługuje je przeglądarka użytkownika. Oczywiście niniejsza książka obejmuje przede wszystkim kod JavaScript. Do sprawdzenia obsługi niezbędnych funkcji programista kodu JavaScript wykorzystuje **wykrywanie obiektów** omówione w rozdziale 1. Dzięki temu powstaje kod JavaScript działający prawidłowo niezależnie od wersji i typu przeglądarki internetowej. Wykrywanie obiektów jest wyjątkowo proste — jeśli przeglądarka zawiera metodę lub obiekt X, zakładamy, że możemy skorzystać z funkcjonalności X. Przykład takiej detekcji pojawi się kilka stron dalej.

Dzięki uważnie przygotowanemu procesowi rozbudowy lepsze przeglądarki uzyskają lepszą funkcjonalność. To rozwiązanie typu „zjeść ciastko i mieć ciastko”!

Nieinwazyjny JavaScript

Zanim przejdziemy do właściwego kodu (najwyższy czas, prawda?), wprowadzę jeszcze jedną koncepcję — **nieinwazyjny JavaScript**. Dawniej JavaScript bardzo często pojawiał się w wielu miejscach w kodzie HTML. Przykładem może być chociażby wywołanie funkcji w momencie kliknięcia łącza:

```
<a href="javascript:createWindow();">Łącze</a>
```

Bardzo podobne rozwiązanie było stosowane także w przypadku wysyłania formularza:

```
<form action="strona.php" method="post" onsubmit="return validateForm();">
```

Oba przykłady działają prawidłowo również dziś, ale nie jest to zalecane podejście. Po pierwsze, osadzanie kodu JavaScript na stronie HTML czyni ją znacznie trudniejszą w analizie, nie wspominając nawet o przyszłej konserwacji. Przebijanie się przez wiele wierszy kodu HTML, by zmienić jeden wiersz kodu JavaScript, nie jest praktyczne. Po drugie, przedstawione rozwiązanie na trzy sposoby łamie zasadę stopniowego ulepszania:

- kod HTML z dołączonym kodem JavaScript nie jest już czystą semantyką,
- zakłada, iż przeglądarka potrafi wykonać kod JavaScript,
- bardzo trudno zastosować w tym przypadku technikę wykrywania obiektów i dostosowywania kodu do możliwości przeglądarki.

Zasada jest prosta — umieszczaj cały kod JavaScript w znacznikach `script` lub w osobnych plikach.

■ **UWAGA:** Unikaj stosowania w HTML-u łączy bez żadnego znaczenia (zawierających jako adres # lub wywołanie funkcji JavaScript), bo nie będą działały w przeglądarce bez obsługi języka JavaScript.

ŁĄCZYMY KOD

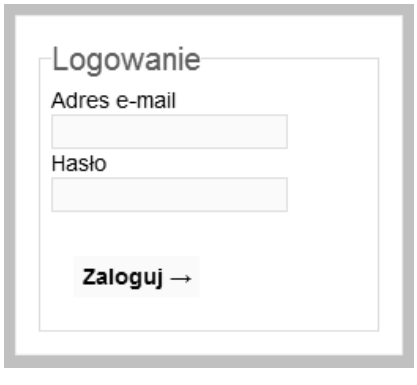
Po omówieniu kilku podstawowych tematów najwyższy czas rozpocząć tworzenie kodu JavaScript. Zakładam, że nie znasz tego języka (przecież czytasz niniejszą książkę po to, by go poznać); posługując się krótkim przykładem, postaram się wykazać, że JavaScript nie jest niczym trudnym, a dodatkowo zapewnię sensowny kontekst dla treści z drugiej części książki.

WYKONANIE PODSTAWOWEJ FUNKCJONALNOŚCI

Jako prosty i praktyczny przykład wykonamy formularz logowania, który zostanie sprawdzony przy użyciu kodu JavaScript. Z dalszych rozdziałów dowiesz się, jak dodać do formularza obsługę technologii Ajax. Na razie pominiemy tę technologię, bo jest zbyt złożona jak na jeden z pierwszych rozdziałów książki.

Na początek utwórz formularz HTML, który będzie składał się z trzech elementów: adresu e-mail, hasła i przycisku wysyłania. Najważniejszy fragment kodu HTML znajduje się poniżej. Znajdziesz go w pliku *logowanie.html* (rysunek 2.8):

```
<form action="logowanie.php" method="post" id="loginForm">
   <fieldset>
      <legend>Logowanie</legend>
      <div><label for="email">Adres e-mail</label><input type="email" name="email"
      ↪id="email" required></div>
      <div><label for="password">Hasło</label><input type="password" name="password"
      ↪="password" required></div>
      <div><label for="submit"></label><input type="submit" value="Zaloguj &rarr;"
      ↪id="submit"></div>
   </fieldset>
</form>
<script src="js/login.js"></script>
```



RYSUNEK 2.8. Formularz logowania z dodanymi kilkoma stylami CSS

■ **WSKAZÓWKA:** Kod wszystkich przykładów prezentowanych w książce znajdziesz pod adresem *ftp://ftp.helion.pl/przyklady/troyaid.zip*.

Dla uproszczenia strona WWW zawiera tylko i wyłącznie formularz. By używanie formularza było nieco uprzyjemnione, strona korzysta z pliku CSS z kilkoma prostymi stylami. Cały kod możesz pobrać pod adresem podanym we wskazówce (plik CSS znajdziesz w plikach w folderze *Rozdzial02*).

W obecnej postaci dane z formularza trafią do skryptu *logowanie.php*. Skrypt ten:

- sprawdziłby poprawność podanego adresu e-mail,
- sprawdziłby podanie hasła,
- sprawdziłby, czy przekazane wartości zgadzają się z wcześniej umieszczonymi w bazie danych,
- w przypadku poprawności danych rozpocząłby nową sesję dla zalogowanego użytkownika,
- przekierowałby użytkownika do strony powitalnej.

W dalszych rozdziałach wykonamy również takie sprawdzenia (być może nie wiesz, jak wykonać odpowiedni kod PHP i MySQL). To podstawowa funkcjonalność, która będzie działała we wszystkich przeglądarkach internetowych niezależnie od tego, czy obsługa kodu JavaScript jest w nich włączona, czy wyłączona. Jeśli przeglądarka obsługuje formularze, wszystko zadziała prawidłowo. W następnym kroku zajmiemy się stopniową rozbudową.

Dodanie warstwy kodu JavaScript

W tym konkretnym przypadku stopniowe ulepszanie będzie polegało na wykorzystaniu kodu JavaScript do sprawdzenia poprawności formularza przed jego wysłaniem, dzięki czemu do serwera będzie trafiała najprawdopodobniej prawidłowa postać danych, a sam użytkownik będzie natychmiast informowany o zauważonych błędach (patrz przykład rejestracji z rysunku 1.4).

Zauważ, że jedyną różnicą w samym formularzu w porównaniu z wersją, która nie używa kodu JavaScript, jest to, że poza wymaganymi atrybutami `name` pojawiły się jeszcze atrybuty `id`. Wartości `name` służą jako nazwy poszczególnych danych i są przesyłane do serwera. Wartości `id` będą wykorzystywane tylko i wyłącznie przez kod JavaScript. W prezentowanym kodzie oba atrybuty mają tę samą wartość, co nie było trudne do uzyskania, ponieważ strona zawiera tylko jeden prosty formularz (na stronie HTML nie mogą wystąpić dwa elementy o takiej samej wartości atrybutu `id`).

