

Poznaj możliwości HTML5!

HTML5

nieoficjalny podręcznik



Matthew MacDonald

O'REILLY®



Tytuł oryginału: HTML5: The Missing Manual

Tłumaczenie: Maciej Reszotnik

ISBN: 978-83-246-3948-9

© 2012 Helion S.A.

Authorized Polish translation of the English edition of HTML5: The Missing Manual, 1st Edition
9781449302399 © 2011 Matthew MacDonald.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/htm5np>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Nieoficjalna czołówka	11
Wstęp	15
Część I. Wprowadzenie do języka	23
Rozdział 1. Wprowadzenie do HTML5	25
Historia HTML5	25
XHTML 1.0: rygor ponad wszystko	26
XHTML 2: niespodziewana porażka	27
HTML5: reaktywacja	27
HTML: żywy język	29
Trzy pryncypia HTML5	30
1. Nie psuj sieci	30
2. Brukuj ścieżki	30
3. Bądź praktyczny	32
Rzut oka na składnię HTML5	32
Element doctype a HTML5	34
Kodowanie znaków	35
Język	36
Dodawanie arkusza stylów	36
Dołączanie JavaScriptu	36
Ostateczny produkt	37
Składnia HTML5 z bliska	38
Rozluźnione reguły	38
Walidacja HTML5	39
Powrót XHTML-u	41
Rodzina znaczników HTML5	43
Dodane elementy	43
Komponenty usunięte ze specyfikacji	44
Elementy zaadaptowane	45
Zmodyfikowane znaczniki	46
Elementy standaryzowane	47

Korzystanie z HTML5 już dziś	48
Ocenianie wsparcia ze strony przeglądarek	49
Statystyki poziomu przyjęcia przeglądarek	51
Wykrywanie obsługi własności z aplikacją Modernizr	52
Uzupełnianie braków przy użyciu wypełniania	55
Rozdział 2. Nowe podejście do projektowania stron	57
Wstęp do elementów semantycznych	58
Modernizacja tradycyjnej strony HTML	59
Struktura strony w stylu klasycznym	60
Struktura strony w HTML5	63
Podtytuły i znacznik <hgroup>	65
Dołączanie rysunków przy użyciu znacznika <figure>	66
Dodawanie ramki redaktorskiej — znacznik <aside>	68
Elementy semantyczne a kompatybilność z przeglądarkami	69
Projektowanie strony z nowymi elementami semantycznymi	72
Więcej o nagłówkach	72
Odnosiniki i element <nav>	74
Więcej o stopce	77
Więcej o sekcjach	81
System tworzenia konspektu strony w HTML5	82
Jak zobaczyć konspekt?	82
Konspekt podstawowy	83
Komponenty sekcji	84
Problemy z tworzeniem konspektów	87
Rozdział 3. Semantyczny kod HTML	91
Elementy semantyczne raz jeszcze	92
Data, czas i znacznik <time>	93
Obliczenia w JavaScriptcie i element <output>	94
Element <mark> i zaznaczanie tekstu	95
Inne standardy kodu semantycznego	97
ARIA (ang. Accessible Rich Internet Applications)	97
RDFa (ang. Resource Description Framework)	98
Mikroformaty	99
Mikrodane	104
Fragmenty sformatowane opracowane przez Google	107
Lepsze wyniki wyszukiwania	107
Wyszukiwarka przepisów	110
Część II. Tworzenie nowoczesnych stron	115
Rozdział 4. Udoskonalone formularze	117
Formularze	118
Modernizowanie tradycyjnego formularza HTML	119
Znak wodny — dodawanie wskazówek	123
Dobry punkt zaczepienia: właściwość focus	124

Walidacja: wykrywanie błędów	125
Proces walidacji w HTML5, krok po kroku	125
Wyłączanie mechanizmu walidacji	127
Formatowanie kontrolek walidacyjnych	128
Walidacja wyrażeń regularnych	129
Własne reguły walidacji	130
Obsługa mechanizmu walidacji	132
Nowe typy znacznika input	134
Adresy e-mail	137
Adresy URL	137
Pola wyszukiwania	137
Telefon	138
Liczby	138
Suwak	139
Czas: daty i godziny	140
Kolor	141
Nowe elementy	141
Sugerowane odpowiedzi i element <datalist>	142
Pasek stanu i miernik	144
Paski narzędzi i menu — znaczniki <command> i <menu>	146
Edytor HTML na stronie	147
Edytowanie zawartości za pomocą contentEditable	147
Edytowanie strony za pomocą atrybutu designMode	149

Rozdział 5. Multimedia 153

Wideo dziś	154
Wprowadzenie do audio i wideo w HTML5	155
Wydobywanie dźwięku z elementu <audio>	156
Znacznik <video> z szerszej perspektywy	158
Wojna o format	159
Więcej o formatach	160
Obsługa multimediów w przeglądarkach	161
Wiele formatów, czyli jak udobruchać każdą przeglądarkę	164
Element <source>	164
Alternatywa — wtyczka Flasha	166
Sterowanie odtwarzaniem za pomocą JavaScriptu	169
Dodawanie efektów dźwiękowych	170
Budowa własnego odtwarzacza filmów	173
Odtwarzacze JavaScript	175
Napisy i dostępność	177

Rozdział 6. Podstawy rysowania na elemencie canvas 179

Płótno — wprowadzenie	180
Linie proste	182
Ścieżki i figury	184
Krzywe	186
Transformaty	188
Przezroczystość	192

Tworzenie prostego programu graficznego	194
Przygotowanie narzędzi	195
Malowanie po płótnie	196
Zachowywanie płótna	198
Płótno i kompatybilność z przeglądarkami	201
Wypełnianie płótna	201
Alternatywne płótna i wykrywanie obsługi	203
Rozdział 7. Więcej o płótnie	205
Inne własności płótna	205
Rysowanie obrazów	206
Wycinanie i zmienianie wielkości obrazu	207
Rysowanie tekstu	208
Cienie i inne ozdobniki	210
Dodawanie cieni	210
Wypełnianie figur deseniem	212
Wypełnianie figur gradientem	213
Składanie wszystkiego w całość: rysowanie wykresów	216
Interaktywne figury	221
Śledzenie rysowanych elementów	221
Współrzędne i lokalizowanie trafień	224
Animowanie płótna	226
Podstawowa animacja	227
Animowanie wielu obiektów	228
Praktyczny przykład: labirynt	233
Rysowanie labiryntu	234
Animowanie ikony	235
Lokalizowanie trafień a barwa pikseli	237
Rozdział 8. Rewolucja w stylach — CSS3	241
Używanie CSS3 już dziś	242
Strategia 1.: Wykorzystaj to, co możesz	242
Strategia 2.: Traktuj własności CSS3 jak usprawnienia	243
Strategia 3.: Dodanie awaryjnych mechanizmów za pomocą Modernizra	244
Style właściwe konkretnym przeglądarkom	246
Typografia w sieci	247
Formaty fontów	248
Używanie zestawów fontów	250
Korzystanie z fontów sieciowych Google	253
Korzystanie z własnych fontów	255
Wielokolumnowy tekst	256
Przystosowanie stron do różnych urządzeń	258
Zapytania medialne	259
Zapytania medialne — wyższa szkoła jazdy	262
Zastępowanie całego arkusza stylów	263
Rozpoznawanie urządzeń mobilnych	264

Kontenery na błysk	265
Przezroczystość	266
Zaokrąglane rogi	268
Tło	269
Cienie	269
Gradienty	271
Efekty przejścia	273
Przekształcanie koloru	274
Przejścia — teczka z pomysłami	276
Transformaty	276

Część III. Konstruowanie aplikacji sieciowych przy użyciu komponentów desktopowych 281

Rozdział 9. Magazyn danych 283

Magazyn sieciowy — podstawy	284
Magazynowanie danych	285
Praktyczny przykład: zapisywanie stanu gry	286
Magazyn sieciowy a obsługa przeglądarek	288
Magazyn sieciowy — na głębszych wodach	289
Usuwanie wpisów	289
Listowanie wszystkich zachowanych wpisów	289
Zapisywanie liczb i dat	290
Zachowywanie obiektów	292
Reagowanie na zmiany w magazynie	293
Odczytywanie plików	295
Pobieranie pliku	295
File API i obsługa przeglądarek	296
Odczytywanie pliku tekstowego	296
Zastępowanie standardowej kontrolki ładowania plików	298
Odczytywanie wielu plików jednocześnie	299
Odczytywanie pliku graficznego	299

Rozdział 10. Aplikacje sieciowe z trybem offline 303

Cachowanie plików	304
Tworzenie manifestu	305
Korzystanie z manifestu	306
Przenoszenie manifestu na serwer	307
Uaktualnianie manifestu	310
Obsługa w przeglądarkach aplikacji w trybie offline	312
Praktyczne techniki cachowania	313
Uzyskiwanie dostępu do cachowanych plików	313
Tryb awaryjny	315
Sprawdzanie stanu połączenia	316
Wykrywanie uaktualniania przy użyciu JavaScriptu	317

Rozdział 11. Komunikacja z serwerem sieciowym	321
Wysyłanie wiadomości na serwer	322
Obiekt XMLHttpRequest	322
Wysyłanie zapytań na serwer	323
Pobieranie nowych treści	327
Zdarzenia przesyłane na serwer	331
Format wiadomości	332
Wysyłanie wiadomości za pomocą skryptu serwera	333
Przetwarzanie wiadomości na stronie	335
Polling a zdarzenia po stronie serwera	336
Technologia WebSocket	337
Ocena technologii WebSocket	338
Prosty klient w technologii WebSocket	339
Przykłady technologii WebSocket w sieci	341
Rozdział 12. Więcej ciekawych sztuczek JavaScriptu	343
Geolokalizacja	344
Jak działa geolokalizacja?	345
Odnajdywanie współrzędnych użytkownika	347
Usuwanie błędów	349
Ustawienia geolokalizacji	350
Generowanie mapy	352
Monitorowanie ruchu użytkownika	355
Obiekt pracownika	355
Czasochłonne zadanie	357
Wykonywanie zadań w tle	359
Obsługa błędów pracownika	361
Anulowanie zadania uruchomionego w tle	362
Przekazywanie bardziej złożonych wiadomości	362
Zarządzanie historią	364
Kwestia URL	366
Tradycyjne rozwiązanie: znak kratki i adres URL	366
Rozwiązanie HTML5: historia sesji	368
Historia sesji i kompatybilność	370
Część IV. Dodatki	373
Dodatek A. Krótki wstęp do arkuszy stylów	375
Załączanie stylów do stron	375
Anatomia arkusza stylów	376
Własności CSS	377
Formatowanie elementów przy użyciu klas	377
Komentarze w arkuszach stylów	379
Odrobinę bardziej zaawansowane arkusze stylów	379
Konstruowanie struktury strony przy użyciu elementu <div>	380
Wiele selektorów	380
Selektory kontekstowe	381

Selektor identyfikatora	382
Selektory pseudoklas	382
Selektory atrybutów	383
Wycieczka po stylach	384
Dodatek B. Krótki wstęp do języka JavaScript	389
W jaki sposób witryny korzystają z JavaScriptu?	390
Zagnieżdżanie kodu w dokumencie HTML	390
Używanie funkcji	391
Przenoszenie kodu JavaScript do oddzielnego pliku	393
Odpowiadanie na zdarzenia	394
Podstawy składni języka	395
Zmienne	395
Wartość null	397
Zakres zmiennych	397
Typy danych	398
Operacje	398
Instrukcje warunkowe	400
Pętle	401
Tablice	401
Funkcje — otrzymywanie i zwracanie danych	402
Interakcja ze stroną	403
Manipulowanie elementem	404
Dynamiczne łączenie ze zdarzeniem	406
Zdarzenia wplątane	408
Skorowidz	410



Udoskonalone formularze

Formularze HTML są zbiorem elementów, których przeznaczeniem jest pobranie informacji od osób odwiedzających stronę. Są wśród nich pola tekstowe, w które można wpisać treści, rozwijane listy, z których mogą wybrać opcję, pola wyboru, które można zaznaczać i usuwać znaczenie itp. Istnieje wiele sposobów na wykorzystanie formularzy HTML, a jeśli przebywałeś w sieci przez więcej niż tydzień, pewnie wiesz, że pojawiają się praktycznie wszędzie — od strony z informacjami o giełdzie, po stronę rejestracji skrzynki e-mail.

Formularze HTML są z nami od czasu wyklarowania się tego języka i od zeszłego stulecia nie zmieniły się ani trochę pomimo wielu prób. Twórcy standardów internetowych przez lata opracowywali ich następcę, określanego mianem XForms, który ostatecznie okazał się takim samym niewypałem jak XHTML 2 (strona 27). Choć XForm dość elegancko rozwiązywał pewne problemy z oryginalnym pomysłem, to jednak tworzył też kolejne. Kod XForms był rozwlekły i zakładał, że projektant strony jest dobrze obeznany z językiem XML. Największą wadą było to, że XForms nie był w żadnym względzie kompatybilny z HTML, co oznaczało tyle, iż twórcy stron musieliby zamknąć oczy i „przeskoczyć” na nowy standard, mając nadzieję, że jakoś to będzie. Co można było przewidzieć, większość webdeveloperów nie zwracała sobie głowy implementacją XForms — był to standard zbyt złożony, by nadawał się do powszechnego wykorzystania — zatem twórcy stron nigdy nie dokonali wymaganego skoku.

W HTML5 obrano inne podejście. Wprowadzono wiele usprawnień do już istniejącego modelu formularzy, dzięki czemu ulepszone ich wersje będą działać na starszych przeglądarkach, choć bez specjalnych wodotrysków (co jest dobrą wieścią, ponieważ z przeglądarek Microsoftu dopiero Internet Explorer 10 zapewni pełną obsługę nowych własności formularzy). W HTML5 dodano również opcje, z których twórcy stron korzystają już dziś. Różnica polega na tym, że HTML5 sprawia, iż są łatwo dostępne, i nie wymaga kodu lub narzędzi JavaScriptu dostarczanych przez zewnętrzne firmy.

W tym rozdziale zapoznasz się z nowymi własnościami formularzy HTML5. Dowiesz się, które z nich są powszechnie obsługiwane, a które obecnie nie cieszą się żadnym wsparciem. Przyjrzyj się również właściwości, która — choć nie jest częścią standardu HTML — wzbogaca warstwę interakcji przez umieszczenie złożonego edytora HTML na stronie.

Formularze

Najprawdopodobniej masz już doświadczenie w tworzeniu formularzy. Jeśli jednak nie jesteś pewien szczegółów, krótka powtórka odświeży Ci pamięć.

Formularz sieciowy to zbiór pól tekstowych, list, przycisków i innych komponentów, przy użyciu których użytkownik wprowadza pewne informacje na stronie. W sieci formularze występują na każdym kroku — pozwalają na rejestrację skrzynki e-mail, zakup produktów, wykonywanie transakcji bankowych itp. Najprostsze formularze, wykorzystywane w wyszukiwarkach internetowych, składają się z pojedynczego pola tekstowego, tak jak na rysunku 4.1.



Wszystkie formularze sieciowe działają na tej samej zasadzie. Użytkownik podaje informacje, po czym klika przycisk. W odpowiedzi strona pobiera wprowadzone dane i wysyła je na serwer. Na serwerze jakaś aplikacja przetwarza te dane i wykonuje kolejny krok. Serwerowy program może zainicjować komunikację z bazą danych (aby odczytać bądź zapisać jakieś informacje) przed wysłaniem do użytkownika nowej strony.

PRZYSPIESZAMY

Obchodzenie mechanizmu przesyłania danych przy użyciu JavaScriptu

Warto wspomnieć, że formularze nie są jedynym rozwiązaniem, które umożliwia przesyłanie wprowadzonych danych na serwer (były inne dawno, dawno temu). Aktualnie chytry twórca stron może komunikować się z serwerem za pomocą napisanego w kodzie JavaScript obiektu XMLHttpRequest (strona 325). Wyszukiwarka Google korzysta z niego na dwa sposoby. Po pierwsze, żeby wyświetlać sugerowane hasła wyszukiwania w liście pod polem tekstowym; po drugie, by generować wyniki wyszukiwania równoległe z wpisywaniem nowych treści. Ta ostatni własność działa po włączeniu opcji wyszukiwania dynamicznego (<http://www.google.pl/instant/>).

Może Ci się wydać, że dzięki JavaScriptowi uda się w ogóle pominąć krok bezpośredniego wysyłania danych, tak jak ma to miejsce z wyszukiwaniem dynamicznym Google. Jednakże, choć warto zaoferować taką możliwość, nie może to być jedyna opcja. Wynika to z prostego faktu, że zaprezentowane podejście w języku JavaScript nie jest

doskonałe (może powodować błędy przy wolniejszych łączach), oraz tego, iż wciąż są osoby, których przeglądarki nie obsługują tego języka lub opcja obsługi jest w nich wyłączona.

Oczywiście, nie ma nic złego w stronach, których formularze nie przesyłają nigdzie danych. Z pewnością widziałeś strony, na których można dokonywać prostych obliczeń (np. kalkulatory odsetek z kredytów hipotecznych). Obsługa serwera nie jest w nich wymagana, gdyż dołączony do tych dokumentów kod JavaScript może wykonać kalkulacje i wyświetlić wyniki.

Z perspektywy HTML5 nie ma znaczenia, czy prześlesz swój formularz na serwer w zwykły sposób, użyjesz danych z niego do obliczeń lokalnych, czy wyślesz je z pomocą obiektu XMLHttpRequest. W każdym z tych przypadków swój formularz skonstruujesz, posługując się standardowymi elementami HTML

Cały szkopuł tkwi w tym, że jest wiele sposobów na skonstruowanie programu działającego po stronie serwera (aplikacji przetwarzającej odczytane z formularza informacje). Niektórzy deweloperzy używają prostych skryptów, które pozwalają im na manipulowanie surowymi danymi, podczas gdy inni korzystają z dobrodziejstw wysokopoziomowych modeli, które opakowują informacje w wygodne obiekty. Niezależnie od podejścia, procedura sprowadza się do tego samego — pobrania danych, ich przetworzenia i zwrócenia rezultatów obliczeń na stronie.