Strona WWW ze stopniowym ulepszeniem korzysta z zewnętrznego pliku JavaScript o nazwie *login.js*. Został on dołączony do strony HTML tuż przed znacznikiem zamykającym `body`:

```
<script src="js/login.js"></script>
```

Teraz nadchodzi moment, w którym sprawy zaczynają się nieco komplikować, w szczególności na tak wczesnym etapie książki. Aby wiedzieć, jaki kod powinien znaleźć się w pliku, trzeba posiadać przynajmniej podstawową wiedzę na temat obsługi zdarzeń.

Obsługa zdarzeń

Jak wspomniałem w rozdziale 1., JavaScript jest językiem sterowanym zdarzeniami, czyli wykonuje operacje dopiero po zajściu określonego zdarzenia. Przykładami zdarzeń są:

- wczytanie strony WWW,
- kliknięcie elementu, na przykład łącza lub przycisku,
- wpisanie tekstu w polu tekstowym formularza,

- przejście kursorem myszy nad element (zdarzenie `mouseover`),
- przejście kursorem myszy poza element (zdarzenie `mouseout`).

Aby kod JavaScript przeprowadził walidację formularza, musimy najpierw zdecydować, na które zdarzenie będziemy reagowali. Zdarzeniami najczęściej wykorzystywanymi do walidacji formularzy są:

- zdarzenie wysyłania formularza,
- kliknięcie przycisku wysyłania (który również wywoła zdarzenie wysyłania),
- zmiana wartości elementu formularza,
- sytuacja, gdy element formularza utraci aktywność (element wywoływany niezależnie od tego, czy jego wartość uległa zmianie, czy też nie).

Rozdział 8. znacznie dokładniej omawia istniejące rodzaje zdarzeń. Na razie wybierzmy jedno z prostszych rozwiązań, czyli walidację w momencie próby wysłania formularza. W tym celu musimy dodać do formularza **procedurę obsługi zdarzenia**. Procedura obsługi zdarzenia wskazuje, że w momencie zajścia **określonego zdarzenia** na **konkretnym elemencie** należy wywołać **wskazaną funkcję**. Każdy obiekt, niezależnie od tego, czy jest to całe okno przeglądarki, czy tylko niewielki element strony (niekoniecznie pole formularza), może zgłosić zdarzenie. Najczęściej samodzielnie definiuje się funkcję, która zostanie wywołana. Połączenie „obiekt –typ zdarzenia – funkcja" daje naprawdę sporo możliwości.

Aby móc podłączyć się pod zdarzenie wysyłania formularza, musimy najpierw uzyskać referencję do formularza. Najprostszym rozwiązaniem okazuje się użycie metody `getElementById()` obiektu `document`. Obiekt `document` dotyczy całego dokumentu HTML — od znacznika otwierającego `html`, poprzez znaczniki `head` i `body`. Obiekt `document` zawiera metodę `getElementById()`, która jako argument przyjmuje nazwę identyfikatora, a w zamian zwraca referencję do elementu (o ile istnieje). Uzyskaną wartość można przechować w zmiennej w celu późniejszego użycia:

```
var loginForm = document.getElementById('loginForm');
```

W tym momencie, o ile tylko istnieje jeden element (dowolnego typu) o atrybucie `id` z wartością `loginForm`, zmienna `loginForm` będzie zawierała referencję do tego elementu. Rozdział 9. znacznie dokładniej omawia obsługę DOM, ale metoda `getElementById()` jest tak ważna i prosta w użyciu, że można ją wprowadzić już w rozdziale 2.

Po uzyskaniu referencji do formularza procedurę obsługi zdarzenia ustalimy w następujący sposób:

```
element.onzdarzenie = funkcja;
```

W naszym przypadku będzie to:

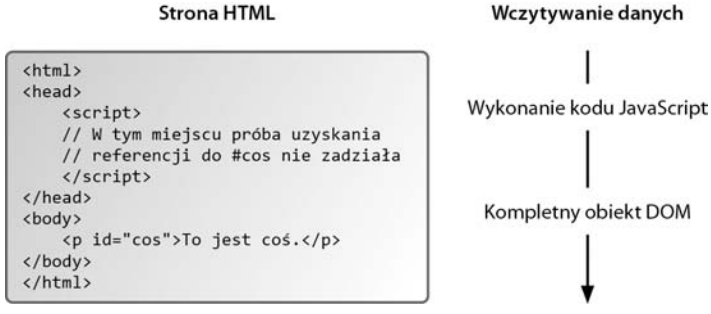
```
loginForm.onsubmit = validateForm;
```

Ramka w dalszej części dokładniej przedstawia szczegóły techniczne składni. Wystarczy powiedzieć, że gdy element `loginForm` będzie zamierzał wysłać dane formularza do serwera, wcześniej wywoła funkcję `validateForm()`. Zauważ, że po prawej stronie przypisania została użyta tylko i wyłącznie nazwa funkcji, bez nawiasów klamrowych lub cudzysłowów. Żadna z poniższych składni nie jest prawidłowa:

```
loginForm.onsubmit = 'validateForm'; // NIE!
loginForm.onsubmit = validateForm(); // NIE!
```

W teorii następny krok polegałby na zdefiniowaniu funkcji `validateForm()`, która przeprowadziłaby sprawdzenie wartości elementów formularza. Niestety, konieczny jest jeszcze jeden krok. Już wyjaśniam…

Gdy klient pobiera dokument z serwera, uzyskuje dane w odpowiedniej kolejności. W przypadku strony HTML oznacza to, że przeglądarka dosłownie najpierw otrzyma `DOCTYPE`, następnie nawias otwierający `html`, znacznik `head` i treść nagłówka, a dopiero później element `body` i jego zawartość, i tak aż do samego końca dokumentu. Gdy przeglądarka znajdzie odnośnik do innych materiałów, które muszą zostać pobrane — plików CSS, obrazów, materiałów Flash, kodu JavaScript itp. — je również musi pobrać. W kwestii dostępu do dokumentu DOM proces ten jest niezwykle istotny, ponieważ przeglądarka nie zna pełnej reprezentacji DOM, dopóki nie przetworzy całej zawartości strony HTML (rysunek 2.9). Oznacza to, że w języku JavaScript nie można bezpiecznie użyć metody `document.getElementById()` do momentu pełnego wczytania strony.



RYSUNEK 2.9. Przeglądarka wczytuje dokument HTML i tworzy obiekt DOM

Tym, co pozwala upewnić się, że można bezpiecznie korzystać z elementów DOM, jest przede wszystkim pojawienie się zdarzenia ukończenia wczytywania strony WWW. Ponieważ jest to zdarzenie, możemy dodać procedurę jego obsługi:

```
window.onload = init;
```

■ **WSKAZÓWKA:** Powodem, dla którego witryny wydają się wczytywać szybciej, gdy kod JavaScript umieści się na końcu dokumentu, jest to, że przeglądarka nie musi zatrzymywać procesu renderowania strony w oczekiwaniu na pobranie i wykonanie kodu JavaScript.

■ **UWAGA:** Celowo uprościłem w opisie pewne szczegóły dotyczące pobierania i wykonywania elementów strony, by nie zaciemnić ogólnej idei. Jeśli jesteś zainteresowany tym tematem, z pewnością znajdziesz w internecie wiele materiałów uzupełniających.

Kod informuje, że w momencie zakończenia wczytania strony HTML należy wywołać funkcję `init()`. To w jej wnętrzu możemy dodać procedurę obsługi zdarzenia do formularza, bo operacje na obiekcie DOM będą już bezpieczne:

```
function init() {
   var loginForm = document.getElementById('loginForm');
   loginForm.onsubmit = validateForm;
}
```

Rozdział 7. zawiera wszystkie niezbędne informacje związane z tworzeniem funkcji, ale podstawy są naprawdę bardzo proste. Najpierw pojawia się słowo kluczowe `function`, a po nim nazwa funkcji i nawiasy okrągłe. (Bardzo często funkcjom inicjalizującym nadaje się nazwę `init`, bo jest krótka i informuje o przeznaczeniu funkcji). Rzeczywisty kod funkcji, czyli kod wykonywany w momencie jej wywołania, znajduje się między nawiasami klamrowymi.

Właściwości zdarzeń obiektu

Jak wspomniałem w rozdziale 1., obiekt jest specjalnym typem, który może zawierać predefiniowane **atrybuty** (własne zmienne) oraz **metody** (własne funkcje). Dostęp do atrybutów i metod zapewnia notacja kropkowa. Kod `loginForm.onsubmit = validateForm` oznacza, że funkcja `validateForm()` zostaje przypisana do właściwości `onsubmit` obiektu `loginForm`. Choć początkowo wydaje się to dziwne, w rzeczywistości działa to dokładnie tak samo jak w przypadku przypisania do zmiennej zwykłej wartości liczbowej:

```
var num = 2;
```

W przypadku obiektu zmienna jest atrybutem tego obiektu, a przypisywaną wartością jest funkcja — nieco bardziej skomplikowane, ale zasada pozostaje taka sama.

Obiekt `loginForm` zawiera właściwość `onsubmit`, ponieważ reprezentuje on formularz, a formularze zgłaszają zdarzenie wysyłania danych. Ten sam kod nie działałby prawidłowo w przypadku łączy, ponieważ te nie zgłaszają zdarzenia `onsubmit` (choć zgłaszają zdarzenie `onclick`). Odnosząc się do zdarzeń jako właściwości, zawsze używaj małych liter, czyli pisz `onsubmit` zamiast `onSubmit`.

Jeśli chodzi o samo przypisanie, ze zdarzeniem musi być powiązana funkcja, więc z prawej strony przypisania pojawiła się jej **nazwa**. Nie umieszczaj nazwy w cudzysłowach, ponieważ wtedy przypisanie będzie dotyczyć wartości tekstowej, a nie funkcji. Nie stosuj również na końcu nawiasów okrągłych, bo wykonasz funkcję, zamiast ją przypisać.

Jako dodatkowe zabezpieczenie dodajmy **wykrywanie funkcjonalności**, by procedura obsługi zdarzenia została przypisana tylko wtedy, gdy przeglądarka rzeczywiście posiada metodę `document.getElementById()`:

```
function init() {
   if (document && document.getElementById) {
      var loginForm = document.getElementById('loginForm');
      loginForm.onsubmit = validateForm;
   }
}
```

W tym momencie istnieją dwie procedury obsługi zdarzeń. Pierwsza nasłuchuje zdarzenia zakończenia wczytywania (`load`); zdarzenie to naturalnie zostanie zgłoszone tylko jeden raz. Wykona ono kod znajdujący się w funkcji `init()`. Druga procedura nasłuchuje rozpoczynania wysyłania danych z formularza i może zostać wykonana dowolną liczbę razy (lub może nie być wykonana nigdy). Każde zajście zdarzenia wykona kod znajdujący się w funkcji `validateForm()`. Określenie zawartości tej funkcji będzie ostatnim krokiem rozbudowy.

UWAGA: W rzeczywistości przeglądarki obsługują obiekt `document` i metodę `getElementById()` od ponad dekady, więc nie ma potrzeby przeprowadzania tej konkretnej detekcji obiektu.

Przeprowadzanie walidacji

Funkcja `validateForm()` powinna sprawdzić zawartość formularza i zwrócić wartość logiczną, która wskaże, czy dane są poprawne. Jeśli funkcja zwróci wartość `true`, wysłanie danych z formularza do serwera zostanie przeprowadzone. Zwrócenie wartości `false` zablokuje wysyłkę i przeglądarka nie skomunikuje się z serwerem.

Definicja funkcji bez treści wygląda następująco:

```
function validateForm() {
}
```

Czas na przeprowadzenie prostej walidacji, która pojawi się wewnątrz funkcji. W przypadku adresu e-mail i hasła należałoby sprawdzić, czy użytkownik wprowadził **jakąkolwiek wartość** (co prawda można jeszcze sprawdzić poprawność adresu e-mail, ale wymaga to znacznie bardziej złożonego kodu). Dla pól tekstowych prosta walidacja polega na sprawdzeniu długości znajdujących się w nich wartości (jeśli coś sprawdzono, będzie większa od zera). Na początek będzie potrzebna jednak referencja do każdego z pól uzyskana dzięki metodzie `getElementById()`:

```
var email = document.getElementById('email');
var password = document.getElementById('password');
```

■ **WSKAZÓWKA:** Pamiętaj, że strona WWW korzysta z języka HTML5, więc użycie przeglądarki z obsługą HTML5 spowoduje automatyczne zastosowanie walidacji po stronie klienta (patrz wcześniejsze rysunki).

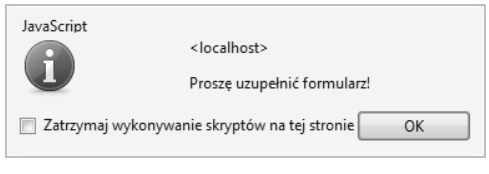
W tym momencie każda zmienna zawiera referencję do odpowiadającego jej elementu. Aby uzyskać wartość wpisaną w polu tekstowym, wykorzystuje się właściwość `value`: `email.value` i `password.value`. Ponieważ obie wartości są typu tekstowego, właściwość `value` będzie obiektem tekstowym (nawet w przypadku pustego tekstu). Wszystkie teksty w języku JavaScript posiadają właściwość `length`, informującą o liczbie znaków, z których składa się tekst. Oznacza to, że `email.value.length` będzie liczbą znaków wpisanych w polu adresu e-mail. Możemy więc pokusić się o napisanie bardzo prostej instrukcji warunkowej:

```
if ( (email.value.length > 0) && (password.value.length > 0) ) {
   return true;
} else {
   return false;
}
```

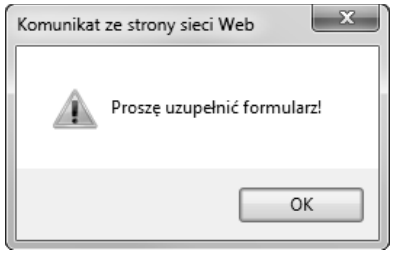
■ **UWAGA:** Pamiętaj, że sprawdzanie długości wartości elementu działa prawidłowo jedynie dla pól tekstowych; inne rodzaje pól formularza wymagają innego sposobu walidacji.