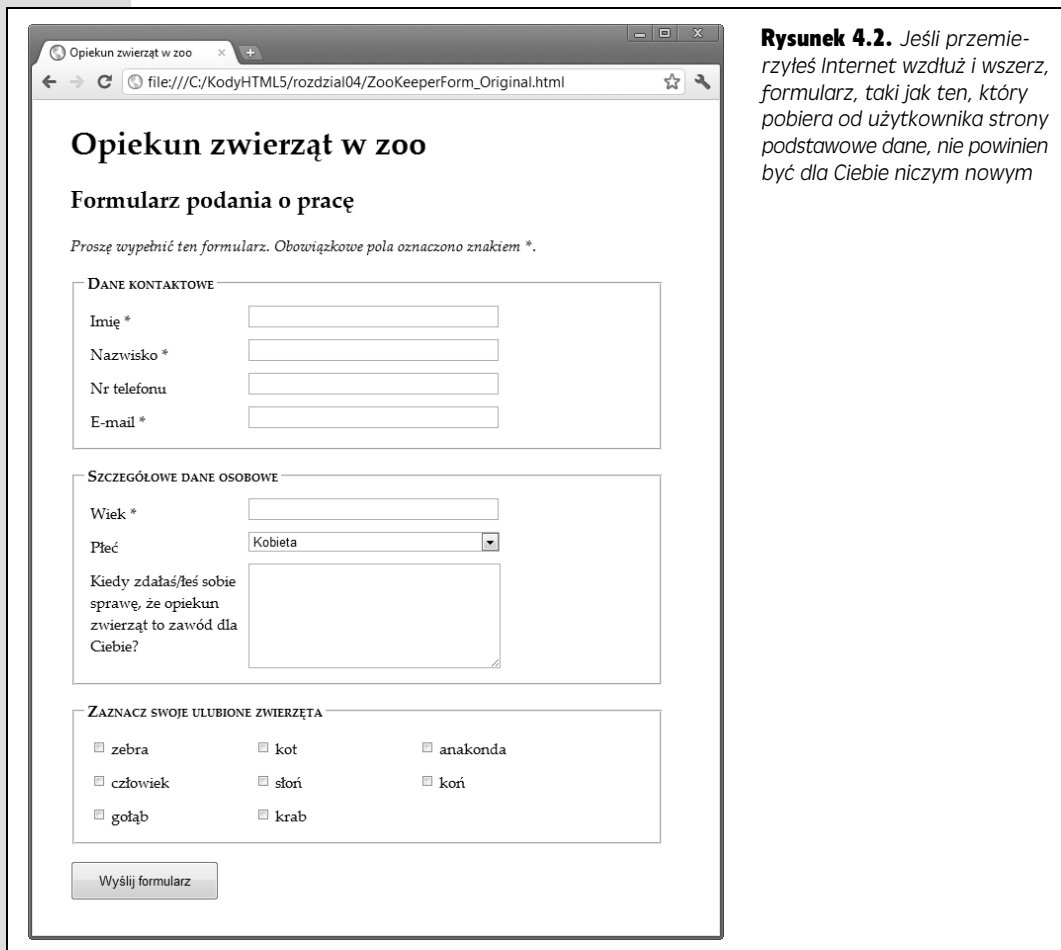
Uwaga: W książce tej nie faworyzuję się żadnego z dostępnych narzędzi do programowania po stronie serwera. W zasadzie nie ma to większego znaczenia, ponieważ każde z rozwiązań korzysta z tych samych elementów formularza, którego dotyczą reguły HTML5.

Modernizowanie tradycyjnego formularza HTML

Najlepszą metodą nauki konstruowania formularzy w HTML5 jest ulepszenie już istniejącego, napisanego w standardowy sposób. Na rysunku 4.2 widać przykład, który zostanie omówiony.

Kod jest prosty jak konstrukcja cepa. Jeśli budowałeś wcześniej formularze, nie ujrzysz tu niczego nowego. Cały formularz został wyznaczony parą znaczników `<form>`.

```
<form id="zooKeeperForm" action="processApplication.cgi">
  <p><i>Proszę wypełnić ten formularz. Obowiązkowe pola oznaczono znakiem
</i><em>*</em></p>
  ...
```



Rysunek 4.2. Jeśli przemie-
rzyłeś Internet wzdłuż i wszerz,
formularz, taki jak ten, który
pobiera od użytkownika strony
podstawowe dane, nie powinien
być dla Ciebie niczym nowym

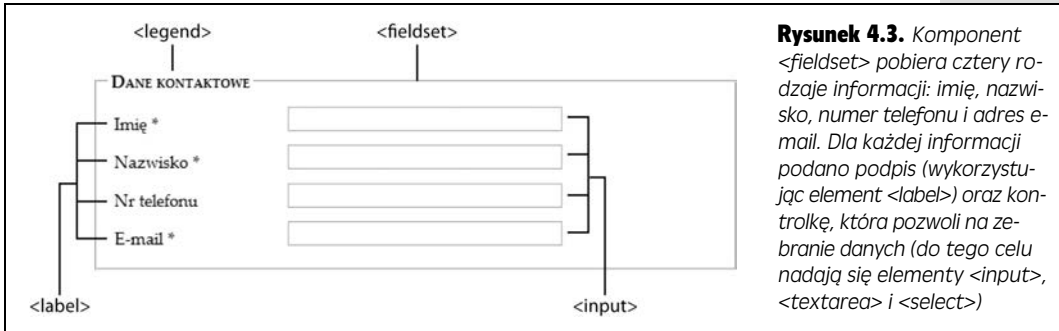
Element `<form>` zawiera wszystkie widżety formularza (zwane też **kontrolkami** lub **polami**). Informuje też przeglądarkę (z wykorzystaniem adresu URL w atrybucie `action`), gdzie ma wysłać zebrane dane. Jeśli całość operacji ma być wykonywana przez kod JavaScript po stronie klienta, możesz nadać mu wartość `#`.

Uwaga: HTML5 dodaje mechanizm, dzięki któremu wolno zamieszczać kontrolki formularza poza jego obrębem. Cała sztuczka polega na użyciu atrybutu `form` (np. `form="zooKeeperForm"`) z wartością atrybutu `id`, który nadano znacznikowi `<form>` (np. `<form id="zooKeeperForm">...</form>`). Niestety, przeglądarki, które nie obsługują tej własności, pominią tak oznaczone dane w trakcie przesyłania formularza, co oznacza, że na razie można zapomnieć o używaniu tej właściwości.

Dobrze zaprojektowany formularz, taki jak ten dla pracownika zoo, dzieli się na logiczne części — co zapewnia użycie znacznika `<fieldset>`. Każda z tych części powinna posiadać tytuł, który wpisuje się w element `<legend>`. Oto obszar komponentu `<fieldset>` dla sekcji danych kontaktowych (rysunek 4.3).

```
<fieldset>
  <legend>Dane kontaktowe</legend>
  <label for="first name">Imię <em>*</em></label>
  <input id="first name"><br>
```





Rysunek 4.3. Komponent `<fieldset>` pobiera cztery rodzaje informacji: imię, nazwisko, numer telefonu i adres e-mail. Dla każdej informacji podano podpis (wykorzystując element `<label>`) oraz kontrolkę, która pozwoli na zebranie danych (do tego celu nadają się elementy `<input>`, `<textarea>` i `<select>`)

```
<label for="family name">Nazwisko <em>*</em></label>
<input id="family name"><br>
<label for="telephone">Nr telefonu</label>
<input id="telephone"><br>
<label for="email">E-mail <em>*</em></label>
<input id="email"><br>
</fieldset>
```

Jak we wszystkich formularzach, i tu najważniejszymi elementami są znaczniki `<input>`, które pobierają tekst, tworzą pola wyborów, przyciski opcji itp. Oprócz nich w formularzach używa się komponentu `<textarea>`, który pozwala na wpisywanie wielu linii tekstu, i znacznika `<select>` tworzącego listy. Jeśli czujesz, że przydałaby Ci się powtórka, rzuć okiem na tabelę 4.1.

Oto pozostała część formularza dla opiekunów zwierząt w zoo z kilkoma nowymi szczegółami (listą ze znacznikiem `<select>`, polami wyboru i przyciskiem *Wyślij formularz*).

```
<fieldset>
  <legend>Szczegółowe dane osobowe</legend>
  <label for="age">Wiek <em>*</em></label>
  <input id="age"><br>
  <label for="gender">Płeć</label>
  <select id="gender">
    <option value="female">Kobieta</option>
    <option value="male">Mężczyzna</option>
  </select><br>
  <label for="comments">Kiedy zdałaś/łeś sobie sprawę, że opiekun
  ↪zwierząt to zawód dla Ciebie?</label>
  <textarea id="comments"></textarea>
</fieldset>

<fieldset>
  <legend>Zaznacz swoje ulubione zwierzęta</legend>
  <label for="zebra"><input id="zebra" type="checkbox"> zebra</label>
  <label for="cat"><input id="cat" type="checkbox"> kot</label>
  <label for="anaconda"><input id="anaconda" type="checkbox">
  ↪anakonda</label>
  <label for="human"><input id="human" type="checkbox"> człowiek</label>
  <label for="elephant"><input id="elephant" type="checkbox"> słoń</label>
  <label for="wildebeest"><input id="wildebeest" type="checkbox">
  ↪antylopa</label>
  <label for="pigeon"><input id="pigeon" type="checkbox"> gołąb</label>
  <label for="crab"><input id="crab" type="checkbox"> krab</label>
</fieldset>
<p><input type="submit" value="Wyślij formularz"></p>
```

Tabela 4.1. Kontrolki formularza

Nazwa kontrolki	Znacznik HTML	Opis
Pole tekstowe (jeden wiersz)	<code><input type="text"></code> <code><input type="password"></code>	Definiuje pole tekstowe, w którym użytkownik może wpisywać informacje. Komponent <code><input type="password"></code> , który służy do wpisywania haseł, nie wyświetla wpisanego tekstu. Zamiast liter użytkownik ujrzy w nim gwiazdki (*) lub kropki (·).
Wieloliniowe pole tekstowe	<code><textarea>...</textarea></code>	Definiuje duże pole tekstowe, w które można wpisać wiele linijek tekstu.
Pole wyboru	<code><input type="checkbox"></code>	Definiuje pole wyboru, które można zaznaczać i usuwać z niego zaznaczenie.
Przycisk radiowy	<code><input type="radio"></code>	Definiuje przycisk radiowy (kółka, które można zaznaczać i usuwać z nich zaznaczenie). Zwykle przyciski radiowe zbiera się w grupy o takiej samej wartości atrybutu <code>name</code> .
Przycisk	<code><input type="submit"></code> <code><input type="image"></code> <code><input type="reset"></code> <code><input type="button"></code>	Definiuje standardowy przycisk. Przycisk typu <code>submit</code> powoduje odczytanie danych z formularza i ich przesłanie pod podany adres URL. Przycisk typu <code>image</code> działa tak samo, lecz wyświetla podany w nim obraz zamiast standardowej grafiki przycisku. Przycisk typu <code>reset</code> czyści wartości wszystkich kontrolki w formularzu. Z kolei przycisk typu <code>button</code> nie robi nic, chyba że dodasz do niego kod JavaScript.
Lista	<code><select>...</select></code>	Definiuje listę, z której użytkownik wybiera jedną lub wiele opcji. Każdą opcję na liście należy zawrzeć w elemencie <code><option></code> .

Stronę oraz załączony do niej prosty arkusz stylów znajdziesz wśród plików pobranych ze strony helion.pl/ksiazki/htm5np.htm. Wersję formularza napisaną w tradycyjnym HTML nazwano *ZookeeperForm_Original.html*. Jeżeli interesują Cię bardziej wprowadzone w HTML5 modyfikacje, rzuć okiem na plik *ZookeeperForm_Revised.html*.

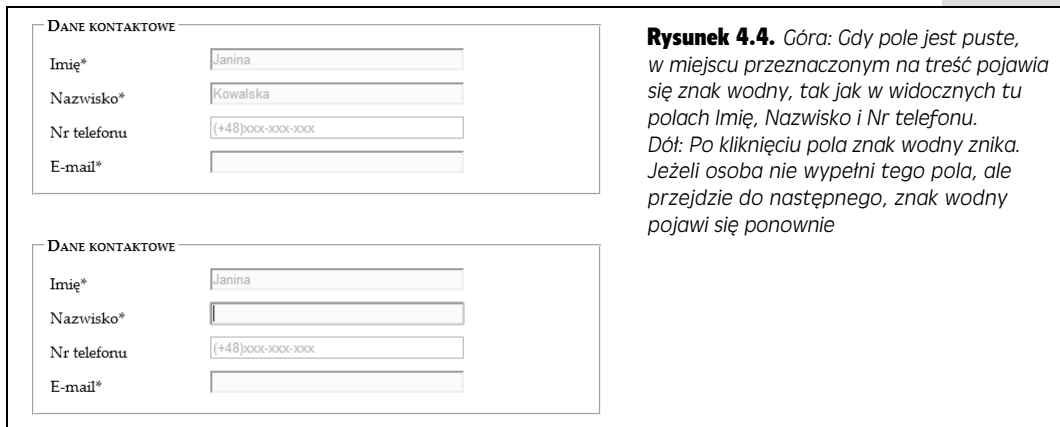
Uwaga: Jednym z podstawowych ograniczeń formularzy HTML jest to, że nie można w prosty sposób zmienić wyglądu generowanych przez przeglądarkę kontrolki. Nie da się np. zmienić zwykłego białego-szarego pola wyboru na duży czerwony znacznik. Możesz, oczywiście, utworzyć przy użyciu JavaScriptu inny element, który będzie zachowywał się tak jak pole wyboru, tzn. zmieniał wygląd po kliknięciu, przechodząc z jednego stanu w drugi i z powrotem.

HTML5 zachowuje to ograniczenie i nakłada je na wszystkie nowe kontrolki, o których dowiesz się więcej w dalszej części tego rozdziału. Oznacza to, że działanie formularzy może się nie spodobać osobom, które wymagają od strony wysublimowanych komponentów i pełnej kontroli nad jej wyglądem.

Zapoznałeś się już z formularzem — czas go zmodernizować przy użyciu HTML5. Zaczniemy od rzeczy prostych, czyli atrybutów `placeholder` i `autofocus`.

Znak wodny — dodawanie wskazówek

Po załadowaniu strony formularze są zwykle puste. Kolumna pustych pól tekstowych może jednak onieśmielać szczególnie wtedy, gdy nie wiadomo, co w nich ma być. Dlatego też zwykłą praktyką jest wypełnianie ich przykładowym tekstem przy użyciu atrybutu `placeholder`. Taki przykładowy tekst nazywa się **znakiem wodnym**, gdyż nadaje się mu jasnoszary kolor, aby odróżnić go od właściwych danych wprowadzonych przez użytkownika. Na rysunku 4.4 zaprezentowano znak wodny w działaniu.



Rysunek 4.4. Góra: Gdy pole jest puste, w miejscu przeznaczonym na treść pojawia się znak wodny, tak jak w widocznych tu polach Imię, Nazwisko i Nr telefonu. Dół: Po kliknięciu pola znak wodny znika. Jeżeli osoba nie wypełni tego pola, ale przejdzie do następnego, znak wodny pojawi się ponownie

Aby utworzyć znak wodny, po prostu wewnątrz znacznika dodaj atrybut `placeholder`.

```
<label for="family name">Nazwisko <em>*</em></label>
<input id="family name" placeholder="Kowalski"><br>
<label for="telephone">Nr telefonu</label>
<input id="telephone" placeholder="+48-xxx-xxx-xxx"><br>
<label for="email">E-mail <em>*</em></label>
<input id="email"><br>
```

Przeglądarki, które nie rozpoznają tego atrybutu, po prostu go zignorują (wśród nich jest, oczywiście, Internet Explorer). Na szczęście, nawet gdy tak się stanie, nie dojdzie do żadnej tragedii. W końcu znaki wodne są jedynie dodatkiem, który w żadnej mierze nie wpływa na działanie formularza. Jeśli jednak Ci to przeszkadza, na stronie <http://tinyurl.com/polyfills> pełno jest napisanych w JavaScriptcie łańcuchów, które uaktualnią nawet IE.

Obecnie nie ma sposobu na zmianę wyglądu tekstu w znaku wodnym (taką jak np. nadanie kursywy lub innego koloru). W przyszłości twórcy przeglądarek dodadzą obsługę arkusza stylów — gdy czytasz te słowa, być może już ją wprowadzili. Teraz w tym celu można eksperymentować ze specyficznymi dla każdej przeglądarki pseudoklasami (np. `-webkit-input-placeholder` i `-moz-placeholder`), ale najlepiej nie zwracać sobie tym głowy (wyjaśnienie działania pseudoklas znajdziesz na stronie 382).

PRZYSPIESZAMY

Prawidłowe znaki wodne

Nie musisz umieszczać znaków wodnych w każdym polu tekstowym. Używaj ich do wyjaśnienia potencjalnej dwuznaczności. Przykładowo nie trzeba wyjaśniać, co należy wpisać w pole *Nazwisko*, ale użycie słowa *Godność* mogłoby nie być zbyt oczywiste. Znak wodny daje jasno do zrozumienia, że jest to miejsce na nazwisko.

Często znakiem wodnym jest przykładowa wartość, czyli coś, co może być podstawową wartością. W wyszukiwarce przepisów Google (<http://www.google.com/landing/recipes>) wartością domyślną jest „chicken pasta” (makaron z kurczakiem), przez co jasne jest, że w pole należy wpisać nazwę potrawy, a nie listę składników lub nazwisko kucharza, który wymyślił przepis.

Znaków wodnych używa się czasem, żeby wskazać, jak wprowadzona wartość powinna być sformatowana. Pole *Nr telefonu* z rysunku 4.4 jest tego dobrym przykładem — wskazuje, że numery telefonu powinny składać się z numeru kodu kraju i dziewięciocyfrowego numeru telefonu. Taki znak wodny nie oznacza, że inaczej sformatowane dane nie zostaną przyjęte, lecz sugeruje prawidłowe rozwiązanie, które może pomóc niepewnym użytkownikom.

Nie powinieneś używać znaku wodnego do dwóch czynności. Po pierwsze, nie próbuj wcisnąć do pola opisów lub instrukcji. Załóżmy, że zdefiniowałeś pole, w które należy wpisać kod bezpieczeństwa karty kredytowej. W takim przypadku tekst: „Wpisz trzy cyfry wytypowane na odwrocie Twojej karty” nie nadaje się na znak wodny. Zamiast niego warto dodać podpis pod polem lub wykorzystać atrybut `title`, aby za każdym razem, gdy użytkownik najedzie na nie myszą, pojawiała się małe okienko z instrukcją.

```
<label for="promoCode">Kod
↳promocyjny</label>
<input id="promoCode" placeholder="QRB001"
↳title="Twój kod promocyjny zaczyna się
↳od trzech liter, po których występują
↳trzy cyfry">
```

Po drugie, nie powinieneś dodawać do znaków wodnych specjalnych znaków, aby je odróżnić od właściwego, wprowadzonego do pól tekstu. Przykładowo na niektórych stronach znak wodny zapisywany jest w nawiasach, np. *[Jan Kowalski]* zamiast *Jan Kowalski*. Nawiasy kwadratowe mają sugerować, że znak wodny jest jedynie przykładem. Taka konwencja jest zbędna i prowadzi do większego zamieszania.