W zasadzie ukończyliśmy prostą walidację. Jeżeli użytkownik nic nie wpisze w formularzu, nie zostanie on wysłany do serwera. Poza samą blokadą wysyłki warto również poinformować użytkownika o zaistniałym problemie. Istnieje kilka profesjonalnych sposobów, ale na razie będzie musiał wystarczyć zwykły komunikat w okienku (rysunki 2.10 i 2.11):

```
if ( (email.value.length > 0) && (password.value.length > 0) ) {
   return true;
} else {
   alert('Proszę uzupełnić formularz!');
   return false;
}
```



RYSUNEK 2.10. Okno z komunikatem w przeglądarce Opera



RYSUNEK 2.11. To samo okno z komunikatem (rysunek 2.10) w przeglądarce Internet Explorer

---

■ **UWAGA:** Walidacja po stronie klienta ma za zadanie ułatwić życie użytkownikowi. Walidacja po stronie serwera nadal jest niezbędna.

Uzyskaliśmy prosty, stopniowo ulepszany i nieinwazyjny kod JavaScript, który sprawdza poprawność formularza przed wysłaniem danych do serwera. Przedstawiony w dalszej części rozdziału kod zawiera wszystkie omówione wcześniej fragmenty wraz z dodatkowym komentarzem. Skrypt są na głównym poziomie (bez zagnieżdżeń) trzy elementy:

- definicja funkcji `validateForm()`,
- definicja funkcji `init()`,
- zgłoszenie funkcji `init()` jako procedury obsługi dla zdarzenia `window.onload`.

■ **UWAGA:** Ponieważ nie powstał jeszcze skrypt *logowanie.php*, podanie prawidłowych danych w formularzu spowoduje zgłoszenie błędu po stronie serwera wskutek niemożności znalezienia wymaganego pliku.

WYWOŁANIE TRYBU ŚCISŁEGO

JavaScript również posiada tryb ścisły, który różni się od omawianego wcześniej trybu ścisłego przeglądarki internetowej. Umożliwia on wymuszenie bardziej restrykcyjnej obsługi języka JavaScript. Tryb ten został wprowadzony w ECMAScript 5. Wywołuje się go, umieszczając jako pierwszy wiersz kodu następujący tekst:

```
'use strict';
```

Wiersz ten można umieścić albo na początku każdego skryptu, albo jako pierwszy wiersz kodu wszystkich funkcji. W niniejszej książce będzie się pojawiać to drugie rozwiązanie.

Włączenie trybu ścisłego wprowadzi pewne subtelne zmiany w działaniu kodu JavaScript w porównaniu z trybem domyślnym. Można powiedzieć, że tryb ścisły:

✓ spowoduje generowanie błędów przez potencjalnie problematyczny kod,
✓ poprawi bezpieczeństwo i wydajność,
✓ ostrzeże o korzystaniu z rozwiązań, które zostaną wycofane w następnej wersji języka.

Tryb ścisły zmusza zatem do pisania lepszego kodu, co jest godne uwagi.

Bardziej szczegółowy opis zmian wprowadzanych w obsłudze języka przez tryb ścisły znajdziesz na wielu stronach internetowych, ale na tym poziomie znajomości języka prawdopodobnie niewiele z tego zrozumiesz.

Choć nie ma technicznych przeciwwskazań, by zapisać kod w innej kolejności, wybrałem kolejność najbardziej logiczną, czyli:

- funkcja `validateForm()` zostaje zdefiniowana przed jej użyciem w funkcji `init()`,
- funkcja `init()` zostaje zdefiniowana przed jej przypisaniem do właściwości `window.onload`.

Kod będzie działał prawidłowo nawet po zmianie kolejności, ale zastosowana tutaj kolejność ma z logicznego punktu widzenia największy sens. Każda funkcja rozpoczyna się od następującego wiersza:

```
'use strict';
```

Powody jej użycia zostały wyjaśnione w ramce „Wywołanie trybu ścisłego".

```
// login.js
// Funkcja wywoływana w momencie wysyłania formularza.
// Funkcja sprawdza dane formularza i zwraca wartość logiczną.
function validateForm() {
   'use strict';

   // Pobierz referencje do pól formularza:
   var email = document.getElementById('email');
   var password = document.getElementById('password');

   // Walidacja!
   if ( (email.value.length > 0) && (password.value.length > 0) ) {
      return true;
   } else {
      alert('Proszę uzupełnić formularz!');
      return false;
   }

} // Koniec funkcji validateForm().

// Funkcja wykonywana po zakończeniu wczytywania strony WWW.
// Funkcja dodaje procedurę obsługi zdarzenia do formularza.
function init() {
   'use strict';

   // Potwierdź, że można użyć document.getElementById():
   if (document && document.getElementById) {
      var loginForm = document.getElementById('loginForm');
      loginForm.onsubmit = validateForm;
   }

} // Koniec funkcji init().

// Przypisz procedurę obsługi zdarzenia do okna przeglądarki:
window.onload = init;
```

PODKRADNIJ TEN KOD JAVASCRIPT

Jak wspomniałem we wprowadzeniu do tego rozdziału, fakt, iż możesz używać języka JavaScript bez jego prawdziwej znajomości, jest zarówno wybawieniem, jak i przekleństwem. Jeśli pisałeś kod JavaScript w jakimś projekcie i nie do końca wiedziałeś, co robisz, nie martw się — wielu programistów czyniło podobnie, nawet ja. Mam nadzieję, że zawsze uda Ci się wykonać powierzone zadanie.

Istnieje jednak spore prawdopodobieństwo, że powstały kod nie będzie optymalny lub nie będzie działał w niektórych sytuacjach. Właśnie dlatego czytasz niniejszą książkę — by lepiej poznać język.

Pod koniec rozdziału jedno zalecenie — proponuję, byś wyrobił sobie w trakcie nauki nawyk zaglądania do kodu JavaScript dostępnego na różnych witrynach. Nie mam tutaj na myśli ćwiczeń i dokumentacji, ale kod odwiedzanych witryn, który w większości przypadków można bez ograniczeń czytać w przeglądarce internetowej. Podobnie jak w przypadku kodu HTML i obrazów, również i kodu źródłowego JavaScript nie można zablokować przed przeglądaniem.

Przeglądaj kod napisany przez inne osoby, ale oczywiście nie kradnij go (tytuł miał przykuwać uwagę, czego tytuł „Przeglądaj kod JavaScript" z pewnością by nie zapewnił). Wykorzystaj go do własnej edukacji. Z pewnością od czasu do czasu natkniesz się na niezrozumiały fragment lub kod pisany dawno temu, który stoi w sprzeczności z zasadami proponowanymi w niniejszej książce. Poznając sposób tworzenia kodu przez inne osoby, zaznajamiasz się z zakresem, możliwościami i historią języka. Jeśli natrafisz na podejście nietypowe lub odmienne od proponowanych, zapisz je. Po przeczytaniu książki sprawdź, czy znalazłeś rozwiązanie problemu.