Natomiast możesz bez przeszkód wykorzystać lepiej znany pseudoklasyk `focus`, by zmienić wygląd pola tekstowego, po tym jak zostanie wybrane. Załóżmy, że chcesz, żeby miało ciemniejszy kolor tła, tak by odróżniało się od reszty. Poniższa deklaracja CSS sprawi, że tak się stanie:

```
input:focus {
  background: #eaeaea;
}
```

Dobry punkt zaczepienia: właściwość `focus`

Zwykle, zaraz po załadowaniu formularza jego użytkownik chce po prostu zacząć go wypełniać. Niestety, nie może tego zrobić — przynajmniej dopóty, dopóki nie zaznaczy kliknięciem jednej z kontroltek lub nie przejdzie do niej przy użyciu klawisza *Tab*, w ten sposób ją aktywując.

Dowolną kontrolkę można aktywować domyślnie, wywołując w języku JavaScript metodę `focus()` na wybranym elemencie `<input>`. Wymaga to napisania całej linii kodu i w niektórych przypadkach może doprowadzić do pojawienia się dziwnych błędów. Jeśli np. użytkownik kliknie kontrolkę zaraz po załadowaniu strony i zacznie ją wypełniać, jeszcze zanim metoda `focus()` zostanie wywołana, w trakcie pisania może zostać przeniesiony na pole, którym nie jest zainteresowany. Rozwią-

zaniem tego problemu byłoby zapewnienie przeglądarkom zdolności kontrolowania aktywacji komponentów. W ten sposób mogłaby ona aktywować jedną kontrolkę, pod warunkiem że inna nie została wcześniej wybrana przez użytkownika.

Właśnie ten pomysł przyczynił się do powstania atrybutu `autofocus`, który można dołączyć do elementów `<input>` i `<textarea>`, tak jak na poniższym przykładzie:

```
<label for="last name">Nazwisko <em>*</em></label>  
<input id="name" placeholder="Janina Kowalska" autofocus><br>
```

Atrybut `autofocus` cieszy się identycznym poziomem wsparcia jak parametr `placeholder`, co oznacza, że wszystkie przeglądarki, z wyjątkiem Internet Explorera, go rozpoznają. Na szczęście i w tym przypadku łatwo załatać dziurę. Korzystając ze skryptu *Modernizr* (strona 52), możesz wykryć, czy przeglądarka obsługuje ten atrybut, i uruchomić swój własny kod automatycznego aktywowania elementu. Możesz też użyć gotowego wypełnienia, które doda odpowiednią obsługę (<http://tinyurl.com/polyfills>). Jednak szkoda zachodu na zabawę z tak mało znaczącą opcją, chyba że przy okazji dodasz obsługę innych opcji, takich jak omawiany niżej system walidacji.

Walidacja: wykrywanie błędów

Podstawową funkcją pól formularza jest pobranie od użytkownika strony informacji. Niecierpliwi lub zdezorientowani użytkownicy mogą pominąć ważne pole, wprowadzić złą treść lub kliknąć niewłaściwy przycisk. Wynikiem czegoś takiego jest otrzymanie mnóstwa bezużytecznych danych.

Każda porządna witryna powinna być wyposażona w mechanizm **walidacji**, odpowiedzialny za wykrywanie błędów (lub, w najlepszym wypadku, za zapobieganie ich powstawaniu). Przez lata deweloperzy tworzyli w tym celu własne procedury JavaScriptu lub wykorzystywali gotowe biblioteki. Oba rozwiązania sprawdzają się w wymiennie. Ze drugiej jednak strony, walidacja jest na tyle powszechnie wykorzystywanym mechanizmem (praktycznie każdy formularz wymaga zweryfikowania poprawności) i na tyle trudnym do zapewnienia (nikt nie chce pisać od podstaw podobnego kodu dla różnych formularzy, nie mówiąc już o jego *testowaniu*), że powinna istnieć inna metoda jego implementacji.

Twórcy HTML5 dostrzegli potencjalną niszę i wymyślili sposób na odebranie projektantom części odpowiedzialności za walidację i przerzucenie jej na przeglądarkę. Wynaleźli system walidacji *po stronie klienta* (szczegóły w ramce na stronie 126), który pozwala osadzić reguły sprawdzania błędów wewnątrz dowolnego znacznika `<input>`. Co najlepsze — system ten jest prosty w obsłudze — wystarczy posłużyć się właściwym atrybutem.

Proces walidacji w HTML5, krok po kroku

Podstawową rzeczą, o której należy pamiętać przy tworzeniu mechanizmu walidacji formularza, jest wskazanie, których elementów dotyczy, lecz nie musisz przy tym kłopotać się zbędnymi detalami. Przypomina to odrobinę awans do pracy w zarządzie, tyle że bez podwyżki.

PRZYSPIEZAMY

Dwie strony walidacji

Przez całe lata zmyślni twórcy witryn atakowali problem walidacji z różnych stron. Z biegiem czasu wyodrębiono najlepszą praktykę. W zwykłym formularzu wykorzystano dwie metody sprawdzania poprawności.

- ◆ **Walidacja po stronie klienta.** Ta forma walidacji zachodzi w przeglądarce użytkownika, jeszcze zanim prześle on formularz. Celem jest ułatwienie życia użytkownikom. Zamiast zwracać informacje o błędach po wypełnieniu wszystkich kontrolek i kliknięciu przycisku *Wyślij*, lepiej identyfikować je zaraz po wpisaniu złej wartości. W rezultacie możesz wywołać pomocną informację o przyczynie omyłki we właściwym miejscu, pozwalając użytkownikowi szybko naprawić błąd przed wysłaniem danych na serwer.
- ◆ **Walidacja po stronie serwera.** Dochodzi do niej już po przesłaniu formularza na serwer. W tym momencie zadaniem kodu serwera jest sprawdzenie

szczegółów i upewnienie się, że wszystko jest w najlepszym porządku. Pamiętaj, niezależnie od wyniku walidacji po stronie klienta walidacja serwerowa jest po prostu konieczna. To najważniejsza linia obrony w walce z hakerami, którzy mogą umyślnie manipulować przesyłanymi na serwer danymi formularza. Jeśli walidacja po stronie serwera wykryje błędne lub podejrzane wpisy, powinna zwrócić stronę z wiadomością o błędzie.

Jak łatwo zauważyć, walidacja kliencka (której przykładem jest omawiany mechanizm w HTML5) ma ułatwić pracę użytkownikom formularza, podczas gdy walidacja po stronie serwera dba o poprawność. Musisz pamiętać, że obie formy walidacji są potrzebne w niemalże każdym przypadku — chyba że chodzi o bardzo prosty formularz, w którym trudno popełnić błąd lub w którym błędy nie stanowią problemu.

Załóżmy, że na Twoim formularzu znajduje się pole, które nie powinno pozostać puste — użytkownik musi wpisać w nie jakieś dane. W HTML5 wymóg ten zadeklarujesz, wpisując w znaczniku atrybut `required`:

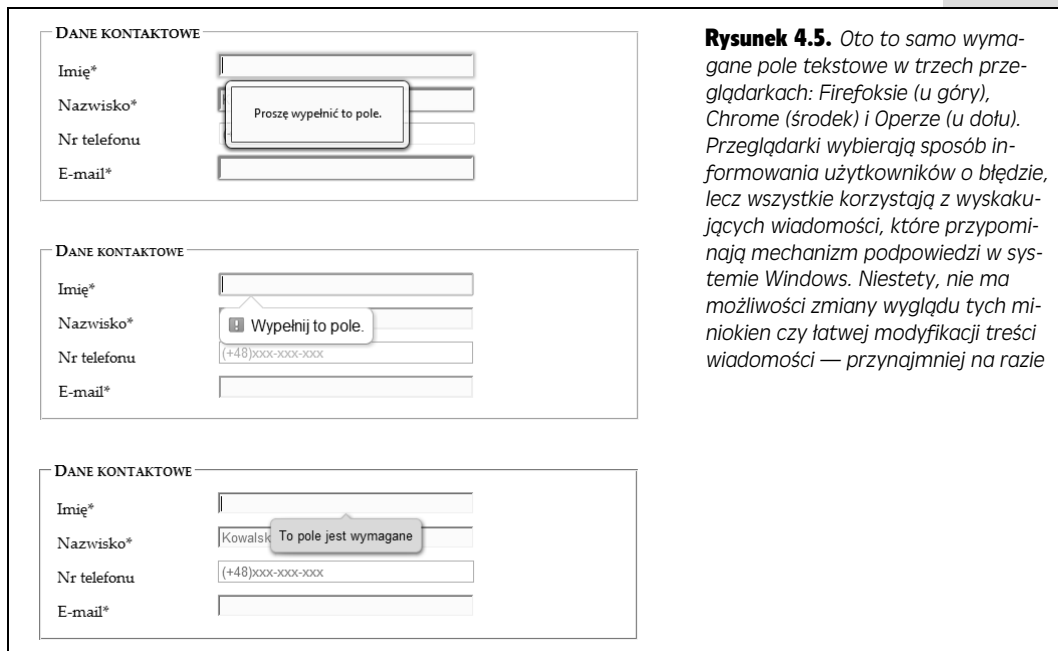
```
<label for="name">Nazwisko <em>*</em></label>  
<input id="name" placeholder="Janina Kowalska" autofocus required><br>
```

Na pierwszy rzut oka to, że dane pole jest wymagane, nie jest w ogóle widoczne. Dlatego też może warto użyć wizualnych wskazówek w rodzaju innego koloru obramowania kontrolki bądź umieszczenia gwiazdki obok wymaganego pola (tak jak na stronie formularza dla opiekunów zwierząt).

Mechanizm walidacji jest uruchamiany po kliknięciu przez użytkownika przycisku wysyłania formularza. Jak widać na rysunku 4.5, jeśli przeglądarka obsługująca formularz HTML5 zauważy, że wymagane pole jest puste, przechwyci formularz przed wysłaniem i wyświetli wyskakujące wiadomości nad błędnie wypełnionymi kontrolkami.

Czytając kolejne punkty, przekonasz się, że różne atrybuty pozwalają na zdefiniowanie odmiennych reguł sprawdzania błędów. W jednym widgecie można określić wiele reguł, a jedną regułę można nanieść na wiele kontrolek (w tym elementy `<input>` i `<textarea>`). Wszystkie warunki walidacji muszą zostać spełnione, nim formularz zostanie przesłany na serwer.

Tu pojawia się zasadnicze pytanie. Co zrobić, kiedy wprowadzone dane łamią więcej niż jedną regułę? Co się stanie, gdy np. użytkownik zapomni wypełnić kilka pól?



Rysunek 4.5. Oto to samo wymagane pole tekstowe w trzech przeglądarkach: Firefoksie (u góry), Chrome (środek) i Operze (u dołu). Przeglądarki wybierają sposób informowania użytkowników o błędzie, lecz wszystkie korzystają z wyskakujących wiadomości, które przypominają mechanizm podpowiedzi w systemie Windows. Niestety, nie ma możliwości zmiany wyglądu tych mińknień czy łatwej modyfikacji treści wiadomości — przynajmniej na razie

Otóż, do momentu gdy użytkownik kliknie przycisk wysyłający formularz, nie stanie się nic. Potem jednak przeglądarka zacznie analizować zawartość pól — od góry w dół. Gdy odnajdzie pierwszą źle użytą kontrolkę, zaprzestaje dalszej analizy i wyświetla wyskakującą wiadomość o błędzie (jeśli dodatkowo po załadowaniu strony kontrolka z niepoprawnymi danymi będzie niewidoczna, przeglądarka przewinie dokument, by znalazła się u góry jej okna). Kiedy użytkownik naprawi błąd i kliknie przycisk przesyłania, przeglądarka wykryje i oznaczy kolejną niepoprawną wartość lub prześle dane na serwer.

Uwaga: Przeglądarki przeprowadzają walidację dopiero po kliknięciu przycisku wysłania. Dzięki temu system walidacji działa wydajnie i prosto, więc sprawuje się wyśmienicie w każdych warunkach.

Niektórzy twórcy stron wolą jednak informować użytkownika o błędzie od razu po tym, jak opuści pole (np. klikając inną kontrolkę lub przyciskając klawisz *Tab*). Ta metoda walidacji sprawdza się szczególnie w długich formularzach, gdzie łatwo popełnić błędy w wielu polach. Niestety, obecnie HTML5 nie zawiera techniki dynamicznego sprawdzania, choć pewnie zostanie dodana do standardu w przyszłości. Jeśli zatem zależy Ci na natychmiastowym generowaniu informacji o błędach, napisz własny kod JavaScript lub wykorzystaj gotową bibliotekę.

Wyłączenie mechanizmu walidacji

Czasem zdarzają się sytuacje, w których trzeba wyłączyć mechanizm walidacji. Przypuśćmy, że chcesz szybko przetestować, jak dobrze identyfikuje błędy Twój kod po stronie serwera — warto wtedy dezaktywować walidację kliencką. Aby wyłączyć ją w całym formularzu, wystarczy wewnątrz znacznika `<form>` umieścić atrybut `novalidate`.

```
<form id="zooKeeperForm" action="processApplication.cgi" novalidate>
```

Innym rozwiązaniem jest dodanie przycisku przesyłania, który omija walidację. Technika ta czasem się przydaje. Możesz np. wymuszać walidację przy użyciu oficjalnego przycisku *Wyślij*, ale jednocześnie pozwolić użytkownikowi na zapisanie niedokończonych danych, aby mógł wysłać formularz później. W tym celu do znacznika `<input>` należy dołączyć atrybut `formnovalidate`:

```
<input type="submit" value="Zachowaj na później" formnovalidate>
```

Wiesz już, jak używać walidacji do wykrywania niewypełnionych pól. Pora, byś dowiedział się, jak szukać błędów w różnych typach danych.

Uwaga: Chcesz sprawdzić poprawność wpisanych liczb? Co prawda, żadna reguła walidacji nie wymusza stosowania cyfr w tekście, lecz pojawił się utworzony z myślą o liczbach nowy typ kontrolki — `number` (szczegóły na stronie 138). Niestety, jej obsługa wciąż jest niepełna.

Formatowanie kontrolek walidacyjnych

Mimo że nie da się obstyłować wiadomości walidacyjnych, można zmienić wygląd pól tekstowych, w zależności od ich stanu. Można np. nadać źle wypełnionym kontrolkom inny kolor tła — taki widжет zmieni wygląd zaraz po tym, jak przeglądarka wykryje problem.

Aby to osiągnąć, należy skorzystać z kilku nowych pseudoklas (strona 382). Oto szczególnie przydatne.

- Pseudoklasy `required` i `optional`, które nakładają formatowanie CSS w zależności od tego, czy element zawiera atrybut `required`.
- Pseudoklasy `valid` i `invalid`, które nanoszą style na kontrolki w zależności od tego, czy zawierają błędy. Pamiętaj, że przeglądarki wykryją błędne wartości dopiero wtedy, jak użytkownik spróbuje wysłać formularz, więc właściwe formatowanie nie pojawi się na stronie od razu.
- Pseudoklasy `in-range` i `out-of-range`, nakładające formatowanie arkusza na kontrolki, do których dodano atrybuty `max` i `min` (strona 138).

Jeśli np. chcesz, by wymagane pola `<input>` miały jasnożółte tło, łatwo zdefiniujesz taki wygląd przy użyciu pseudoklasy `required`:

```
input:required {  
  background-color: lightyellow;  
}
```

Nie ma przeciwwskazań, by tym kolorem wyróżnić wymagane pola z błędnymi wartościami; musisz połączyć pseudoklasy `required` i `invalid` w taki oto sposób:

```
input:required:invalid {  
  background-color: lightyellow;  
}
```

Takie ustawienia spowodują wyróżnienie pustych pól, gdyż łamią regułę narzucaną przez atrybut `required`.

Możesz wykorzystać inne sztuczki, np. połączyć pseudoklasę `validation` z pseudoklasą `focus`, użyć przesuniętego tła, w którym zawarłeś specjalną ikonkę do oznaczenia błędnych wartości itp. Warto jednak pamiętać o jednej podstawowej

zasadzie: możesz użyć tych własności, aby ulepszyć Twoją stronę, ale ze względu na kompatybilność ze starszymi przeglądarkami upewnij się, że prezentuje się dobrze bez nich.

Walidacja wyrażeń regularnych

Fundamentem najbardziej użytecznego (i złożonego) typu walidacji są tzw. **wyrażenia regularne**. JavaScript już od dawna obsługuje wyrażenia stałe, zatem dodanie tej własności do formularzy HTML to krok uzasadniony.

Wyrażenie regularne jest swego rodzaju schematem zdefiniowanym w specyficznym języku. Wyrażenia regularne są ustanawiane w celu wymuszenia poprawności pewnych wzorców tekstu. Przykładowo wyrażenie regularne może ustanawiać sekwencje wprowadzania cyfr i liter tylko tak, by tworzyły istniejący kod pocztowy, wymuszać zapisanie symbolu @ w adresie e-mail lub narzucać dodanie przynajmniej dwuliterowego rozszerzenia w adresie domeny. Spójrz na tę formułę:

```
[A-Z]{3}-[0-9]{3}
```

Kwadratowy nawias na początku wyrażenia otwiera deklarację zakresu dozwolonych znaków. Zakres `[A-Z]` umożliwia wprowadzenie dużych liter od A do Z (niestety, tylko liter alfabetu łacińskiego). Występująca po nim cyfra pomiędzy nawiasami klamrowymi określa liczbę zdefiniowanych w zakresie liter. Tak więc w omawianym przykładzie formuła `{3}` mówi, że należy wpisać trzy duże litery. Myślnik w środku wyrażenia nie ma żadnego wyjątkowego znaczenia — po prostu wskazuje, że wymaga się wstawienia go po trzech pierwszych literach. Na koniec, formuła `[0-9]` pozwala na wpisanie cyfr od 0 do 9, a fraza `{3}` wymusza podanie trzech cyfr.