■ **UWAGA:** Nie powinieneś kraść kodu JavaScript znajdującego się na innych witrynach nie tylko z powodów moralnych, ale również z racji tego, że może on zawierać błędy lub zależności mogące zagrozić witrynie.

PODSUMOWANIE

Rozdział 1. był bardzo ogólnym wprowadzeniem do języka JavaScript, natomiast rozdział 2. delikatnie przybliżył rzeczywisty kod i jego implementację. Dowiedziałeś się z niego o:

- wpisie `DOCTYPE` i trybach przeglądarek,
- HTML5 oraz nowych elementach i atrybutach formularzy,
- osadzaniu kodu JavaScript na stronach WWW przy użyciu elementu `script`.

Dodatkowo zapoznałeś się z szablonem HTML5, który będzie wykorzystywany we wszystkich przykładach.

Większość rozdziału została poświęcona rzeczywistemu kodowi wykonującemu praktyczne zadanie — sprawdzanie poprawności formularza przed jego wysłaniem do serwera. Pojawiły się podstawowe informacje na temat obsługi zdarzeń, tworzenia własnych funkcji i dostępu do elementów strony za pomocą `document.getElementById()`. Możesz powrócić do tego prostego przykładu, jeśli w kolejnych rozdziałach będziesz mieć wątpliwości dotyczące wykorzystania pewnych podstawowych rozwiązań.

Zalecam, abyś:

- uważał na ścieżki do plików stosowane w kodzie HTML (względne lub pełne),
- pamiętał o dodaniu atrybutu `novalidate` do znacznika otwierającego `form`, jeśli chcesz przeprowadzić własną walidację formularza, zamiast polegać na walidacji HTML5,
- pamiętał o **nieinwazyjnym JavaScript**, **stopniowym ulepszaniu** i **detekcji obiektów**,
- przyglądał się kodowi JavaScript na odwiedzanych witrynach internetowych.

Jeśli jeszcze nie wiesz, w jaki sposób przeglądać kod JavaScript w przeglądarce internetowej, przejdź do następnego rozdziału, w którym wyjaśnię tę kwestię, a także przedstawię wiele ciekawych narzędzi programistycznych.

SKOROWIDZ

A

ActionScript, 28
Ajax, 18, 307, 313, 328, 407
akcje domyślne, 222
aktualna data, 142
alerty, 230
aplikacja
  Adobe Dreamweaver, 61
  Aptana Studio, 62
  Eclipse, 62
  IntelliJ IDEA, 62
  Komodo IDE, 62
  NetBeans, 62
  PhpStorm, 62
  Putty, 59
  Spoon, 68
  WebStorm, 62
aplikacje
  IDE, 62
  RIA, 27
argumenty funkcji, 173
arkusze stylów, 261
asercje, 343, 346
asercje w narzędziu Firebug, 344
atrybut, 49
  action, 276
  autofocus, 38
  id, 47
  lang, 36
  maxlength, 39, 100
  novalidate, 39, 53
  pattern, 39
  placeholder, 39
  required, 39
  src, 40
  type, 40
aukcja, 403, 419
automatyczne
  aktualizowanie notowań, 334
  uzupełnianie, 282

B

baza danych MySQL, 400
biblioteka, 373
  Blackbird, 373
  cURL, 334
  Head JS, 373
  jQuery, 348, 353–367
    moduły dodatkowe, 359
  jsUnity, 345
  MediaElement.js, 373
  Modernizr, 373
  RequireJS, 373
  shiv, 37
  SWFObject, 373
  Video JS, 373
  YUI, 363–367
biblioteki z funkcjami pomocniczymi, 206
blok catch, 342
blokowanie
  akcji domyślnej, 226
  przycisku, 281
  wykonywania programu, 75
błędy, 58, 71, 340
  logiczne, 72, 103
  składniowe, 72, 413
  wykonania, 72

C

CDN, Content Delivery Network, 354
ceny akcji, 335, 371
ciasteczka, 262–267, 412
ciągi znaków, 96
CORS, Cross-Origin Resource Sharing, 334
CSS, Cascading Style Sheets, 26
czas, 142
  lokalny, 169
  uniksowy, 138
  uniwersalny UTC, 143, 398
czujka, 78

D

data, 138
debuger, 14, 58
debugowanie, 73
definiowanie
  funkcji, 172, 185
  testów, 345
  własnego obiektu, 380
  wzorców, 294
deklaracja
  typu dokumentu, 34
  zmiennej, 82
delegacja obsługi zdarzeń, 226
detekcja obiektów, 68
długość wartości elementu, 50
dodatek Firebug, 58, 76–79, 317, 344
dodawanie
  metod prototypów, 385
  zdarzeń, 206
dokumentacja
  JavaScript, 79
  MySQL, 398
  YUI, 364
DOM, Document Object Model, 26, 247, 354, 417
domknięcie, 200, 386, 388
dostęp do
  elementów tablicy, 152, 157
  globalnych właściwości, 242
  właściwości HTML, 291
  właściwości obiektów, 164
  zmiennej, 387
dostępność witryny, 215
drukowanie strony, 245
dynamiczna obsługa typów, 17
dynamiczne generowanie kodu, 27
działanie technologii Ajax, 313
dziedziczenie prototypowe, 165, 384

E

edycja DOM, 58
edytor
  Aptana Studio, 58
  TextMate, 60
  vi, 60
  WYSIWYG, 58
edytory tekstu, 13, 56–61
efekt zanikania, 388
ekran, 235
ekrany dotykowe, 210
element
  article, 37
  nav, 37
  errorDiv, 411
  footer, 37
  header, 37
  link, 37, 267
  loginForm, 47
  script, 37, 41
  section, 37
  span, 277, 279, 418
elementy, 16
  formularzy HTML5, 37
  HTML, 252
  strony, 247
  tablicy, 152

F

faza
  bąbelkowania, 223
  wyłapywania, 225
fazy zdarzeń, 224
Firebug, 58, 76–79, 317, 344
Flash, 27
folder
  htdocs, 398
  includes, 398
format
  JSON, 23, 317, 320
  XML, 18, 319
formatowanie liczb, 92
formularz, 275
  dodawania pracowników, 166
  HTML, 19, 100
  kontaktowy, 128, 328
  logowania, 45
  rejestracyjny, 20, 301
  zadań, 382
  zgłoszenia oferty, 404
framework, 24, 353, 373
  ExtJS, 25
  jQuery, 25, 353–367
  MooTools, 24
  Prototype, 25
  script.aculo.us, 24
  The Dojo Toolkit, 25
  YUI, 24, 363–367
  Zend Framework, 324
funkcja
  $(), 347
  adddErrorMessage(), 277
  addTask(), 387
  alert(), 152, 233
  assert(), 343
  calculate(), 124
  COALESCE(), 400
  confirm(), 233
  Employee, 379
  encodeURIComponent(), 315
  eval(), 270, 321
  getBids(), 415, 418
  getXMLHttpRequestObject(), 342, 410
  handleAjaxResponse(), 329
  handleGetBidsAjaxResponse(), 416
  IF(), 403
  init(), 48, 51, 75, 418
  inspect(), 77
  isFinite(), 127
  isNaN(), 126
  jQuery(), 354
  json_encode(), 324
  log(), 348
  MAX(), 401
  now(), 145
  onclick, 332
  onload, 333
  onreadystatechange, 313, 335, 410

onsubmit, 330
parseFloat(), 104
parseInt(), 104, 172
process(), 128, 148, 202
prompt(), 233
removeErrorMessage(), 278
reportEvent(), 219
setInterval(), 269, 388
setTheme(), 267
setThemeCookie(), 267
setTimeout(), 269
setTimer(), 325
setUp(), 346
SHA1(), 398
submitBid(), 415
tearDown(), 346
toggleSubmit(), 304
UNIX_TIMESTAMP(), 404
updateDuration(), 210
updateMenu(), 285
validateForm(), 47–52, 302, 410, 412
funkcje
anonimowe, 188, 195
czasowe, 269
globalne, 270
jako obiekty, 186
jako wartości, 187
wyrażeń regularnych, 293
zagnieżdżone, 195
zewnętrzne, 387

G

generator obiektów, 379
generowanie
daty i czasu, 138
kodu JavaScript, 413
liczb, 133
globalny obiekt window, 232
głębokie łącza, 244
główna opcja wyboru, 289
GMT, Greenwich Mean Time, 143

H

historia przeglądarki, 241
HTML, HyperText Markup Language, 26
HTML5, 36, 40

I

IDE, 14, 56, 60
identyczność, 114
identyfikator funkcji, 188
IE Developer Toolbar, 68
indeks tablicy, 153
inicjalizacja
obiektu Date, 139
tablicy, 157
zmiennej, 85
inspektor
obiektów, 165
WWW, 197
instrukcja
if-else, 110, 117
if-else if, 118
switch, 119
instrukcje warunkowe, 110
inteligentne uzupełnienia, 57
interwał, 147

J

język
ActionScript, 28
ECMAScript, 29
HTML, 26
HTML5, 35
JavaScript, 25
PHP, 26
języki
obiektowe, 16
skryptowe, 17
jQuery, 353–367
JSON, JavaScript Object Notation, 23
JSON-P, JSON with Padding, 334

K

kalkulator, 89, 95
kaskadowe arkusze stylów, 26
klasa tooltip, 279
klasy, 379
klasy znaków, 298, 299
klauzula
catch, 341
finally, 341
GROUP BY, 401
klawisze, 221
kod nieinwazyjny, 31
kod pliku view.js, 414
kodowanie UTF-8, 36
kody znaków, 221
kolejka
FIFO, 159
LIFO, 159
kolejność wykonywania działań, 87
kolorowanie składni, 56
komentarze, 42, 112
komunikacja
Ajax, 308
między oknami, 240
komunikat o błędzie, 121, 128, 149, 277, 301, 409, 411, 416
konfiguracja jsUnity, 345
konserwacja kodu, 31
konsola YQL, 370
konstrukcja
try...catch, 342
try...finally, 349
konstruktor, 379
kontekst, 192
kontrola typów, 17
konwersja
typów, 103
tekstu na liczbę, 104
kończenie aukcji, 419
korzeń, 247
kursor, 222
kwantyfikatory, 296

L

licytacja, 397
liczba
argumentów, 175
parametrów, 176
liczby, 86
liczby losowe, 133
lista
aukcji, 396, 400
ofert, 406
zadań, 160
listy
dynamiczne, 284
wyboru, 283
wyboru powiązane, 285
literały, 105, 151
logowanie, 402

Ł

łańcuch
  prototypów, 384
  wywołań, 16
łączenie
  tekstów, 107
  wartości, 101, 104

M

menu Programowanie, 318
metoda
  $.ajax(), 358
  addEmployee(), 384
  addEvent(), 207
  addEventListener(), 204, 217
  ajax(), 358
  append(), 316, 366
  appendChild(), 253
  assert(), 345
  attachEvent(), 217, 225
  autocomplete(), 361
  before(), 356
  charAt(), 97
  click(), 291
  close(), 236
  concat(), 101, 159, 175
  console.trace(), 75
  create(), 365
  deleteRule(), 261
  document.writeln(), 246
  document.getElementById(), 48, 261
  document.write(), 246
  every(), 189
  exec(), 294
  fadeIn(), 358
  fadeOut(), 358
  filter(), 189
  forEach(), 189
  fromCharCode(), 221
  getElementById(), 47, 50, 96, 246, 250
  getElementsByName(), 290
  getElementsByTagName(), 250, 319
  getFullYear(), 147
  getMonth(, 154
  getName(), 194
  getTime(), 145
  getTimeZoneOffset(), 144
  go(), 241
  hide(), 358
  indexOf(), 97, 127, 154
  insert(), 366
  insertBefore(), 253
  insertRule(), 261
  io(), 367
  join(), 162
  lastIndexOf(), 97, 154
  log(), 75
  map(), 189
  match(), 294
  moveTo(), 235
  on(), 366
  open(), 237, 310
  parse(), 321
  pop(), 159
  prepend(), 366
  preventDefault(), 222
  push(), 159, 161
  reduce(), 189
  removeEvent(), 207
  replace(), 243, 294
  run(), 346
  search(), 293
  setContent(), 365
  setDate(), 147
  setFullYear(), 147
  setRequestHeader(), 315
  setText(), 207
  show(), 358
  slice(), 97, 98
  some(), 189
  sort(), 188
  splice(), 160
  split(), 163, 294
  substr(), 98
  substring(), 98
  toFixed(), 92, 193
  toISOString(), 141
  toJSON(), 141
  toLowerCase(), 101, 127
  toString(), 105, 381
  toUpperCase(), 101, 127
  toUTCString(), 144
  trim(), 101, 385
  unshift(), 159
  use(), 363
  valueOf(), 381
  window.print(), 245
metody, 16, 49, 163
  DOM, 250
  obiektu Date, 140, 141
  obiektu Math, 94
  tablicy, 158
  zmiany daty, 145
minifikacja kodu, 390
moduł
  autocomplete, 368
  autocomplete-filters, 368
  console, 364
  DataTables, 362
  Flash Player, 28
  profiler, 364
  test, 364
modyfikacja
  CSS, 256
  DOM, 247, 254, 306, 356, 365
  elementów, 355, 365
  tekstu, 100, 102
  wymiarów okna, 234
  zmiennej lokalnej, 184
monitor komunikacji sieciowej, 317
monitoring sieci, 58, 59
MVC, Model, View, Controller, 26

N

narzędzia
  do minifikacji, 392
  do walidacji, 113
narzędzie, *Patrz także* aplikacja
  Dromaeo, 348
  Firebug, 58, 76–79, 317, 344
  JS Bin, 69, 70
  jsFiddle, 71
  jSHint, 74
  JSLint, 74, 348
  W3C Markup Validator, 34
  YQL, 369
nasłuchiwanie zdarzeń, 280
nawiasy
  klamrowe, 163
  kwadratowe, 152, 164
  okrągłe, 145
nazewnictwo procedur, 203
nazwy zmiennych, 84
nieinwazyjny JavaScript, 44
notacja kropkowa, 16, 164
notowania, 334