Porównywanie wyrażeń zwykłych przydaje się przy przeszukiwaniu (odnajdywaniu pasujących wzorców) i walidacji (sprawdzaniu, czy wartość pasuje do formuły). W HTML5 wyrażeń zwykłych używa się głównie w tym drugim celu.

Uwaga: Maniacy wyrażeń regularnych, zwróćcie uwagę, że nie musicie korzystać z czarodziejskich znaków `^` i `$`, żeby porównać początek i koniec wartości w polu. HTML5 zakłada wprowadzenie obu tych symboli, co oznacza, że *cała* wartość musi odpowiadać wzorcowi, by została uznana za prawidłową.

Po porównaniu ze zdefiniowaną wyżej regułą następujące wartości okazały się prawidłowe:

```
QRB-001
TTT-952
LAA-000
```

Te zaś wartości zostały zidentyfikowane jako błędne:

```
qrb-001
TTT-0952
LA5-000
```

Wyrażenia regularne są zwykle o wiele bardziej złożone niż w omawianym tu przykładzie. Ich poprawne napisanie może przyprawić o ból głowy, toteż większość twórców stron korzysta z gotowych formuł, które sprawdzają interesujący ich typ danych.

Wskazówka: Aby dowiedzieć się więcej o wyrażeniach regularnych, wypróbuj dwa znakomite samouczki www.w3schools.com/js/js_obj_regexp.asp i <http://tinyurl.com/jsregex>. Gotowe formuły znajdziesz na stronie <http://regexlib.com>. Jeżeli masz aspirację zostać guru wyrażań regularnych, przeczytaj książkę *Wyrażenia regularne*, autorstwa Jeffreya Friedla (Helion, 2001).

Gdy już przygotujesz wyrażenie regularne, możesz nałożyć je na elementy `<input>` lub `<textarea>`, dodając do nich atrybut `pattern`:

```
<label for="promoCode">Promotion Code</label>
<input id="promoCode" placeholder="QRB-001" title=
"Kod promocyjny składa się, w kolejności: z trzech dużych liter,
myślnika i trzech cyfr."
pattern="[A-Z]{3}-[0-9]{3}">
```

Na rysunku 4.6. zaprezentowano, co się dzieje, jeśli złamie się narzuconą przez wyrażenie regularne regułę.

Kod promocyjny:

QRB-1234

Podaj wartość w wymaganym formacie.
Kod promocyjny składa się, w kolejności: z trzech dużych liter, myślnika i trzech cyfr.

Rysunek 4.6. Najnowocześniejsze przeglądarki (np. widoczny na rysunku Google Chrome) nie tylko wylapują błąd, lecz również pobierają tekst z atrybutu `title` i wyświetlają go wraz z komunikatem, żeby wyjaśnić użytkownikowi, co jest nie tak

Wskazówka: Przeglądarki nie przeprowadzają walidacji pustych pól. W tym przykładzie puste pole z kodem promocyjnym przejdzie walidację. Jeśli nie chcesz na to pozwolić, równocześnie zastosuj atrybuty `pattern` i `required`.

Uwaga: Wyrażenia regularne wydają się wprost stworzone do definiowania adresów e-mail i prawdą jest, że sprawdzają się w tym wyśmienicie. Na razie jednak powstrzymaj się od ich stosowania w tym celu. Specjalnie z myślą o adresach poczty elektronicznej HTML5 wprowadza nowy typ kontrolki z wbudowanym regularnym wyrażeniem (strona 137).

Własne reguły walidacji

Specyfikacja HTML5 wyróżnia zbiór gotowych właściwości JavaScriptu, które pozwolą Ci określić, czy podane wartości są poprawne (lub zmusić przeglądarkę do zrobienia tego za Ciebie). Najbardziej przydatna z nich wszystkich jest metoda `setCustomValidity()`, która umożliwia utworzenie własnych zasad sprawdzania poprawności dla określonych pól i ich współgrania z wbudowanym w HTML5 systemem walidacji.

Działa w następujący sposób. Najpierw należy sprawdzić, czy pole nie zawiera błędów. Można to osiągnąć, obsługując zdarzenie `onInput`, co nie jest niczym nowym:

```
<label for="comments"> Kiedy zdałaś/łeś sobie sprawę, że opiekun zwierząt
to zawód dla Ciebie?</label>
<textarea id="comments" oninput="validateComments(this)"></textarea>
```


W tym przykładzie zdarzenie `onInput`, uruchamiane po podaniu przez użytkownika danych, wywołuje funkcję `validateComments()`. Napisanie tej funkcji, sprawdzenie wartości elementu `<input>` i wywołanie w niej metody `setCustomValidity()` należy do Ciebie.

Jeśli pobrana wartość jest nieprawidłowa, do wywołania `setCustomValidity()` powinieneś przekazać wiadomość o błędzie. Z drugiej strony, jeżeli okaże się w pełni poprawna, do tej samej metody należy przekazać pusty typ łańcuchowy. W ten sposób usuniesz dowolną inną wiadomość, którą ustawiłeś wcześniej.

Podany niżej przykład wymusza wpisanie w pole tekstowe przynajmniej 20 znaków:

```
function validateComments(input) {
  if (input.value.length < 20) {
    input.setCustomValidity("Tekst musi być dłuższy. Podaj więcej
    ↪szczegółów.");
  }
  else {
    //Poprawna wartość. Czyści wiadomość o błędzie.
    input.setCustomValidity("");
  }
}
```

Na rysunku 4.7. pokazano, co się stanie, jeśli ktoś złamie narzuconą regułę i spróbuje wysłać formularz.

SZCZEGÓLWY DANE OSOBOWE

Wiek*

Płeć

Kiedy zdałaś/teś sobie sprawę, że opiekun zwierząt to zawód dla Ciebie?*

Miałam sen.

Rysunek 4.7. Jeżeli do metody `setCustomValidity` dodasz jako parametr własną wiadomość, przeglądarka zinterpretuje ją jak wybudowany w nią tekst. Po wysłaniu niepoprawnej wiadomości wyświetlone zostanie napisana przez Ciebie wyskakująca wiadomość

Tekst musi być dłuższy. Podaj więcej szczegółów.

Oczywiście, zaprezentowany tu scenariusz da się rozwiązać przy użyciu wyrażeń regularnych, narzucających wpisanie długich haseł. Jednak nadają się one do walidacji niektórych typów danych, a własnoręcznie skonstruowany kod może zweryfikować poprawność każdej informacji — od skomplikowanych działań algebraicznych, po szyfr połączenia z sieciowym serwerem.

Uwaga: Pamiętaj, że użytkownik Twojej strony może podejrzec wszystko, co się dzieje w kodzie JavaScript, więc nie nadaje się on do wykonywania szyfrowych algorytmów. Ty jako programista możesz np. wiedzieć, że w prawidłowym kodzie promocyjnym suma cyfr jest zawsze równa 12. Ostatnią rzeczą, którą chcesz zrobić, jest zdradzenie tej informacji wewnątrz procedury walidacji, ponieważ pomogłoby to oszustom wygenerować fałszywe kody. Taki typ walidacji powinien zachodzić na serwerze.

Obsługa mechanizmu walidacji

Twórcy przeglądarek dodają kolejne fragmenty obsługi mechanizmów walidacji, kawałek po kawałku. W rezultacie część przeglądarek obsługuje niektóre własności weryfikacji, ignorując przy tym inne. W tabeli 4.2 znajdziesz informacje o najstarszych wersjach przeglądarek, na których wszystkie z zaprezentowanych sztuczek działają.

Tabela 4.2. Obsługa mechanizmów walidacji

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android
Najstarsza wersja	10*	4	10	5 (wersja na system Windows)	10	-	-

*Aktualnie przeglądarka ta jest dostępna wyłącznie we wcześniejszej wersji Beta.

Z uwagi na to, że walidacja w HTML5 nie zastąpi tego rodzaju mechanizmów działających po stronie serwera, możesz uznać, że jest ona zaledwie dodatkiem, i jakoś znieść nierówne wsparcie. Przeglądarki, które nie implementują tego rozwiązania, jak choćby Internet Explorer 9, pozwalają ich użytkownikom wpisywać niepoprawne daty, bo przecież i tak zostaną one wykryte na etapie weryfikacji na serwerze i ukazane po ponownym załadowaniu strony.

Z drugiej strony, na Twojej witrynie mogą mieścić się złożone formularze, w których łatwo o pomyłkę, i może Ci zależeć na tym, żeby nie przysparzać użytkownikom IE niepotrzebnych frustracji. W takim przypadku masz do wyboru dwie możliwości: napisany przez Ciebie awaryjny kod walidacji lub bibliotekę JavaScriptu, która doda brakującą funkcjonalność. Twój wybór zależy od stopnia złożoności wymaganej walidacji.

Jeżeli wszystkie formularze na witrynie wymagają mechanizmu weryfikacji, warto dodać własne instrukcje sprawdzające. Korzystając ze skryptu Modernizr (patrz strona 52), możesz przetestować obsługę wielu własności HTML5. Przykładowo instrukcja `Modernizr.input.pattern` sprawdzi, czy przeglądarka rozpoznaje atrybut `pattern`:

```
if (!Modernizr.input.pattern) {
    // Wybrana przeglądarka nie przeprowadza walidacji wyrażeń regularnych.
    // Warto użyć wyrażeń regularnych JavaScriptu.
    ...
}
```

Uwaga: Atrybut `pattern` jest jedną z wielu własności testowanych przez obiekt `Modernizr.input`. Inne własności, których obsługę warto ustalić, to: `placeholder`, `autofocus`, `required`, `max`, `min` i `step`.

Oczywiście, w podanym przykładzie nie określono, kiedy należy przeprowadzać weryfikację i jak reagować na jej wyniki. Jeśli zależy Ci na tym, aby Twój system walidacji naśladował zdefiniowany przez HTML5, przeprowadzaj ją dopiero po wysłaniu formularza na serwer. W tym celu należy obsłużyć dołączane do znacznika `<form>` zdarzenie `onSubmit` funkcją, która zwróci wartość `true` (prawda — co oznacza, że formularz został wypełniony prawidłowo i przeglądarka może go przesłać dalej) lub `false` (fałsz — w formularzu znaleziono błąd i należy przerwać operację).

```
<form id="zooKeeperForm" action="processApplication.cgi"
onsubmit="return validateForm()">
```

Oto przykład prostej procedury walidacji, która wymaga wypełnienia wszystkich pól:

```
function validateForm() {
  if (!Modernizr.input.required) {
    // Jeśli przeglądarka nie rozpoznaje atrybutu required, należy samodzielnie sprawdzić,
    // czy wymagane pola zostały poprawnie wypełnione.

    // Najpierw utwórz tablicę ze wszystkimi elementami.
    var inputElements = document.getElementById("zooKeeperForm").elements;

    // Następnie przeszukaj tablicę element po elemencie.
    for(var i = 0; i < inputElements.length; i++) {

      // Sprawdź, czy element posiada atrybut required.
      if (inputElements[i].hasAttribute("required")) {
        // Jeśli znaleziono atrybut required, sprawdź, jaką ma wartość.
        // Jeśli nie zidentyfikowano go, formularz nie przechodzi walidacji,
        // a cała funkcja zwraca wartość false (fałsz).
        if (inputElements[i].value == "") {
          return false;
        }
      }
    }
    // Jeśli przeglądarka dotarła do tego punktu, walidacja się powiodła
    // i formularz może zostać wysłany.
    return true;
  }
}
```

Wskazówka: Ten blok kodu powstał w oparciu o kilka technik języka JavaScript, takich jak przeszukiwanie elementów, pętla i instrukcje warunkowe. Aby dowiedzieć się o nich więcej, zapoznaj się z treścią dodatku B.

Jeśli Twój formularz jest skomplikowany, a Ty chcesz zaoszczędzić sobie wysiłku (lecz mimo to przygotować się na przyszłość), użycie prostej łatki JavaScriptu rozwiąże wszystkie Twoje problemy. Z technicznego punktu widzenia, podejście polega na tym samym — Twoja strona sprawdzi, czy walidacja jest obsługiwana przez przeglądarkę i przeprowadzi ją ręcznie, jeśli okaże się to konieczne. Różnica polega na tym, że biblioteka zawiera już cały potrzebny kod.

Pod adresem <http://tinyurl.com/polyfills> znajdziesz całą onieśmielającą listę bibliotek, które próbują zapewnić poszukiwaną funkcjonalność. Jedną z najlepszych nosi nazwę webforms2 i jest dostępna na stronie <https://github.com/westonruter/webforms2>. Aby ją ściągnąć, odszukaj dobrze ukryty przycisk *Download* (ściągnij).

Biblioteka webforms2 implementuje wszystkie poznane tu atrybuty. Aby jej użyć, umieść jej pliki w folderze głównym witryny (lub, co jest częstszą praktyką, w podfolderze) i dodaj ścieżkę do niej na swojej stronie:

```
<head>
<title>...</title>
<script src="webforms2.js"></script>
...
</head>
```

NIEOSZLIFOWANY DIAMENT

Kilka pominiętych atrybutów kontrolek

HTML5 rozpoznaje kilka innych atrybutów, które kontrolują zachowanie przeglądarki w trakcie edytowania formularza, lecz nie są używane do walidacji. Nie wszystkie te własności działają równie dobrze na wszystkich przeglądarkach. Mimo to, warto z nimi poeksperymentować.

- ◆ **Atrybut spellcheck.** Niektóre przeglądarki pomagają uniknąć wstydu, bo sprawdzają poprawność ortograficzną słów wpisywanych w pola tekstowe. Oczywistym problemem jest to, że nie zawsze tekst musi składać się z ortograficznie prawidłowych słów, a użytkownik zniesie tylko ograniczoną liczbę podkreślonych wyrazów, nim się zdenerwuje. Ustaw wartość atrybutu `spellcheck` na `false`, jeśli nie chcesz, aby przeglądarka sprawdzała ortografię użytkownika, lub na `true`, kiedy właśnie na tym Ci zależy. Warto zauważyć, że przeglądarki różnią się pod względem domyślnej obsługi tego atrybutu — są takie, które wspierają go bez żadnej deklaracji.
- ◆ **Atrybut autoComplete.** Niektóre przeglądarki chcą Ci zaoszczędzić czasu, oferując podpowiedzi z niedawno wpisanymi w pola wartościami. Takie zachowanie nie zawsze jest pożądane — jak stwierdzono

w specyfikacji HTML5, niektóre informacje mogą być zbyt istotne (np. kod odpalenia rakiet z ładunkiem jądrowym) lub ważne tylko przez krótki czas (np. jednorazowy kod logowania banku). W takich przypadkach zaleca się nadanie atrybutowi `autocomplete` wartości `off` (wyłączony). Możesz też przedstawić ją na `on` (włączony) dla pojedynczego pola.

- ◆ **Atrybuty autocorrect i autocapitalize.** Własności te kontrolują mechanizm autokorekty i automatycznego pisania dużych liter na niektórych urządzeniach przenośnych, szczególnie na działającej na iPhone lub iPadzie przeglądarce Safari.
- ◆ **Atrybut multiple.** Projektanci stron już od dawna dodawali atrybut `multiple` do znacznika `<select>`, aby utworzyć listę elementów wielokrotnego wyboru. Teraz jednak możesz go dołączyć do znacznika `<input>`, w tym do pola wyboru pliku (używanego do ładowania go na serwer) i pola e-maila (patrz strona 137). Dzięki niemu na przeglądarkach, które obsługują tę własność, użytkownik może wybrać kilka plików lub dodać kilka adresów poczty jednocześnie.

Biblioteka webforms współpracuje z inną użyteczną łątką JavaScriptu, zwaną `html5Widgets`. Skrypt ten dodaje obsługę kilku własności, z którymi zapoznasz się później. Więcej o tym pliku dowiesz się na stronie www.useragentman.com/tests/html5Widgets. Obie przedstawione biblioteki oferują solidne wsparcie dla formularzy, jednak nawet używając ich, często natkniesz się na drobne błędy i potknięcia. Tylko czas pokaże, czy te już dość imponujące biblioteki będą usprawniane i utrzymywane.

Nowe typy znacznika input

Chyba największym dziwactwem formularzy HTML jest to, że pojedynczy element, nazwany ogólnie `<input>`, jest używany do tworzenia różnego rodzaju kontrolek: od pól wyboru, przez pola tekstowe, po przyciski. Atrybut `type` określa, co dany element `<input>` tworzy po wczytaniu strony.

Jeżeli przeglądarka natrafi na element `<input>` o typie, którego nie rozpoznaje, traktuje go jak zwykłe pole tekstowe. Oznacza to, że następujące trzy elementy zostaną wczytane tak samo na każdej przeglądarce:

```
<input type="text">
<input type="super-udziwniony-typ-znacznika">
<input>
```

HTML5 wykorzystuje ten sposób interpretacji. Jego twórcy dodali kilka nowych typów elementu `<input>`, mając świadomość, że starsze przeglądarki zinterpretują je jak zwykłe pola tekstowe. Jest to przydatne, gdy np. chcesz utworzyć pole do wprowadzania adresów e-mail; bez przeszkód możesz zadeklarować nowy typ email:

```
<label for="email">E-mail <em>*</em></label>
<input id="email" type="email"><br>
```

Jeśli otworzysz stronę w przeglądarce, która nie rozpoznaje kontrolki typu `email` (np. w Internet Explorerze 9), wygenerowane zostanie zwykłe pole tekstowe, co można zaakceptować. Przeglądarki obsługujące HTML5 są jednak trochę bardziej zmyślne. Oto, co potrafią.

- **Oferują wygodne edytowanie.** Inteligentne przeglądarki pozwalają przenieść adres e-mail z komputerowej książki adresów na kontrolkę.
- **Zapobiegają potencjalnym błędom.** Przykładowo przeglądarki mogą zignorować wpisywane litery, gdy w polu należy umieścić wyłącznie cyfry, lub odrzucić nieprawidłowe daty (bądź zmusić użytkownika do wybrania ich z minalendarza, co jest prostszą i bezpieczniejszą praktyką).
- **Przeprowadzają walidację.** Przeglądarki mogą dokonać bardziej złożonej weryfikacji po kliknięciu przycisku wysyłania formularza. Najnowocześniejsze mogą rozpoznać nieprawidłowy adres e-mail i odmówić wykonania operacji.

Specyfikacja HTML5 nie daje twórcom przeglądarek żadnych wskazówek dotyczących pierwszego punktu. Innymi słowy, przeglądarki mogą rozwiązać problem wyświetlania i edytowania różnych typów danych w dowolny racjonalny sposób, a różne przeglądarki mogą zawierać odmienne mechanizmy ułatwiające. Przykładowo przeglądarki mobilne wykorzystują brak ograniczeń, ukrywając klawisze interfejsu, które nie są potrzebne do wypełnienia wybranego pola (rysunek 4.8).

Dużo ważniejsze są mechanizmy zapobiegania i wykrywania błędów. Jako minimum przeglądarki obsługujące formularze HTML5 nie pozwalają na przesłanie formularza, jeśli zawiera dane, które łamią narzucone ograniczenia. Jeżeli więc przeglądarce nie uda się zapobiec błędom (według obserwacji z drugiego punktu listy powyżej), musi je zweryfikować, po tym jak użytkownik potwierdzi wysłanie informacji (punkt trzeci na liście).

Niestety, nie wszystkie przeglądarki spełniają te trzy wymagania. Niektóre z nich rozpoznają nowe typy danych i wprowadzają ciekawe gadżety do edytowania zawartości, lecz nie obsługują walidacji. Inne prawidłowo interpretują tylko niektóre typy danych. Szczególnych problemów przysparzają przeglądarki mobilne — co prawda, pełno w nich udogodnień edytowania, lecz nie działają na nich żadne wbudowane mechanizmy walidacji.

W tabeli 4.3 znajdziesz listę typów danych i przeglądarek, które w pełni je obsługują — co oznacza, że przerywają wysyłanie formularza, jeśli wykryją w nim nieprawidłowe dane.

Wskazówka: Tak na marginesie, łatwo zweryfikować obsługę Modernizra, wykorzystując właściwości obiektów `inputtypes`. Przykładowo wywołanie funkcji `Modernizr.inputtypes.range` zwróci wartość `true` (prawda) w przeglądarkach, które obsługują typ danych `range`.



Rysunek 4.8. Osoby korzystające z urządzeń przenośnych przy wypełnianiu formularza są na ogół pozbawione luksusu, jakim jest pełna klawiatura. iPod ułatwia życie takim użytkownikom strony, dopasowując wirtualną klawiaturę do spodziewanego typu danych. Pole tekstowe wymagające podania numeru telefonu wywołą klawiaturę numeryczną (po lewej), podczas pole z adresem e-mail spowoduje pojawienie się klawiatury z dedykowanym przyciskiem @ i mniejszym klawiszem spacji (po prawej)

Tabela 4.3. Kompatybilność przeglądarek a nowe typy elementu <input>

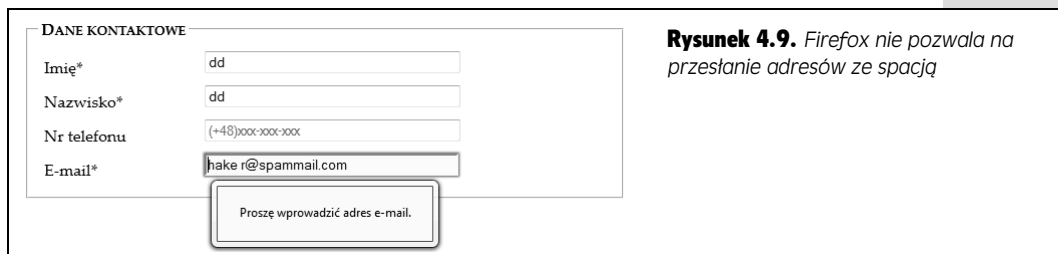
Wartość atrybutu	IE	Firefox	Chrome	Safari	Opera	Safari iOS*	Android
email	-	4	10	5 (tylko wersja Windows)	10.6	-	-
url	-	4	10	5 (tylko wersja Windows)	10.6	-	-
search*	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy
tel*	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy
number	-	-	10	5	-	-	-
range	-	-	6	5	11	-	-
datetime, date, month, week, time	-	-	10	-	11	-	-
color	-	-	-	-	11	-	-

* Standard HTML5 nie wymaga walidacji dla tego typu danych.

** Choć przeglądarka iOS nie obsługuje walidacji, jej zdolność do dynamicznego przystosowania klawiatury (rysunek 4.8) jest istotnym udogodnieniem, więc wciąż warto używać specjalistycznych typów danych.

Adresy e-mail

Pola adresów e-mail deklaruje się, nadając atrybutowi `type` wartość `email`. Zwykle poprawny adres e-mail jest ciągiem znaków (w którym nie dozwolono użycia pewnych symboli). Adres e-mail musi zawierać znak `@` oraz przynajmniej jedną kropkę. Co więcej, między nimi muszą znajdować się przynajmniej dwa znaki. Te zasady dotyczą w praktyce wszystkich adresów poczty elektronicznej. Zdefiniowanie solidnych reguł walidacji lub wyrażenia regularnego dla adresów e-mail okazuje się jednak nad wyraz trudnym zadaniem, które przerosło siły wielu webdeveloperów. To sprawia, że przeglądarki, które obsługują atrybut `e-mail` i przeprowadzają jego walidację automatycznie, są wspaniałe (rysunek 4.9).



Kontrolka typu `email` może zawierać atrybut `multiple`, który pozwala przenieść do pola kilka adresów na raz. Domyślnie adresy te zleją się w jeden ciąg tekstu, dlatego należy rozdzielić je przecinkiem.

Uwaga: Pamiętaj, że niewstawienie żadnej wartości w polu omija walidację. Jeśli chcesz zmusić użytkownika do wprowadzenia prawidłowego adresu, do kontrolki typu `email` dodaj atrybut `required` (strona 126).

Adresy URL

Pole adresów URL deklaruje się, nadając atrybutowi `type` wartość `url`. To, co konstytuuje prawidłowy adres URL, jest przedmiotem zacieklej dyskusji. W większości przeglądarek używa się jednak dość mało restrykcyjnych algorytmów walidacji. Adres URL wymaga podania przedrostka (który może być poprawny, jak `http://`, lub wymyślony, niczym `bonk://`). Można w nim też umieszczać spacje i inne symbole, z wyjątkiem dwukropka (`:`).

Niektóre przeglądarki przy wpisywaniu generują listę sugerowanych adresów, bazując na historii ostatnio odwiedzanych witryn.

Pola wyszukiwania

Pola wyszukiwania powstają po zadeklarowaniu w kontrolce atrybutu `type` o wartości `search`. Pole to ma mieścić słowa klucze, których używa się przy wyszukiwaniu haseł. Mogą być użyte do przeszukania całej sieci (tak jak w widocznej na

rysunku 4.1 wyszukiwarce Google), pojedynczej strony lub katalogu z interesującą informacją. Pole wyszukiwania zachowuje się i wygląda jak zwykłe pole tekstowe.

Na niektórych przeglądarkach (np. Safari) wyglądają one odrobinę inaczej — mają zaokrąglone rogi. Ponadto podczas wpisywania w nie treści na przeglądarce Safari i Chrome po prawej stronie kontrolki pojawia się mała ikona X, kliknięcie której spowoduje oczyszczenie pola. Poza tymi drobnymi różnicami, pola wyszukiwania są polami tekstowymi. Cała różnica tkwi w semantyce. Innymi słowy, typu danych `search` używa się, aby umożliwić przeglądarkom i oprogramowaniu dla osób niepełnosprawnych łatwiejszą identyfikację kontrolki. W rezultacie są one w stanie naprowadzić użytkownika we właściwe miejsce na stronie i zaoferować dodatkowe opcje kontroli — to drugie być może w przyszłości.

Telefon

Numery telefonów deklaruje się za pomocą typu `tel`. Numery telefonów występują w wielu formach. Niektóre składają się z cyfr, inne zawierają spacje, myślniki, znak `+` i nawiasy. Być może z powodu takiej różnorodności standard HTML5 nie wymaga od przeglądarek wykonania walidacji numerów telefonów. Trudno jednak nie życzyć sobie, aby pole typu `tel` odrzucało litery (które przyjmuje).

Obecnie jedynym powodem, dla którego warto zwrócić uwagę na kontrolkę `tel`, jest to, iż na przeglądarkach urządzeń mobilnych wywołuje ona zmodyfikowaną wirtualną klawiaturę numeryczną.

Liczby

Standard HTML5 definiuje dwa numeryczne typy danych. Typ danych `number` powstał z myślą o zwykłych liczbach.

Sposób jego zastosowania jest niemal oczywisty. Zwykłe pola tekstowe przyjmują dowolne wartości: liczby, litery, spacje, znaki interpunkcyjne i symbole używane do przedstawiania przekleństw w komiksowych dymkach. Dlatego też jedną z najczęściej wykonywanych form walidacji jest weryfikowanie, czy wprowadzana wartość jest liczbą i mieści się w narzuconym zakresie. Teraz jednak wystarczy wprowadzić kontrolkę typu `number` i przeglądarka zignoruje naciśnięcia nieliczbowych klawiszy. Oto przykład:

```
<label for="age">Wiek<em>*</em></label> <input id="age"
  type="number"><br>
```

Oczywiście, jest wiele rodzajów liczb i nie wszystkie nadają się do wpisywania każdego typu danych. Zaprezentowany wyżej kod HTML pozwala na wpisanie w polu wieku takich wartości jak 43 000 i -6 . Z myślą o takich scenariuszach zaprojektowano atrybuty `min` i `max`. W przykładzie niżej zakres wieku zdefiniowano w rozsądnych granicach od 0 do 120 lat.

```
<input id="age" type="number" min="0" max="120"><br>
```


Zazwyczaj typ `number` akceptuje wyłącznie liczby całkowite, więc liczba 30,5 zostałaby uznana za błędnie napisaną. Niektóre przeglądarki nie pozwolą nawet wpisać przecinka. Umożliwia to dodanie atrybutu `step` wraz z wartością ułamka dziesiętnego, który wskaże najmniejszą wartość po przecinku. Przykładowo podstawienie wartości 0.1 sprawi, że liczby 0, 0,1, 0,2, 0,3 itd. będą rozpoznawane jako prawidłowe (zwróć uwagę, że w deklaracji użyto angielskiego formatowania ułamka dziesiętnego; niestety, obecnie tylko przeglądarka Chrome poprawnie interpretuje znak przecinka w ułamku — w innych przeglądarkach, wpisując ułamki, należy posługiwać się kropką). Jednak próba przesłania liczby 0,15 zakończy się pojawieniem znanego Ci już wyskakującego komunikatu o błędzie. Domyślnie atrybut `step` ma wartość 1.

```
<label for="weight">Waga (w kilogramach)</label>
<input id="weight" type="number" min="0.1" max="650" step="0.1"
value="70"><br>
```

Jak widać na rysunku 4.10, atrybut `step` wpływa też na działanie przycisków po prawej stronie kontrolki.



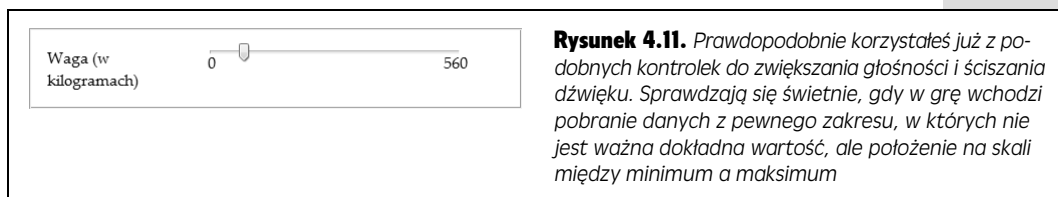
Rysunek 4.10. Najnowsze przeglądarki dotychczas przyciski do pól numerycznych. Po każdym kliknięciu górnego przycisku liczba w polu jest zwiększana o zdefiniowaną atrybutem `step` wartość. Podobnie kliknięcie dolnego przycisku spowoduje zmniejszenie liczby

Suwak

Innym liczbowym typem danych w HTML5 jest typ `range`. Podobnie jak typ `number`, może on reprezentować liczby całkowite i ułamki. Obsługuje również te same atrybuty zakresu dozwolonych wartości (czyli `min` i `max`). Oto przykład:

```
<label for="weight">Waga (w kilogramach)</label>
<input id="weight" type="range" min="0" max="560" value="70"><br>
```

Różnica tkwi w sposobie reprezentowania informacji. Zamiast po prostu wpisywać wartość w polu tekstowym, przeglądarki nowej generacji wczytują suwak wartości (rysunek 4.11).



Rysunek 4.11. Prawdopodobnie korzystałeś już z podobnych kontrolki do zwiększania głośności i ściszenia dźwięku. Sprawdzają się świetnie, gdy w grę wchodzi pobranie danych z pewnego zakresu, w których nie jest ważna dokładna wartość, ale położenie na skali między minimum a maksimum

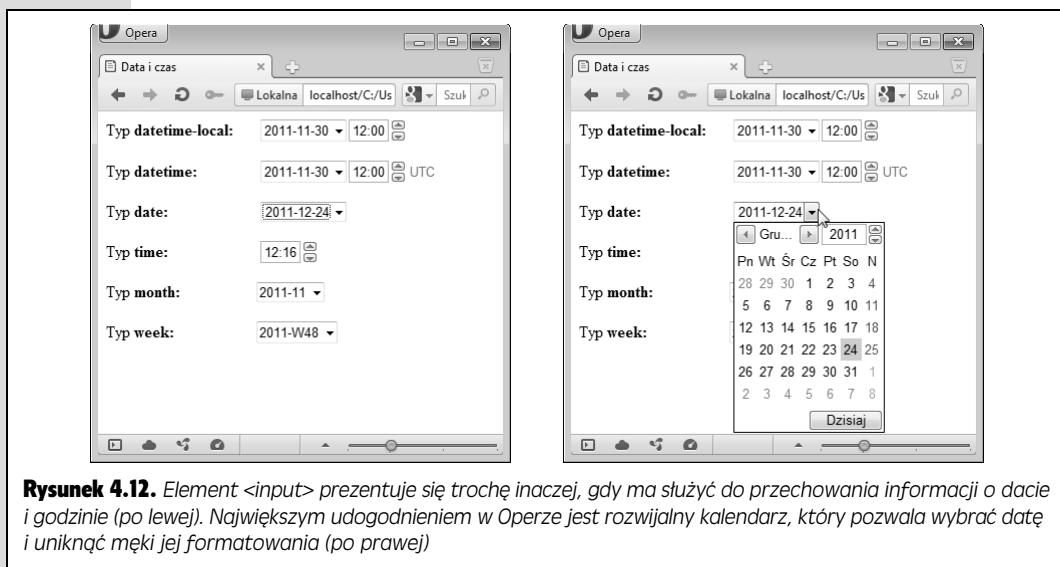
Wartość na tej kontrolce ustawia się, przesuwając suwak w granicach zdefiniowanego zakresu. Przeglądarki, które rozpoznają typ `range`, nie zwracają żadnych informacji o wartości. Jeśli te dane są Ci w jakimś celu potrzebne, musisz dodać kilka linii kodu JavaScript, który zareaguje na zmianę położenia suwaka (np. poprzez obsługę zdarzenia `onChange`) i wyświetli informacje w pobliżu. Oczywiście,

warto sprawdzić, czy wybrana przeglądarka obsługuje typ `range` (choćby przy użyciu narzędzie Modernizr). Jeśli okaże się, że tak nie jest, nie trzeba implementować kodu, gdyż ustawiona wartość pokaże się w zwykłym polu tekstowym.

Czas: daty i godziny

HTML5 definiuje kilka typów danych. Przeglądarki, które potrafią je zidentyfikować, wyświetlają wygodne kalendarze, z których da się wybrać datę. W rezultacie nie tylko unikniesz zamieszania z formatowaniem czasu, ale zapobiegiesz też przypadkowemu (lub celowemu) wybraniu daty, która nie może istnieć. Inteligentne przeglądarki przyszłości pójdą krok dalej — zintegrują pole z osobistym kalendarzem.

Obecnie, pomimo oczywistych zalet, typy dat nie cieszą się powszechnym wsparciem. Opera jest jedyną przeglądarką, która wyświetla rozwijane kalendarze (rysunek 4.12). Chrome zapewnia jedynie podstawową obsługę: wyświetla pola tekstowe z przyciskami, takie same jak dla typu `number`, i odrzuca zły format daty.



Rysunek 4.12. Element `<input>` prezentuje się trochę inaczej, gdy ma służyć do przechowania informacji o dacie i godzinie (po lewej). Największym udogodnieniem w Operze jest rozwijalny kalendarz, który pozwala wybrać datę i uniknąć męki jej formatowania (po prawej)

Wskazówka: Jeśli zdecydujesz się użyć jednego z typów dat, warto zastosować wypełnienie, w rodzaju opisanej wcześniej biblioteki `html5Widgets` (www.useragentman.com/tests/html5Widgets). W końcu osobom używającym przeglądarek innych niż najnowsza Opera czy Chrome łatwo wpisać datę w złym formacie, a walidacja tego typu danych oraz podawanie wskazówek jest trudnym i niewdzięcznym zadaniem. Z tego względu napisane w języku JavaScript kontrolki dat cieszą się takim wzięciem w sieci.

W tabeli 4.4 znajdziesz opis sześciu formatów dat HTML5.

Wskazówka: Przeglądarki, które obsługują typy dat, wspierają również użycie atrybutów `min` i `max`. Możesz więc zdefiniować maksymalną lub minimalną datę, pod warunkiem że zapiszesz ją we właściwym formacie. Gdybyś chciał ograniczyć wartości w polu dat, mógłbyś wpisać `<input type="date" min="2012-01-01" max="2012-12-31">`.

Tabela 4.4. Typy dat

Wartość atrybutu type	Opis	Przykład
date	Data w formacie YYYY-MM-DD.	25 stycznia 2012 — 2012-01-25
time	Dokładny czas na 24-godzinnym zegarze w formacie GG:mm:ss.ss.	14:35 (i 52 sekundy) — 14:35 lub 14:35:50.2
datetimelocal	Data i godzina oddzielone dużą literą T (format: YYYY-MM-DDTHH:mm:ss).	25 stycznia 2012, 14:35 — 2012-01-15T14:35
datetime	Data i godzina z różnicą w strefie czasowej. Przyjmuje taki sam format jak w elemencie <time> (YYYY-MM-DDTHH:mm:ss-HH:mm) omówionym na stronie 93.	25 stycznia 2012 r., 14:35 w Warszawie — 2012-01-15T14:35+02:00
month	Rok i miesiąc w formacie YYYY-MM.	Styczeń 2012 r. — 2012-01
week	Rok i numer tygodnia w roku. Zauważ, że w roku są 52 lub 53 tygodnie.	Drugi tydzień 2012 roku — 2012-W02

Kolor

Kontrolka kolorów definiowana jest za pomocą typu `color`. Ten typ danych jest bardzo ciekawym, choć rzadko używanym gadżetem, który pozwala gościowi strony pobrać z niej kolor przy użyciu wysuwanej pipety kolorów, która wygląda podobnie jak w programie Paint, w systemie Windows. Aktualnie tylko Opera zapewnia pełną obsługę. W innych przeglądarkach należy własnoręcznie wpisać szesnastkowy kod koloru. Można też posłużyć się biblioteką `html5Widgets`, do ściągnięcia ze strony www.useragentman.com/tests/html5Widgets.