O

obiekt, 49
  ActiveXObject, 342
  Ajax, 308
  Date, 138, 271
  document, 47, 246
  DOM, 48
  Employee, 381
  Error, 342
  FormData, 316, 325
  loginForm, 49
  Math, 93
  Number, 92, 130
  String, 130
  window, 232
  XHR, 308
  XMLHttpRequest, 308, 319
  YUI, 364
obiektowość prototypowa, 379
obiektowy model dokumentu, 26
obiekty, 105, 163
  konfiguracyjne, 378
  niezmienne, 166
  własne, 381
  zmienne, 166
obliczanie silni, 197
obsługa
  ciasteczek, 266
  formularza kontaktowego, 128
  kliknięć, 332
  odpowiedzi serwera, 311
  przesyłu plików, 290
  wyjątków, 340
  wysyłki formularza, 329, 405
  zdarzeń, 46, 201, 357, 366
    standard W3C, 204
    tradycyjna, 203
    w IE, 205
    zaawansowana, 217
  żądań Ajax, 409
odczyt ciasteczek, 263
okienka wyskakujące, 249
okna
  dialogowe, 230
  modalne, 230, 257
  potwierdzeń, 231
okno
  przeglądarki, 233
  wyboru pliku, 291
  z komunikatem, 50
  zapytania, 231
określanie daty zdarzenia, 148
opcje wyboru, 287
operator
  identyczności, 114
  iloczynu logicznego, 125
  in, 164
  instanceof, 390
  new, 390
  przypisania, 82, 101, 164
  sumy logicznej, 125
  trójargumentowy, 87
  typeof, 130, 164, 205, 389
  warunku, 124
operatory
  arytmetyczne, 86
  jednoargumentowe, 87
  logiczne, 115
  porównań, 111, 112
osadzanie kodu, 40

P

pamięć podręczna przeglądarki, 336
parametry funkcji, 172, 183
parowanie zdarzeń, 215
PCRE, Perl Compatible Regular Expressions, 292
pętla
  for, 131
  while, 134
PHP, 26
piaskownica, 363
plik
  ajax.js, 313, 342
  contact.js, 330
  content.js, 331
  employee.js, 167, 194
  epoch.js, 210
  errorMessages.js, 302
  event.js, 148
  fader.js, 388
  login.js, 46, 326, 412
  membership.js, 223
  os.js, 285
  popup.js, 239, 249
  print.js, 254
  quote.js, 335
  random.js, 133, 181
  register.js, 302, 304
  shopping.js, 89
  tasks.js, 155, 196, 382
  test.js, 313, 348
  text.js, 212
  theme.js, 267
  today.js, 179
  utilities.js, 206, 302
  view.js, 413
  widok.php, 412
  words.js, 190
pobieranie
  cen akcji, 333
  ciasteczka, 268
  daty, 141
  elementów, 355
  elementu tablicy, 153
podpowiadanie kodu, 57
podpowiedź, 279, 282, 301
pola tekstowe, 281, 282
polecenia SQL, 397
polecenie
  continue, 135
  echo, 322
  finally, 340
  return, 180, 327
  throw, 340, 341
  try, 340
poprawność
  adresu e-mail, 303
  kodu pocztowego, 303
  numeru telefonu, 303
porównanie identyczności, 114
porównywanie
  liczb, 125
  tekstów, 127
potwierdzenia, 231
pozycjonowanie okna, 234
prezentacja postępu pobierania, 324
problem asocjacyjności, 87
procedury obsługi zdarzeń, 47, 202
proces renderowania strony, 48
program do emulacji, 68
progresywna rozbudowa, 202
projekt portalu akcyjnego, 396
proste typy zmiennych, 85
protokół bezstanowy, 262
prototypy, 16, 163, 384
przechwytywanie błędów, 340

przeglądarka, 13, 23
  Google Chrome, 63
  Internet Explorer, 65
  Mozilla Firefox, 64
  Opera, 66
  Safari, 66
przekazywanie
  funkcji, 189
  obiektu, 178
  tablic, 185
  wartości, 173
    przez referencję, 177
    przez wartość, 177
przekierowanie przeglądarki, 243
przeliczanie daty, 144
przestrzeń nazw, 185, 378
przycisk drukowania, 254
przyciski opcji, 289
przyjazna degradacja, 42
punkt
  wstrzymania, 77
  wstrzymania warunkowy, 78
pusty tekst, 85

R

ramki, 242
raportowanie zdarzenia, 219
refaktoryzacja, 57
referencja, 175
  do arkusza stylów, 261
  do elementów, 355
  do formularza, 47, 277
  do obiektu zdarzenia, 217
reguła tego samego źródła, 242
rekurencja, 197
RIA, Rich Internet Applications, 27
rodzaje
  błędów, 72
  zdarzeń, 208
rozpoznawanie typu, 389
rozszerzenia
  Chrome, 63
  Firefoksa, 64
  Opery, 66
równość, 114

S

sekwencje sterujące, 103
selektory CSS, 251, 355, 357
semantyczny HTML, 31, 43
serwer WWW, 14
sesje, 262
silnia, 197
składnia XPath, 252
składowe, 152
skrypt
  getBids.php, 408
  logowania, 407
  PHP, 322, 361, 399
  pobrania oferty, 408
  utilities.js, 409
  widok.php, 403, 408
  zgłoszenia oferty, 407
skrypty pośredniczące, 334
słaba kontrola typów, 83
słowo kluczowe
  function, 48, 172
  new, 152
  this, 194, 379
  var, 82, 173
sortowanie, 188
specyfikacja DOM Level 2, 204
sprawdzanie poprawności
  formularza, 21, 46
  parametrów, 174
stałe, 102
standard W3C, 204
stopniowe ulepszanie, 30, 46, 215, 309
strefy czasowe, 143
strona logowania, 402
struktura plików witryny, 399
style, 256
system kontroli wersji, 57
szablon
  HTML, 399
  HTML5, 36

Ś

ścieżki pełne, 41
ścieżki względne, 41

T

tabela z ofertami, 417
tabele HTML, 362
tablica, 151, 168
  asocjacyjna, 257
  events, 219
  wielowymiarowa, 158
  words, 192
TDD, Test Driven Development, 58
technologia
  Ajax, 18, 307, 313, 328, 407
  Flash, 27
test kaczki, 390
testowanie
  kodu, 68, 69
  komunikacji, 316
  na przeglądarkach, 348
  online, 68
  typu wartości, 390
testy, 58, 345
  jednostkowe, 343, 347
  wydajnościowe, 348
tryb
  przeglądarki, 35
  ścisły, 51
tworzenie
  adresów URL, 244
  bazy danych, 397
  biblioteki obsługi ciasteczek, 264
  ciasteczek, 263
  dynamicznych list wyboru, 284
  efektów, 358, 366
  elementów, 253
  formularza kontaktowego, 328
  formularza logowania, 326
  formularza z walidacją, 302
  formularza zgłoszenia oferty, 404
  funkcji, 171
  funkcji asercji, 343
  kalkulatora, 89, 95
  kodu, 45, 412
  komunikatu o błędzie, 277
  obiektów, 163
  obiektu Ajax, 308
  obiektu daty, 138
  obiektu konfiguracyjnego, 378
  okien, 235
  okien modalnych, 257
  okienka, 239

pliku login.js, 409
pliku view.js, 413
podpowiedzi, 279
pojedynczego obiektu, 379
procedur obsługi zdarzeń, 202
skryptów dla żądań Ajax, 406
skryptu PHP, 361
stron, 79
tablic, 151
tekstów, 96
widgetu, 334
własnego obiektu, 382
wyrażenia regularnego, 293
typ dokumentu, 34
typy pól tekstowych, 37