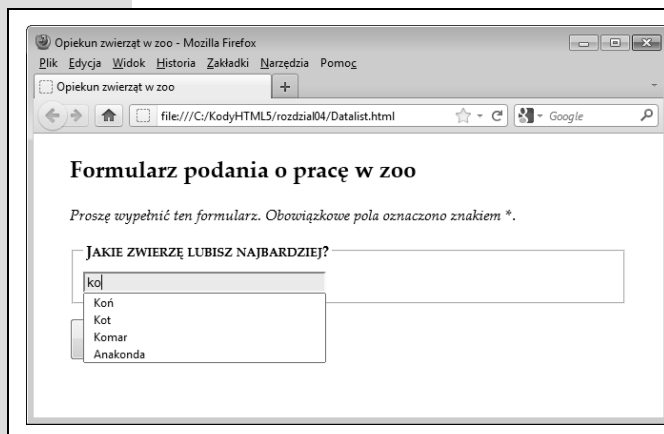
Nowe elementy

Do tej pory dowiedziałeś się, w jakim stopniu HTML5 rozszerza funkcjonalność formularzy dzięki nowym własnościom walidacji oraz jak wiele nowych typów kontrolek do niego dodano. Są to najbardziej praktyczne i najszerzej obsługiwane opcje, lecz nie stanowią jedynych nowości w formularzach HTML5.

W HTML5 wprowadzono też kilka innych innowacji, które wypełniają luki w zapotrzebowaniu i dodają większą funkcjonalność. Nowe elementy, o których tu mowa, pozwolą Ci osadzać na stronie rozwijane listy z sugestiami, paski stanu, paski narzędzi i wiele innych komponentów. Cały szkopuł w tym, że starsze przeglądarki nigdy nie będą ich rozpoznawać, a nawet w nowych ich wersjach są wprowadzane bardzo niechętnie, z uwagi na częste zmiany w specyfikacji. W konsekwencji, elementy te zaliczają się do *najgorzej* obsługiwanych własności w tym rozdziale. Najprawdopodobniej zechcesz zobaczyć je w akcji, lecz będziesz musiał poczekać z ich wykorzystaniem, chyba że nie przeszkadza Ci bycie pionierem w krainie błędów i niekompatybilności.

Sugerowane odpowiedzi i element <datalist>

Element <datalist> umożliwia zespolenie listy z sugerowanymi odpowiedziami ze zwykłym polem tekstowym. Dzięki temu użytkownik formularza może wybrać z listy dostępnych opcji lub wpisać własną (rysunek 4.13).



Rysunek 4.13. W miarę jak piszesz, przeglądarka wyświetla potencjalne odpowiedzi. Wpisanie np. liter „ko” spowoduje wywołanie wszystkich zwierząt z tą sekwencją liter w nazwie (niekoniecznie od początku)

Listy danych używa się wspólnie ze standardowym polem tekstowym. Załóżmy, że wprowadziłeś na stronę taki element <input>:

```
<legend>Jakie zwierzę lubisz najbardziej?</legend> <input id="favoriteAnimal">
```

Aby dodać rozwijalną listę sugestii, zadeklaruj element <datalist>. Teoretycznie, możesz umieścić ten komponent w dowolnym miejscu. Jest to spowodowane tym, że element datalist nie pojawia się na stronie — zamiast tego dostarcza danych używanych przez kontrolkę <input>. Rozsądnie jednak będzie umieścić element <datalist> w pobliżu pola tekstowego, z którym ma być zespolony. Oto przykład:

```
<datalist id="animalChoices">
  <option label="Koń" value="koń">
  <option label="Zebra" value="zebra">
  <option label="Kot" value="kot">
  <option label="Komar" value="komar">
  <option label="Gąsienica" value="gąsienica">
  <option label="Anakonda" value="anakonda">
  <option label="Człowiek" value="człowiek">
  <option label="Słoń" value="słoń">
  <option label="Gnu" value="gnu">
  <option label="Gołąb" value="gołąb">
  <option label="Krab" value="krab">
</datalist>
```

Podobnie jak w tradycyjnym elemencie <select>, w komponencie <datalist> zagnieżdża się znaczniki <option>. Każdy element <option> oznacza odrębną sugestię wyświetlaną pod polem tekstowym w trakcie wypełniania. Atrybut label pokazuje tekst, który pojawi się pod kontrolką, podczas gdy własność value — wartość, która zostanie wysłana na serwer, jeśli użytkownik wybierze jej opcję z listy. Lista danych jest całkowicie niewidoczna. Nim zaczniesz wyświetlać sugestie, trzeba ją załączyć do pola tekstowego, dodając do niego atrybut list i nadając mu wartość atrybutu id znacznika <datalist>.

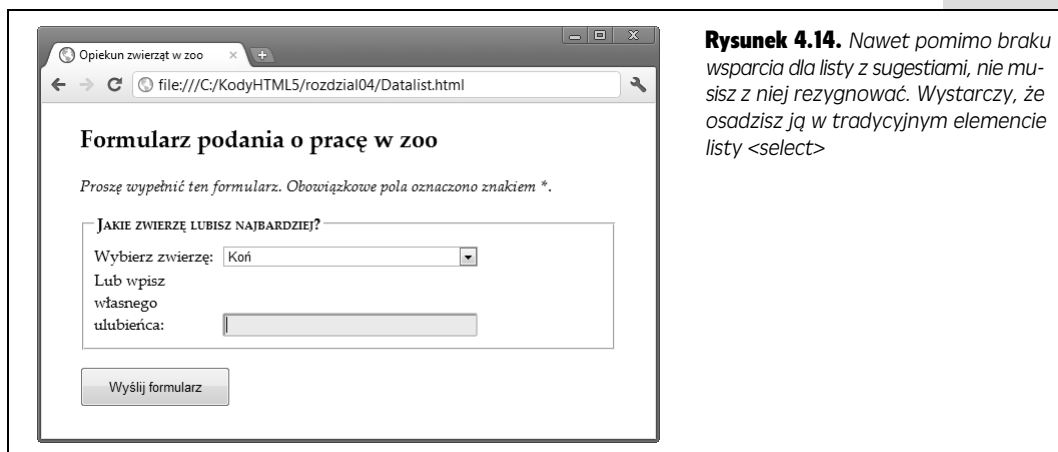
```
<input id="favoriteAnimal" list="animalChoices">
```

W przeglądarce, która rozpoznaje element `<datalist>` — obecnie są to wyłącznie Opera 10 i Firefox 4 (lub ich nowsze wydania) — użytkownicy ujrzą stronę z rysunku 4.13. Inne przeglądarki zignorują atrybut `list` i cały kod `<datalist>`, co sprawi, że lista sugestii stanie się bezużyteczna.

Istnieje jednak pewna awaryjna sztuczka, który zmusi wszystkie przeglądarki do właściwego zachowania. Polega ona na umieszczeniu dodatkowych treści wewnątrz komponentu `<datalist>`. Sprawdza się wyśmienicie, gdyż przeglądarki, które obsługują znacznik `<datalist>`, zauważają wyłącznie osadzony wewnątrz niego element `<option>` i ignorują inne treści. Oto ulepszony kod, który wykorzystuje to zachowanie. Pogrubiony kod oznacza fragmenty, które zostaną zignorowane przez przeglądarki rozpoznające element `<datalist>`.

```
<legend>Jakie zwierzę lubisz najbardziej?</legend>
<datalist id="animalChoices">
  <span class="Label">Wybierz zwierzę:</span>
  <select id="favoriteAnimalPreset">
    <option label="Koń" value="koń">
    <option label="Zebra" value="zebra">
    <option label="Kot" value="kot">
    <option label="Komar" value="komar">
    <option label="Gąsienica" value="gąsienica">
    <option label="Anakonda" value="anakonda">
    <option label="Człowiek" value="człowiek">
    <option label="Słoń" value="słoń">
    <option label="Gnu" value="gnu">
    <option label="Gołąb" value="gołąb">
    <option label="Krab" value="krab">
  </select>
<br>
<span class="Label">Lub wpisz własnego ulubieńca: </span>
</datalist>
<input list="animalChoices" id="favoriteAnimal">
```

Po usunięciu pogrubionego kodu otrzymałbyś poprzednią zawartość dokumentu. Przeglądarki, które rozpoznają element `<datalist>`, wygenerują pojedyncze pole tekstowe i rozwijaną listę sugestii, podobną do tej z rysunku 4.13. Na innych przeglądarkach dodane komponenty umożliwią użytkownikom wpisanie zwierzaka lub wybranie go z listy (rysunek 4.14).



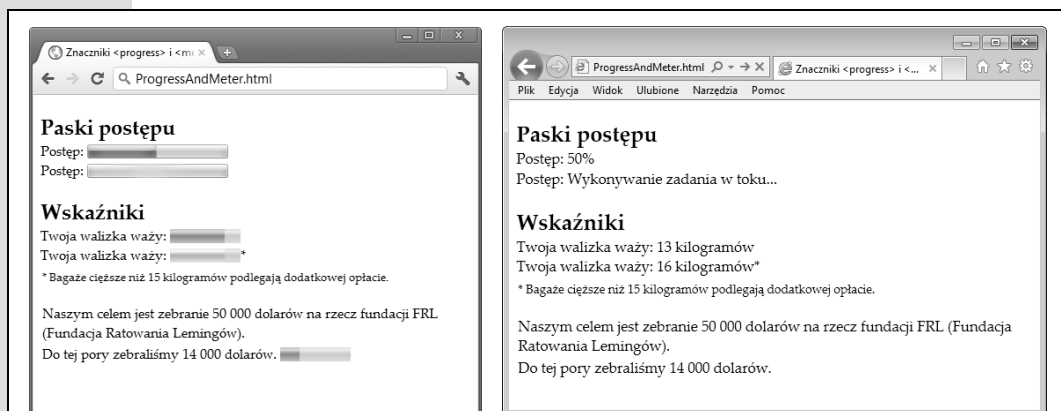
Rysunek 4.14. Nawet pomimo braku wsparcia dla listy z sugestiami, nie musisz z niej rezygnować. Wystarczy, że osadzisz ją w tradycyjnym elemencie listy `<select>`

Rozwiązanie nie jest doskonałe. Przed wysłaniem danych na serwer będziesz musiał sprawdzać zarówno wartości wybrane z tradycyjnej listy (tutaj o identyfikatorze `favouriteAnimalPreset`), jak i z pola tekstowego (identyfikator `favoriteAnimal`). Mimo tej małej wady, możesz łatwo dodać nowoczesne udogodnienie, bez zostawiania większości swoich użytkowników w tyle.

Uwaga: Na początku element `<datalist>` miał posiadać własność, która umożliwiałaby pobranie opcji z zewnętrznego źródła, np. serwera, który mógłby je wyciągnąć z bazy danych. Być może własność ta zostanie dodana w kolejnych wersjach HTML, lecz na razie podobną funkcję da się zapewnić za pomocą napisanego w JavaScriptcie obiektu `XMLHttpRequest` (strona 322).

Pasek stanu i miernik

Elementy `<progress>` i `<meter>` są graficznie bardzo podobne, ale używa się ich w innych celach (rysunek 4.15).



Rysunek 4.15. Elementy `<progress>` i `<meter>` są przydatnymi udogodnieniami, przynajmniej na przeglądarkach, które je rozpoznają (po lewej). Pozostałe przeglądarki po prostu wyświetlają zapisaną w nich wartość w formie tekstu (po prawej)

Znacznik `<progress>` wskazuje postęp jakiegoś zadania. Jest to szary komponent w pewnej części wypełniony pulsującym, zielonym paskiem. Element `<progress>` przypomina bardzo pasek stanu, z którymi spotkałeś się wcześniej (przypomnij sobie choćby pasek wyświetlany w systemie Windows w trakcie przesyłania plików), choć jego dokładny wygląd zależy od przeglądarki, na której otwarto stronę.

Element `<meter>` oznacza wartość w znanym zakresie. Po załadowaniu prezentuje się bardzo podobnie do komponentu `<progress>`, choć zawarty w nim zielony pasek ma trochę inny odcień i nie pulsuje. W zależności od przeglądarki pasek miernika może zmieniać kolor, gdy nadana mu wartość jest „za mała” lub „zbyt duża” — np. w drugim przypadku Chrome zmienia barwę paska na żółtą. Najważniejsza różnica pomiędzy dwoma znacznikami wynika z ich odmiennego interpretowania przez przeglądarkę.

Uwaga: Technicznie rzecz ujmując, nowe elementy `<meter>` i `<progress>` nie muszą się znajdować w obrębie formularza. W zasadzie nie są one nawet kontrolkami (ponieważ nie pobierają żadnych danych od użytkownika). Mimo to, oficjalne wytyczne HTML5 zaliczają je do grupy komponentów formularza, gdyż oba te znaczniki wyglądają jak widżety i służą do wyświetlania informacji w graficznej formie.

Aktualnie elementy `<progress>` i `<meter>` działają w Chrome (wersja 9. i późniejszej), Operze (wersja 11. i nowsze) oraz Safari (wersja 5.1 lub nowsze). Firefox rozpoznaje znacznik `<progress>` (ale nie `<meter>`), począwszy od wersji 6. Tylko Internet Explorer nie obsługuje obu.

Obydwoma komponentami łatwo się posługiwać. Najpierw przyjrzyj się znacznikowi `<progress>`. Przyjmuje on atrybut `value`, którego wartość wyrażona w ułamku dziesiętnym (od 0 do 1) ustawia procent postępu operacji (a tym samym długość zielonego wypełnienia). Można np. ustawić wartość parametru `value` na 0.25, co odpowiada 25% ukończenia zadania.

```
<progress value="0.25"></progress>
```

Możesz też wykorzystać atrybut `max` i zmienić skalę paska. Gdybyś ustawił maksymalną wartość na 200, wartość parametru `value` musiałaby się zmieścić między 0 a 200. Dlatego też wpisanie do niego liczby 50 spowoduje wypełnienia paska stanu do 25%, tak jak w poprzednim przykładzie.

```
<progress value="50" max="200"></progress>
```

Ustawienia skali są kwestią wygody. Użytkownik strony nie widzi wartości paska.

Uwaga: Element `<progress>` po prostu powoduje wyświetlenie cieniowanego paska postępu. Sam *nie robi nic*. Przypuśćmy, że chcesz użyć paska stanu do pokazania stopnia ukończenia pewnego zadania (np. przy użyciu omówionych na stronie 359 własności pracowników). Do Ciebie należy napisanie kodu, który wykryje element `<progress>` i zmieni jego wartość.

Przeglądarki, które nie rozpoznają elementu `<progress>`, po prostu go ignorują. Częściowym rozwiązaniem tego problemu jest podanie awaryjnej wartości, co robi się w taki oto sposób:

```
<progress value="0.25">25%</progress>
```

Pamiętaj, że zawarta między znacznikami treść nie pojawi się na przeglądarkach, które *rozpoznają* ten komponent.

Paska postępu można użyć również w inny sposób. Można wyświetlić *nieoznaczony* pasek postępu, który wskazuje, że zadanie jest wykonywane, lecz nie wiadomo, kiedy się zakończy (w takim przypadku myśl o pasku jak o wymyślnym sposobie przekazania wiadomości „ładowanie w toku”). Nieoznaczony pasek postępu wygląda jak zwykły szary słupek, po którym co pewien czasu porusza się zielony błysk, od lewej do prawej. Aby utworzyć taki element, po prostu pomiń atrybut `value` w deklaracji:

```
<progress>Wykonywanie zadania w toku... </progress>
```

Element `<meter>` korzysta z podobnego modelu, lecz wskazuje pewną miarę. Komponent ten czasem nazywany jest **miarką** lub **wskaznikiem**. Często dokładna wartość wskaźnika będzie odpowiadać rzeczywistym pomiarom (np. sumom pieniężnym, liczbie dni, wadze itd.). Sposób, w jaki znacznik `<meter>` prezentuje te

informacje, można zmodyfikować z wykorzystaniem wartości maksymalnych i minimalnych (czyli atrybutów `min` i `max`).

```
Twoja walizka waży: <meter min="2" max="40" value="">13
kilogramów</meter>
```

Tak jak ma to miejsce w elemencie `<progress>`, zawartość elementu `<meter>` jest wyświetlana tylko wtedy, gdy przeglądarka go nie rozpoznaje. Oczywiście, mogą zdarzyć się sytuacje, kiedy warto pokazać dokładną liczbę, którą komponent ma odzwierciedlać. W takim przypadku należy ją po prostu dodać na stronie samemu — nie trzeba wtedy polegać na awaryjnej treści. W następującym przykładzie zaprezentowano, jak to zrobić. Metoda ta pozwala zawrzeć wszelkie informacje i dodaje pasek wskaźnika na obsługujących go przeglądarkach.

```
<p>Naszym celem jest zebranie 50 000 dolarów na rzecz fundacji FRL
↳(Fundacja Ratowania Lemingów).</p>
<p>Do tej pory zebraliśmy 14 000 dolarów. <meter max="50000"
value="14000"></meter>
```

Element `<meter>` jest również wyposażony w gadzety, które umożliwiają oznaczenie zbyt wysokich lub niskich wartości, przy jednoczesnym prawidłowym ich wyświetlaniu. W tym celu używa się atrybutów `low` i `high`. Przykładowo wartość parametru `value`, która przekracza liczbę w atrybucie `high` (ale jest mniejsza od maksimum), jest uznana za zbyt wysoką, lecz dozwoloną. Podobnie wartość poniżej dołączonej do atrybutu `low` liczby zostanie oznaczona jako zbyt niska:

```
Twoja walizka waży: <meter min="2" max="40" high="15" value="16">16
↳kilogramów</meter>* <p><small>* Bagaże cięższe niż 15 kilogramów
↳podlegają dodatkowej opłacie. </small></p>
```

Przeglądarki mogą wykorzystać te dodatkowe atrybuty lub nie. I tak Chrome zmienia kolor paska na żółty dla zbyt wysokich wartości (jak w poprzednim przykładzie). Nie reaguje jednak przy zbyt małych liczbach. Na koniec, możesz oznaczyć pewną liczbę jako optymalną wartość za pomocą atrybutu `optimum`, lecz nie zmieni to sposobu ładowania komponentu na stronie we współczesnych przeglądarkach.

Podsumowując, można stwierdzić, że znaczniki `<progress>` i `<meter>` są drobnymi fajerkami, które staną się użyteczne, gdy więcej przeglądarek zacznie je rozpoznawać.

Paski narzędzi i menu — znaczniki `<command>` i `<menu>`

Prawdopodobnie wśród wszystkich własności, które nie zostały zaimplementowane, elementy `<command>` i `<menu>` są najważniejsze. Podstawą tych dwóch elementów jest koncepcja odzwierciedlenia akcji uruchamianych przez użytkownika (w komponencie `<command>`) i zgrupowania ich w jednym miejscu (przy użyciu znacznika `<menu>`). W zależności od podejścia do problemu oraz użytych sztuczek formatowania, element `<menu>` mógłby przekształcić się we wszystko: od paska narzędzi osadzonego obok krawędzi okna, po wyskakujące menu, które pojawia się, gdy użytkownik kliknie gdzieś na stronie. Niestety, obecnie żadna przeglądarka nie rozpoznaje tych komponentów, więc przyjdzie poczekać, nim będzie Ci dane sprawdzić, czy będą tak wspaniałe, jak spodziewają się twórcy stron.

Edytor HTML na stronie

Jak przekonałeś się w rozdziale 1., twórcy HTML5 obrali sobie za cel brukowanie ścieżek — innymi słowy, włączenie do specyfikacji HTML5 często wykorzystywanych przez współczesnych deweloperów, aczkolwiek niestandardowych, rozwiązań. Chyba najlepszym przykładem tego procesu są dwa dziwaczne atrybuty `contentEditable` i `designMode`, które pozwalają przekształcić zwykłą przeglądarkę w edytor HTML-u.

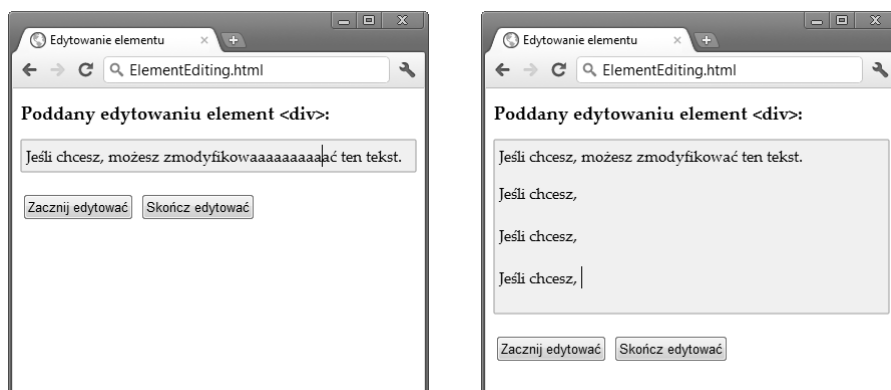
Te dwa atrybuty nie są nowością. W rzeczywistości dodano je do Internet Explorera 5, w ciemnych wiekach Internetu. W owym czasie większość twórców stron uznała je za sieciowe rozszerzenie przeznaczone wyłącznie dla systemu Windows i dlatego je zignorowano. Z biegiem czasu coraz więcej przeglądarek obrało praktyczne, choć niepozabawione wad, podejście do edytowania kodu HTML. Obecnie wszystkie przeglądarki na komputery stacjonarne obsługują te atrybuty, mimo że nigdy wcześniej nie były częścią żadnego oficjalnego standardu.

Edytowanie zawartości za pomocą `contentEditable`

Pierwszym narzędziem do dynamicznego edytowania HTML-u jest parametr `contentEditable`. Wystarczy wpisać parametr w dowolnym znaczniku, aby można było go edytować.

```
<div id="editableElement" contentEditable>Jeśli chcesz, możesz  
zmodyfikować ten tekst.</div>
```

W pierwszej chwili zapewne nie zauważysz żadnej różnicy. Jeżeli jednak załadujesz stronę z tym kodem i klikniesz wewnątrz elementu `<div>`, pojawi się kursor pisania tekstu, zwany też **karetką** (rysunek 4.16).



Rysunek 4.16. Po kliknięciu edytowalnego obszaru będziesz mógł się po nim poruszać, używając klawiszy strzałek, oraz kasować tekst lub wpisywać nową treść (po lewej). Możesz też zaznaczyć słowa, przytrzymując klawisz `Shift` i dowolnie je kopiować, wycinać i wklejać (po prawej). Przypomina to pisanie w edytorze tekstu, z tą różnicą, że użytkownik nie może wyjść poza obręb elementu `<div>` i zmodyfikować treść reszty strony

PRZYSPIESZAMY

Kiedy warto implementować funkcję edytowania HTML-u?

Nim wypróbujesz funkcję edytowania HTML-u, warto zadać pytanie, czemu ma ona służyć. Należy pamiętać, że edytowanie HTML-u jest specjalistyczną własnością, której użycie jest kontrowersyjne. Umieszczanie jej na stronie ma sens, jeśli chcesz pozwolić użytkownikowi szybko i bezboleśnie zmodyfikować treść, aby np. dodać posty na blogu, wpisać własne recenzje, umieścić dozwolone reklamy lub wpisywać wiadomości przeznaczone dla innych użytkowników.

Jeśli nawet okaże się, że taka opcja jest potrzebna, atrybuty `contentEditable` i `designMode` mogą nie być najlepszym wyborem. Wynika to z prostego faktu, że żaden

z tych parametrów nie oferuje opcji dostępnych w prawdziwych narzędziach projektowania stron, w rodzaju zmieniających kod instrukcji, możliwości edytowania dokumentu źródłowego HTML, sprawdzania pisowni itp. Z drugiej strony, posługując się nimi, przy odrobinie wysiłku *da się* skonstruować o wiele bardziej złożony edytor. Jeśli jednak naprawdę potrzebujesz funkcji edytowania strony, może lepiej będzie użyć gotowego rozwiązania, które po prostu do niej dołączysz? Informacje o najpopularniejszych opcjach znajdziesz na blogu <http://ajaxian.com/archives/richtexteditors-compared>.

W tym przykładzie oznaczony atrybutem `contentEditable` element `<div>` zawiera wyłącznie tekst. Nic nie stoi na przeszkodzie, aby umieścić w nim inny komponent. W zasadzie ten element `<div>` mógłby objąć całą stronę, co sprawi, że stanie się otwarta na wprowadzanie zmian przez użytkowników. Idąc dalej tym tropem, atrybut `contentEditable` możesz zadeklarować w kilku komponentach, dzięki czemu na stronie pojawi się wiele sekcji, jakie można będzie do woli modyfikować.

Wskazówka: Niektóre przeglądarki rozpoznają pewne kombinacje klawiszy. Przykładowo w przeglądarce Internet Explorer istnieje opcja pogrubienia, podkreślenia lub nałożenia kursywy na tekst, aktywowana wciśnięciem klawiszy (odpowiednio) — `Ctrl+B`, `Ctrl+U` oraz `Ctrl+I`. W podobny sposób, naciskając klawisze `Ctrl+Z`, można cofnąć ostatnio wykonaną operację w Firefoksie. Wszystkie te skróty działają w przeglądarce Chrome. Jeżeli chcesz dowiedzieć się więcej o komendach formatowania i poznać metodę tworzenia paska narzędzi, który mógłby je obsługiwać, przeczytaj dwuczęściowy artykuł opublikowany na witrynie Opery, pod adresami <http://tinyurl.com/htmlEdit1> i <http://tinyurl.com/htmlEdit2>.

Atrybut `contentEditable` rzadko deklaruje się bezpośrednio na stronie. Zamiast tego włącza się go przy użyciu JavaScriptu i wyłącza po dokonaniu zmian na stronie. Dwie poniższe funkcje powstały właśnie w tym celu:

```
function startEdit() {
    // Włącza opcję edytowania wybranego elementu.
    var element = document.getElementById("editableElement");
    element.contentEditable = true;
}

function stopEdit() {
    // Przywracaabrany element do poprzedniego stanu.
    var element = document.getElementById("editableElement");
    element.contentEditable = false;

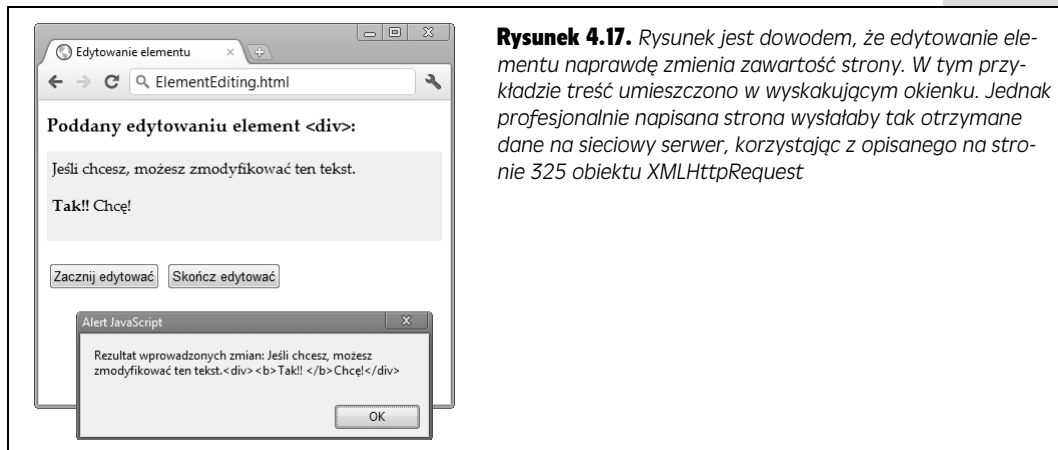
    // Wyświetla kod w polu wiadomości.
    alert("Rezultat wprowadzonych zmian: " + element.innerHTML);
}
```

Oto dwa przyciski, w których je zaimplementowano:

```
<button onclick="startEdit()">Zacznij edytować</button>
<button onclick="stopEdit()">Skończ edytować</button>
```

Upewnij się, że nie umieściłeś przycisków w edytowanym obszarze — jeśli tak zrobiłeś, przestaną reagować na zdarzenia i nie będą mogły zostać użyte do uruchamiania kodu.

Na rysunku 4.17 przedstawiono wynik zmian po zmodyfikowaniu elementu i pogrubieniu tekstu (wszystko dzięki kombinacji klawiszy *Ctrl+B*).



Rysunek 4.17. Rysunek jest dowodem, że edytowanie elementu naprawdę zmienia zawartość strony. W tym przykładzie treść umieszczono w wyskakującym okienku. Jednak profesjonalnie napisana strona wysłaby tak otrzymane dane na sieciowy serwer, korzystając z opisanego na stronie 325 obiektu XMLHttpRequest

Uwaga: W sposobie działania funkcji dynamicznego edytowania kodu HTML na stronie, można w zależności od przeglądarki wyróżnić wiele subtelnych różnic. Przykładowo wciśnięcie kombinacji klawiszy *Ctrl+B* na przeglądarce Chrome spowoduje dodanie elementu ``, podczas gdy na Internet Explorerze będzie to komponent ``. Do podobnej sytuacji dochodzi po naciśnięciu klawisza *Enter* w celu dodania nowej linii lub klawisza *Backspace*, aby usunąć znacznik. Jednym z powodów standaryzacji własności dynamicznego edytowania dokumentów HTML5 jest chęć narzucenia spójnego modelu jej działania.

Edytowanie strony za pomocą atrybutu `designMode`

Własność `designMode` działa podobnie do parametru `contentEditable`, ale w odróżnieniu od niego pozwala na modyfikację całej strony. Może się to wydać trochę niedorzeczne. W końcu jeśli cała strona ma być poddana temu procesowi, jak użytkownik uzyska dostęp do kontrolujących edytowanie przycisków? Rozwiązaniem jest umieszczenie modyfikowanej strony wewnątrz elementu `<iframe>`, który działa jak pole edycji tekstu (rysunek 4.18).

Cały użyty w tym przykładzie kod jest wyjątkowo prosty. Oto zawartość elementu `<body>` tej strony:

```
<h1>Poddawana modyfikacji strona</h1>
<iframe id="pageEditor" src="ApocalypsePage_Revised.html"></iframe>
<div>
  <button onclick="startEdit()">Zacznij edytować</button>
  <button onclick="stopEdit()">Skończ edytować</button>
</div>

<h1>Zmodyfikowany kod HTML</h1>
<div id="editedHTML"></div>
```



Rysunek 4.18. Na tej stronie umieszczono dwa pola. W pierwszym polu, utworzonym przez element `<iframe>`, widać omawianą w rozdziale 2. apokaliptyczną stronę. Drugie pole jest zwykłym elementem `<div>`, który wyświetla kod dokumentu HTML po tym, jak został zmodyfikowany. Dwa przyciski pośrodku strony służą do przełączania komponentu `<iframe>` w tryb projektowania i z powrotem w tryb normalny

Jak widać, podobnie jak na poprzednio omawianej stronie, przykład ten działa dzięki metodom `startEdit()` i `stopEdit()`. Zmodyfikowano jednak kod JavaScript po to, żeby dodawał atrybut `designMode` zamiast `contentEditable`:

```
function startEdit() {
    // Włącza tryb projektowania w elemencie <iframe>.
    var editor = document.getElementById("pageEditor");
    editor.contentWindow.document.designMode = "on";
}

function stopEdit() {
    // Wylacza tryb projektowania w elemencie <iframe>.
    var editor = document.getElementById("pageEditor");
    editor.contentWindow.document.designMode = "off";

    // Wyświetla zmodyfikowany kod HTML (po to tylko, aby pokazać, że wciąż tam jest).
    var htmlDisplay = document.getElementById("editedHTML");
    htmlDisplay.textContent =
    editor.contentWindow.document.body.innerHTML;
}
```

Ten przykład pozwoli Ci lepiej poznać skalę możliwości edytowania dokumentów HTML. Zauważ, jak po kliknięciu grafiki na stronie przeglądarka pozwala dowolnie nią manipulować. Możesz zmieniać jej rozmiar, przemieszczać po stronie lub usunąć ją jednym kliknięciem lub wciśnięciem klawisza *Delete*. W podobny sposób da się modyfikować kontrolki formularza, jeśli są obecne na edytowanej stronie.

Naturalnie, nim będziesz zdolny zmienić ten przykład w coś działającego bardziej praktycznie, musisz pokonać istotną lukę. Najpierw warto by było dodać lepsze kontrolki edytowania. Po raz kolejny na ratunek przychodzi zespół twórców Opery, którzy wesprą Cię w poznawaniu modelu kontroli wykraczającego poza zakres tego rozdziału (odwiedź strony <http://tinyurl.com/htmlEdit1> oraz <http://tinyurl.com/htmlEdit2>). Poza tym przydałoby się zrobić coś pożytecznego ze zmienionym przez Ciebie kodem, np. wysłać go na serwer za pośrednictwem obiektu XMLHttpRequest (więcej szczegółów na stronie 322).

Należy też wspomnieć o jednej ważnej rzeczy. Jeśli uruchomisz ten przykład z dysku twardego Twojego własnego komputera, może nie działać w niektórych przeglądarkach (Internet Explorer i Chrome natrafiają na ograniczenia bezpieczeństwa, podczas gdy Firefox śmiga bez większych problemów). Problemu tego możesz uniknąć, uruchamiając ten kod na utworzonej przez nas stronie www.prosetech.com/html5.