U

ukrywanie elementów, 257
uruchamianie testów, 346
usuwanie
ciasteczek, 264
elementów tablicy, 158
elementu, 279
właściwości obiektu, 166
UTC, 143

W

walidacja, 21, 34, 49
formularza, 301
identyfikatora elementu, 403
imienia, 303
kodu, 113
listy wyboru, 283
nazwy użytkownika i hasła, 402
oferty, 416
po stronie klienta, 51
po stronie serwera, 51
pól i obszarów tekstowych, 281
walidator
kodu, 74
W3C, 80
XML, 318
warstwy CSS, 273
wartości parametrów, 176
wartość
false, 111
Infinity, 88
NaN, 88
null, 85
true, 111
undefined, 85, 106, 176
warunek sprawdzający, 116
wczytywanie
danych, 331
obrazów, 333
wersja
HTML, 34
XHTML, 34
wersje JavaScript, 29
węzły, 247, 365
widget
automatycznego uzupełniania, 360, 367, 368
cen akcji, 334
wyboru daty, 359
właściwości, 16
ekranu, 235
elementów, 356
myszy, 222
obiektu, 164
okien, 237
węzła, 248
zdarzeń, 218
właściwość
checked, 288
childNodes, 248
children, 248
className, 256
constructor, 390
currentStyle, 256
dataType, 358
defaultValue, 281
display, 260
document.cookie, 246
document.styleSheets, 261
documentElement, 319
firstChild.nodeValue, 320
hash, 243
innerHTML, 252
length, 175
location, 243
nodeType, 248
onreadystatechange, 310
onsubmit, 49
opener, 241
position, 280
readyState, 311, 325
search, 243
statusText, 312
style, 256
textContent, 132
visibility, 280
window.frames, 242
window.location, 243
włączanie
listy wyboru, 286
obsługi JavaScript, 24
wpis DOCTYPE, 34
wskazanie etykiety, 135
wskaźnik postępu, 324
wstawianie ciasteczka, 268
wybieranie elementów, 355
wyjątek, 341
wykonanie kodu, 69
wykrywanie
funkcjonalności, 49
obiektów, 44
wyliczanie interwałów, 147
wyrażenia
czujki, 78
regularne, 292–299
XPath, 252
WYSIWYG, What You See Is What You Get, 58
wyskakujące okienko, 239
wysyłanie
danych, 315, 321
formularza, 281
plików, 290, 325
wyszukiwanie znaków, 97
wyświetlanie
elementów, 257
listy ofert, 406
szczegółów przedmiotu, 403
wywoływanie funkcji, 174
wzorce, 294
wzorzec projektowy MVC, 26

X

XHR, XML HTTP Request, 308
XHTML, 40
XML, eXtensible Markup Language, 18

Y

YQL, Yahoo! Query Language, 369–372
YUI, Yahoo! User Interface, 363–367

Z

zagnieżdżanie
  instrukcji warunkowych, 119
  pętli, 134
zakładki, 244
zapamiętywanie stanu, 262, 330
zapobieganie
  akcji domyślnej, 222, 304
  bąbelkowaniu zdarzeń, 226
zapytania, 231
  SQL, 400
  YQL, 370, 372
zarządzanie zadaniami, 155
zasięg
  globalny, 83, 182
  lokalny, 182
  zmiennej, 82, 182
zaśmiecanie przestrzeni nazw, 185
zdarzenia
  ekranów dotykowych, 210
  formularza, 214
  klawiatury, 211
  kursora, 209
  przeglądarki, 213
  przycisków, 208
zdarzenie
  blur, 214, 238
  change, 214
  click, 208
  focus, 214, 238
  keydown, 221
  keyup, 221
  load, 213
  mousemove, 209
  mouseover, 210
  progress, 325
  reset, 214
  resize, 213
  scroll, 213
  unload, 213
zgłaszanie wyjątków, 341
złączenia, 100
zmiana
  aktywności, 238
  daty, 144, 147
  elementów, 251
  prototypu, 385
  wielkości liter, 101
  wyglądu widgetów, 368
zmienna
  loginForm, 47
  tasks, 387
  this, 193, 220
zmienne, 82
  globalne, 83, 182
  lokalne, 183
  proste, 82
znacznik
  czasowy, 139, 145
  otwierający, 42
  zamykający, 42
znaczniki granic znaków, 300
znajdowanie elementów tablicy, 154
znak
  #, 244
  pionowej kreski, 323
  ucieczki, 85
znaki
  specjalne, 295
  wieloznaczności, 295
zwracanie
  danych JSON, 324
  danych XML, 323
  wartości, 180

Ź

źródła błędów, 72

Ż

żądania
  Ajax, 310, 330, 367, 407, 416
  asynchroniczne, 310, 332
  synchroniczne, 332
żądanie
  GET, 310, 316, 403
  POST, 310, 315, 403

JavaScript — prosty do opanowania i najlepszy do wykorzystania w zaawansowanych projektach internetowych!

Najwyższy czas na aktualną i przystępną książkę na temat JavaScriptu. Za pośrednictwem tej znakomitej pozycji dla początkujących autor bestsellerów Larry Ullman nauczy Cię zasad korzystania z tego języka i zaprezentuje najlepsze współczesne praktyki.

To książka, dzięki której przekonasz się, że nie musisz być programistą, by swobodnie poruszać się w świecie JavaScriptu. Liczne rysunki, czytelne przykłady oraz instrukcje krok po kroku sprawią, że nauka stanie się przyjemnością. W trakcie lektury poznasz fantastyczne narzędzia dla programistów, typy zmiennych oraz składnię języka. Ponadto nauczysz się obsługiwać zdarzenia, konstruować funkcje, komunikować się z przeglądarką oraz korzystać z techniki AJAX. Znajdziesz tu również omówienie najlepszych bibliotek na rynku, a wśród nich informacje na temat jQuery. Książka ta jest idealną pozycją dla każdego pasjonata stron WWW, który chce wykorzystać potencjał języka JavaScript.

Dzięki tej książce:

- poznasz dostępne narzędzia
- zaznajomisz się ze składnią języka
- sprawdzisz dostępne biblioteki
- błyskawicznie opanujesz JavaScript!

Larry Ullman — pisarz, programista, trener, instruktor, prelegent i konsultant. Napisał dwadzieścia dwie książki i dziesiątki artykułów. Wielu czytelników potwierdza, że jego mocną stroną jest tłumaczenie skomplikowanych zagadnień w przystępny sposób, zrozumiały nawet dla osób, które dopiero rozpoczynają przygodę z programowaniem lub projektowaniem.

Helion
Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.p

Cena: 69,00 zł
ISBN 978-83-246-5148-1
9 788324 651481