Skorowidz

.NET, 342

A

Access, 14
Adobe Dreamweaver, 371
Adobe Flash, 32, 154
Adobe Illustrator, 234
Adobe Photoshop, 200
adres
 IP, 345
 URL, 137
agregacja, 86
Ajax, 322
akceleracja sprzętowa, 230
Android, 230, 264
animacja, 79, 227, 279
aplikacja
 offline, 304
 sieciowa, 179
Apple, 15, 32, 159, 169, 255
Apple Quick Time, 154
argument, 403
ARIA, 97
arkusz stylów, 36, 60, 175, 180, 376
ASP.NET, 321, 325
asynchroniczna metoda, 297
asynchroniczne żądanie, 323
atrybut
 alt, 177
 autocapitalize, 134
 autocomplete, 134
 autocorrect, 134
 autofocus, 123, 125
 autoplay, 157
 class, 378
 formnovalidate, 128

height, 158
high, 146
id, 382
itemprop, 105
itemreviewed, 109
itemscope, 105
itemtype, 105
lang, 36
language, 37
loop, 157
low, 146
media, 259, 266
multiple, 134, 137, 299
novalidate, 127
onclick, 195
pattern, 130, 132
placeholder, 123
poster, 158
spellcheck, 134
step, 139
type, 36
width, 158
zdarzenia, 407

B

baner, 73
base64, 198
biblioteka
 CanvasExplorer, 202
 ExplorerCanvas, 203
 FlashCanvas, 203
 graficzna, 190
 html5Widgets, 141
 PathJS, 367
 VideoSub, 178
 webforms2, 133
blok treści, 81

Bluetooth, 347
Blu-ray, 159
bookmarklet, 82

C

cache, 304, 311
Cascading Style Sheets, 375
Chrome, 78, 82, 102, 145, 247, 307, 308, 312, 399
 Frame, 344
ciasteczka, 283
ciąg Fibonacciego, 365
cień, 210, 269
Cisco, 153
class, 102
cookies, 283, 289
CSS, 36, 57, 61, 124, 183, 254, 264, 376
 overflow, 358
 overflow-x, 358
 reguła, 378
 zapytania medialne, 385
CSS3, 241, 242, 243, 246
cytat, 68
czas zegarowy, 93
czat, 342

D

dane semantyczne, 110
debugowanie, 399
deseń, 212
doctype, 34
DOM, 31, 34, 405
domena witryny, 285
dostawca lokalizacji, 345

dostęp do danych, 285
Dreamweaver, 35, 40
DTP, 12
dziedziczenie własności, 380

E

efekt cienia, 269
eksploracja danych, 97
ekstrawersja, 218
element
 <div>, 36
 body, 85
 canvas, 179
 nadmiarowy, 39
elementy semantyczne, 58,
 59, 69, 70, 91, 92
email, 134, 135, 137
Embedded OpenType, 248
EOT, 248, 249, 250, 255
Excel, 11, 14
Expression Web, 35, 40

F

Facebook, 368
FALLBACK, 315
File API, 287, 295, 296, 302
Firefogg, 166
Firefox, 143, 145, 199, 246,
 250, 298, 307, 399
Flash, 32, 164, 167, 202
 Video Format, 167
Flickr, 368, 370
FloatFigure, 68
flv, 167
font
 sieciowy, 384
 zagnieżdżenie, 254
Font Squirrel, 250, 256
font-face, 251
fonty, 247
 licencja, 255
 sieciowe, 242
 zestawy, 250
formatowanie, 377
formularz, 118, 126, 404
fotografia cyfrowa, 14
FOUT, 250
funkcja
 addBall(), 230
 addRandomCircle(), 223
 alert(), 391
 boing(), 172
 canvasClick(), 225
 changeColor(), 195
 checkForCollision(), 238
 Circle(), 221, 222
 clearCanvas(), 197
 clearInterval(), 227
 draw(), 197
 drawCircles(), 223
 drawFrame(), 228,
 230, 231
 drawImage(), 207
 drawMaze(), 234
 drop(), 301
 fillRect(), 189
 flush(), 334
 gradient(), 272
 ignoreDrag(), 301
 linear-gradient(), 272
 lineTo(), 197
 Number(), 291, 292
 obsługująca, 395
 plotScore(), 218, 219
 processFiles(), 297,
 299, 301
 radial-gradient(), 273
 randomFromTo(), 223
 repeating-linear-
 gradient(), 273
 repeating-radial-
 gradient(), 273
 restore(), 191
 rgb(), 183, 386
 rgba(), 192, 193, 266
 rotate(), 276
 setInterval(), 227, 331
 setTimeout(), 227,
 235, 331
 setTimout(), 227
 startDrawing(), 196
 stopDrawing(), 197
 storageChanged(), 294
 stroke(), 183, 186, 197
 strokeRect(), 186
 time(), 334
 url(), 252
 validateComments(), 131
 wplatana, 408
 zwrotna, 347

G

Geolocator, 347
geolokalizacja, 343, 344,
 346, 348
GET, 326

GitHub, 55, 202, 289
globalCompositeOperation,
 193
GlobalStats, 52
Gmail, 368
Google, 107, 110, 253
 AdSense, 74
 Chrome, 18
 Frame, 55
 Finance, 331
 Fonts, 256
 Gears, 344, 354, 363
 Maps API, 352
 Mapy, 346, 352, 353,
 394
 search preview, 109
 Web Fonts, 253
GPS, 346, 347, 351
gra MMO, 338
gradient, 210, 212, 213,
 215, 245, 271
 liniowy, 271, 276
 promienisty, 216, 271
 wielokolorowy, 216
gruba stopka, 79

H

H.264, 159, 160, 161
 licencja, 160
hCalendar, 102, 103
hCard, 99, 102
hierarchia nagłówków, 74
historia sesji, 343, 368, 370
hMedia, 102
Hotmail, 331
HTML5, 78, 135, 164,
 177, 364
 audio i wideo, 155
 ograniczenia, 155
 formularz, 122
 nowe znaczniki, 44
 Outliner, 82
 semantyka, 59
 składnia, 43
 styl, 39
 system konspektów, 86
 walidacja, 39
 walidator, 35
 znaczniki
 semantyczne, 91
HTTP, 311

I
IE, 43, 52, 202, 289
Indexed DB, 302
instrukcje warunkowe, 133
Internet Explorer, 18, 125,
145, 171, 244, 257, 370,
391, 399
iPad, 32, 307
iPhone, 14, 32, 230,
264, 307
ISO, 111

J
jasność barwy, 266
Java, 321, 342
JavaScript, 390
funkcje, 402
operatory logiczne, 400
pętle, 401
plik, 393
tablice, 402
zdarzenia, 396
język użytkownika, 36
JPEG, 198
jQuery, 17, 279, 389, 407
JS-API, 49
JSON, 292, 332, 360
JW Player, 169

K
Kaazing, 342
kanał alfa, 192, 238, 266
Kinetic JS, 224
klasa
applicationCache, 319
FileReader, 297, 301
klient, 283
klucz, 285
kod semantyczny, 104
kodowanie
audio, 166
ISO 8859-2, 35
mediów, 166
plików wideo, 154
UTF-8, 35
wideo, 166
znaków, 35, 41
kolor
przekształcanie, 274
tła, 180
komentarze, 379
kompatybilność, 43

komponent
<body>, 39
<head>, 39
<video>, 266
doctype, 35
warstwy prezentacji, 44
konspekt
algorytm, 89
strony, 82
kontekst graficzny, 181
kontener, 94, 265
kontrolka, 95, 122
<input>, 298
tel, 138
korzenie sekcji, 87
krzywa Béziera, 187

L
labirynt, 233, 234
liczby pierwsze, 357
lineCap, 183
linia prosta, 182
localhost, 312
lokalizacja, *Patrz:*
geolokalizacja
lokalizowanie trafień, 221

Ł
łańcuch zapytań, 324

M
Mac OS, 17, 252, 253
magazyn
danych, 286
lokalny, 284
sesji, 284, 286
sieciowy, 283, 286
magenta, 216
manifest, 304, 305, 306,
307, 308, 310, 311, 319
mapa, 354
mechanizm awaryjnego
ładowania, 169
menu nawigacji, 77
metadane, 98
metoda
addColorStop(), 216
arc(), 186
arcTo(), 186
back(), 365
beginPath(), 184, 185

bezierCurveTo(), 186
canPlayType(), 172
clear(), 289
clearWatch(), 347, 355
click(), 296, 299
close(), 336, 337, 362
closePath(), 185, 188
createImageData(),
206, 237
createPattern(), 212
document, 404
document.getElementById(),
181
drawImage(), 206, 209
fill(), 185, 186, 221
fillRect(), 186, 188
fillStroke, 192
fillStyle, 192
fillText(), 209
getContext(), 181
getCurrentLocation(),
349, 350
getCurrentPosition(),
347, 348, 349, 355
getImageData(), 198, 237
key(), 289
lineTo(), 182
measureText(), 209
moveTo(), 182
open(), 326
pause(), 171
play(), 171
postMessage(), 359,
362, 363
pushState(), 365, 368
putImageData(), 237
quadraticCurveTo(), 186
readAsArrayBuffer(), 298
readAsBinaryString(),
298
readAsDataURL(), 298,
299, 301
readAsText(), 298
removeItem(), 289
save(), 191
send(), 326, 340
setCustomValidity(), 130
setInterval(), 356
setTimeout(), 228, 356
startEdit(), 150
stopEdit(), 150
stroke(), 182, 221
strokeText(), 210
swapCache(), 319
terminate(), 362

- toDataURL(), 198
- update(), 319
- watchPosition(), 347, 355
- window.addEventListener(), 294
- Microsoft, 159, 255
- Microsoft Expression Web, 371
- Microsoft Paint, 200
- mikrodane, 19, 91, 93, 104, 106, 107, 111
- mikroformat
 - hCalendar, 103
 - hCard, 100
- mikroformaty, 99, 107, 111
- MIME, 161, 162, 165, 304, 308, 334
- mobilne urzadzenia, 258
- mobilny ekran, 266
- Modernizr, 53, 244
- MooTools, 279
- Mozilla, 27, 159, 246
- MP3, 159, 160, 161
- MPEG-4, 160

N

- nagłówek, 92
 - rozbudowany, 73
- najechanie myszą, 278
- nakładanie własności, 379
- narzędzie do rysowania, 190
- negatyw, 238
- node.JS, 342
- null, 397
- numer telefonu, 138

O

- obiekt
 - applicationCache, 318
 - daty, 291
 - HTML, 406
 - localStorage, 286, 287
 - navigator, 347
 - pracownika, 356
 - sessionStorage, 286, 287
 - Worker, 359
 - XMLHttpRequest, 322, 326, 329, 338, 366
- obszar
 - nawigacji, 76
 - zwijany, 78
- odtworzacz
 - Flowplayer, 177

- jPlayer, 175
- LeanBack Player, 178
- VideoJS, 175, 176
- Office, 14
- offline, 304, 309, 313, 317
 - aplikacja, 310
- ogg, 160, 161
- okrąg, 187
- online, 314
- Oomph, 102
- OpenType PostScript, 248
- Opera, 82, 257, 399
- operatory arytmetyczne, 398
- opływanie, 66
- optymalizacja, 107
- OTF, 248, 249, 250, 252, 255

P

- padding, 387
- Paint, 141
- pamięć podręczna, 305, 310, 312, 314, 319
- panel boczny, 74, 76
- parametr, 403
 - contentEditable, 147
 - designMode, 149
 - max-device-width, 264
- pasek postępu, 145
- peer-to-peer, 338
- PersonalityScore, 315
- pętla, 401
 - for, 223
- Photoshop, 209, 238
- PHP, 321, 324, 332, 342
- piaskownica, 404
- plótno, 179, 205
- PNG, 198
- podpisy implementacja, 177
- polecenie echo, 334
- polling, 331, 336
- położenie, 346
- POST, 326
- PostScript, 248, 249
- PowerPoint, 14
- pozycjonowanie, 106
 - stałe, 79
 - stron, 107
- praca w tle, 343
- preferencje aplikacji, 288
- proces w tle, 359
- przeciągnij-i-upuść, 31, 296
- przedrostki autorskie, 246
- przejścia, 277

- przeźrzeń nazw, 42
- prześwitujące tło, 79
- przezroczystość, 192, 266
- przycisk, 122
- pseudoklasa
 - focus, 128, 273
 - hover, 273, 383
 - in-range, 128
 - invalid, 128
 - optional, 128
 - out-of-range, 128
 - required, 128
 - valid, 128
 - visited, 383
- pubdate, 94
- punkt kontrolny, 187
- Python, 321, 342

Q

- query string, 324

R

- radial-gradient, 246
- ramka HTML, 44
- RDFa, 98, 106, 107, 111
- reklama, 68
- rendering, 323
- RFID, 347
- Rich Snippets Testing Tool, 107
- robot indeksujący, 65, 94
- rola prezentacyjna, 96
- RSS, 103
- Ruby, 321, 342
- rysowanie, 180

S

- Safari, 18, 138, 145, 257, 302, 307, 399
- sandbox, 404
- sans-serif, 385
- sekcja <head>, 376
- selektor kontekstowy, 62, 381
- semantyczne ujęcie danych, 112
- semantyka, 91
- SEO, 107
- serif, 385
- serwer, 311
 - sieciowy, 283

sieciowe fonty, 247
Silverlight, 154, 202, 224
skrypt
 html5.js, 71
 html5Widgets, 134
stan aplikacji, 288
standard osadzania, 98
stopka, 92
SVG, 28, 248, 249, 250,
 252, 255
syndykacja, 86
szablony stron, 74
szachy, 239

Ś

średnik, 390

T

tablica, 401
technika poślizgowych
 drzwi, 269
telefony komórkowe, 266
TextEdit, 17
Theora, 160, 161, 165,
 166, 167
this, 222
tło, 269
 półprzezroczyste, 266
 zespólone, 269
transformata, 188, 277
 macierzy, 190
 obrotów, 190
 skali, 190
 tłumaczeniowa, 190
 trójwymiarowa, 279
transitions, 330
TrueType, 248, 253
tryb
 dziwactw, 34
 offline, 303, 304, 306
 standardów, 35
TTF, 248, 249, 250, 252,
 255
Twitter, 366
typ
 color, 141
 date, 141
 datetime, 141
 datetimelocal, 141
 MIME, 43
 month, 141
 number, 138
 range, 139
 time, 141

week, 141
typy danych, 398

U

układ dwukolumnowy, 261
ukośnik zamykający, 38
Unicode, 36
Untitled, 85
URL, 199
 przedrostek, 137
UTF-8, 36

V

VML, 201
Vorbis, 160, 161

W

W3C, 15, 19, 28, 91
WAI, 59
walidacja, 125, 132, 135, 404
 liczba, 138
 po stronie klienta,
 125, 126
 po stronie serwera, 126
 wprowadzonego kodu, 40
walidator, 40
 W3C, 40
warunek, 400
WAV, 161
web storage, 284
web worker, 343, 356
webfonts, 256
WebM, 160, 165, 166
WebSocket, 246, 321, 322,
 337, 338, 339, 340
 przykład, 341
WebSRT, 178
WHATWG, 28, 29, 97, 343
window.onload, 183
Windows 7, 52
Windows Media Player, 154
Windows Vista, 52
Windows XP, 52, 200, 250
wizytówka, 99
własność
 @font-face, 248
 accuracy, 349
 background-color, 386
 background-position, 269
 background-repeat, 269
 border-radius, 268
 border-thickness, 244
 border-top-left-radius,
 268
 box-shadow, 269, 276
 column-count, 257
 column-gap, 257
 column-rule, 257
 corner-radius, 247
 currentTime, 175
 enableHighAccuracy, 350
 fillStyle, 185, 212
 fillStyle(), 186
 globalAlpha, 193
 isSelected, 222
 lineWidth, 183, 186, 210
 max-device-width, 262
 opacity, 267, 276
 playbackRate, 174
 radial-gradient, 247
 strokeStyle, 183, 186,
 210, 212
 text-shadow, 269
 transition, 274
własny odtwarzacz, 173
właściwość
 border-radius, 243
 coords, 349
 focus, 124
 font-family, 384
 innerHTML, 405
 max-device-width, 265
 maximumAge, 352
 messageType, 364
 orientation, 265
 readyState, 326
 rgb(), 192
 status, 326
 timeout, 351
 transform, 242
 transparency, 242
WOFF, 248, 249, 250,
 252, 255
Word, 14
współczynnik korekcji, 265
wtyczka
 h5o, 82, 85
wykres, 216
wypełnianie, 55
wyrażenia regularne,
 129, 131
wyszukiwarka, 111
 Google, 107, 110
 internetowa, 36
 przepisów, 111
wzorce tekstu, 129

X

XForms, 117
XHTML, 25, 41, 117, 167
XHTML5, 41
 walidator, 42
XML, 105, 298
 dokument, 43
XMLHttpRequest, 119, 151
XPath, 43
XQuery, 43

Y

YouTube, 32, 153, 167

Z

zadania asynchroniczne, 408
zagnieżdżanie kodu, 390
zapętlone nagranie, 157
zapytania medialne, 259, 265
zdarzenie
 findAllItems(), 290
 onBeforeUnload, 287
 onChange, 297, 299
 onClick, 408
 onClose, 340
 onDragEnter, 301
 onDragOver, 301
 onError, 340, 361
 onInput, 130
 onLoad, 297
 onLoadEnd, 302
 onMessage, 340, 359
 onMouseDown, 197
 onMouseOut, 197
 onMouseOver, 394

onMouseUp, 197
onOpen, 340
onProgress, 302
onStorage, 289
onSubmit, 132
onTimeUpdate, 175
onUpdateReady, 318
window.onStorage, 293
zmienna, 395
 globalna, 397
 lokalna, 397
znacznik
 <p>, 33
 </script>, 37
 <a>, 47
 <address>, 46
 <article>, 64, 85, 261
 <aside>, 76
 <aside>, 68, 75, 89,
 261
 <audio>, 153, 156,
 170, 171
 , 46
 <body>, 33, 335, 384,
 391
 <canvas>, 180, 201,
 203, 206, 239
 <cite>, 47
 <command>, 146
 <datalist>, 142
 <details>, 78
 <div>, 63, 77, 97, 301,
 352, 380
 , 46
 <embed>, 47, 154
 <fieldset>, 120
 <figcaption>, 68
 <figure>, 68
 <footer>, 58, 80
 <form>, 95, 119
 <h1>, 85
 <header>, 65, 72, 74
 <hgroup>, 65, 72
 <hr>, 45
 <i>, 46
 <iframe>, 44, 149
 , 38, 195, 212,
 235
 <input>, 121, 126,
 135, 357
 , 76
 <link>, 36, 376
 <mark>, 95
 <menu>, 146
 <meter>, 144
 <nav>, 59, 75
 <nobr>, 48
 <object>, 169
 <output>, 94
 <progress>, 144
 <s>, 45
 <script>, 390, 392
 <section>, 81
 <select>, 121, 134
 <small>, 45
 <source>, 164, 169
 , 95, 102, 380
 , 46
 <summary>, 78
 <textarea>, 126
 <time>, 58, 93
 <title>, 33
 , 76
 <video>, 153, 158,
 165, 166, 169
 <wbr>, 47
 meta, 35
znak sieci, 37, 391
znak wodny, 123
znaki diakrytyczne, 253

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

HTML5

nieoficjalny podręcznik

HTML5 to coś więcej niż język służący do tworzenia stron WWW – to zbiór kilkunastu niezależnych standardów sieciowych pod jednym wspólnym szyldem. Z rozmachem wkracza do codziennego życia projektantów stron internetowych. Jego nowe możliwości naprawdę robią wrażenie: obejmują ścisłą integrację ze środowiskiem przeglądarki internetowej, usługi geolokalizacyjne, doskonałe wsparcie dla multimediiów i aplikacji offline. Jeszcze niedawno takie możliwości nie śniły się żadnym webmasterom, a dziś są w zasięgu każdego!

Dzięki kolejnej książce z serii *Nieoficjalny podręcznik* nie musisz odkrywać tajników HTML5 na własną rękę. Znajdziesz tu wszystkie istotne informacje, dzięki którym błyskawicznie zaczniesz korzystać z dobrodziejstw HTML5. W trakcie lektury nauczysz się dynamicznie rysować elementy, używać geolokalizacji oraz przechowywać dane użytkowników w lokalnych magazynach danych. Ponadto poznasz nowe znaczniki oraz ich przeznaczenie. HTML5 to przyszłość sieci, dlatego już dziś warto poznać jego możliwości!

HTML5 to:

- wsparcie dla plików multimedialnych
- usługi geolokalizacyjne
- wygodne przechowywanie danych
- aplikacje offline
- standard nowoczesnej sieci Internet!

Sprawdź, jak będzie wyglądać sieć jutra,
i zacznij korzystać z tego już dziś!

helion.pl
księgarnia
internetowa

Nr katalogowy: 8629



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/nowosci>

Helion SA

ul. Fosduchki 1c, 44-100 Gliwice

tel.: 32 200 98 83

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-246-3948-9



9 788324 639489

Cena 69,00 zł

Informatyka w najlepszym wydaniu