

Przewodnik po tajnikach HTML5!

HTML5

nieoficjalny podręcznik

Wydanie II



Matthew MacDonald

O'REILLY

Helion 

Tytuł oryginału: HTML5 The Missing Manual, 2nd Edition

Tłumaczenie: Maksymilian Gutowski
na podstawie: „HTML5. Nieoficjalny podręcznik” w tłumaczeniu Macieja Reszotnika

ISBN: 978-83-246-9251-4

© 2014 Helion S.A.

Authorized Polish translation of the English edition of HTML5: The Missing Manual, 2nd Edition ISBN 9781449363260 © 2014 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/htm5n2>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/htm5n2.zip>

Printed in Poland.

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

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

Spis treści

Nieoficjalna czołówka	11
Wstęp	15
Cześć I. Nowe oblicze języka HTML	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	29
1. Nie psuj sieci	30
2. Brukuj ścieżki	30
3. Bądź praktyczny	31
Rzut oka na składnię HTML5	32
Element doctype a HTML5	34
Kodowanie znaków	35
Język	35
Dodawanie arkusza stylów	36
Dołączanie JavaScriptu	36
Ostateczny produkt	37
Składnia HTML5 z bliska	38
Rozluźnione reguły	38
Walidacja HTML5	39
Powrót XHTML-u	42
Rodzina znaczników HTML5	43
Dodane elementy	43
Komponenty usunięte ze specyfikacji	43
Elementy zaadaptowane	45

Zmodyfikowane znaczniki	46
Elementy standaryzowane	47
Korzystanie z HTML5 już dziś	48
Ocenianie wsparcia ze strony przeglądarek	50
Statystyki poziomu przyjęcia przeglądarek	51
Wykrywanie obsługi własności z aplikacją Modernizr	53
Uzupełnianie braków przy użyciu wypełnienia	55
Rozdział 2. Zastosowanie elementów semantycznych na stronie	59
Wstęp do elementów semantycznych	60
Modernizacja tradycyjnej strony HTML	61
Struktura strony w stylu klasycznym	62
Struktura strony w HTML5	65
Dołączanie rysunków przy użyciu znacznika <figure>	67
Dodawanie ramki redaktorskiej — znacznik <aside>	70
Elementy semantyczne a kompatybilność z przeglądarkami	71
Stylizacja elementów semantycznych	72
Zastosowanie HTML5 Shiv	72
Modernizr: rozwiązanie uniwersalne	73
Projektowanie strony z nowymi elementami semantycznymi	73
Więcej o nagłówkach	74
Odnosiniki i element <nav>	76
Więcej o sekcjach	79
Więcej o stopce	81
Wskazywanie treści głównej przy użyciu elementu <main>	83
System tworzenia konspektu strony w HTML5	85
Jak zobaczyć konspekt?	85
Konspekt podstawowy	86
Komponenty sekcji	87
Problemy z tworzeniem konspektów	89
Rozdział 3. Jeszcze bardziej wyrazisty kod	93
Elementy semantyczne raz jeszcze	94
Data, czas i znacznik <time>	95
Obliczenia w JavaScriptcie i element <output>	96
Element <mark> i zaznaczanie tekstu	98
Inne standardy kodu semantycznego	99
ARIA (ang. Accessible Rich Internet Applications)	100
RDFa (ang. Resource Description Framework)	100
Mikroformaty	101
Mikrodane	102
Praktyczny przykład: modernizacja strony O mnie	105
Ekstrakcja danych semantycznych w przeglądarce	109
Wykorzystanie metadanych przez wyszukiwarki	110
Fragmenty sformatowane przez Google	110
Lepsze wyniki wyszukiwania	111
Wyszukiwarka przepisów	114

Rozdział 4. Udoskonalone formularze	119
Formularze	120
Modernizowanie tradycyjnego formularza HTML	121
Znak wodny — dodawanie wskazówek	124
Dobry punkt zaczepienia: właściwość focus	126
Walidacja: wykrywanie błędów	127
Proces walidacji w HTML5 krok po kroku	127
Wyłączanie mechanizmu walidacji	129
Formatowanie kontrolek walidacyjnych	130
Walidacja wyrażeń regularnych	131
Własne reguły walidacji	132
Obsługa mechanizmu walidacji	133
Testowanie obsługi za pomocą skryptu Modernizr	134
Uzupełnianie kodu przy użyciu biblioteki HTML5Forms	135
Nowe typy znacznika input	137
Adresy e-mail	139
Adresy URL	140
Pola wyszukiwania	140
Telefon	140
Liczby	140
Suwak	141
Czas: daty i godziny	142
Kolor	144
Nowe elementy	144
Sugerowane odpowiedzi i element <datalist>	144
Pasek stanu i miernik	146
Paski narzędzi i menu — znaczniki <command> i <menu>	149
Edytor HTML na stronie	149
Edytowanie zawartości za pomocą contentEditable	150
Edytowanie strony za pomocą atrybutu designMode	152

Cześć II. Filmy, obrazki i inne cudeńka..... 155

Rozdział 5. Multimedia	157
Wideo dziś	157
Wprowadzenie do audio i wideo w HTML5	159
Wydobywanie dźwięku z elementu <audio>	159
Wczytywanie multimediów po załadowaniu strony	160
Automatyczne odtwarzanie	161
Zapętlone odtwarzanie	161
Znacznik <video> z szerszej perspektywy	161
Wprowadzenie do formatów multimediów w HTML5	163
Więcej o formatach	164
Obsługa multimediów w przeglądarkach	165
Wiele formatów, czyli jak udobruchać każdą przeglądarkę	168
Obsługa różnych formatów	169
Alternatywa — wtyczka Flasha	170

Sterowanie odtwarzaniem za pomocą JavaScriptu	174
Dodawanie efektów dźwiękowych	174
Budowa własnego odtwarzacza filmów	177
Odtwarzacze JavaScript	180
Napisy i dostępność	181
Ścieżki napisów i WebVTT	181
Dodawanie podpisów elementem <track>	184
Obsługa podpisów w przeglądarkach	185
Rozdział 6. CSS3 a wygląd strony	187
Używanie CSS3 już dziś	188
Strategia 1.: Wykorzystaj to, co możesz	188
Strategia 2.: Traktuj własności CSS3 jak usprawnienia	189
Strategia 3.: Dodanie awaryjnych mechanizmów za pomocą Modernizra	190
Style właściwe dla konkretnych przeglądarek	192
Kontenery na błysk	194
Przezroczystość	195
Zaokrąglane rogi	196
Tło	197
Cienie	199
Gradyenty	200
Efekty przejścia	204
Przekształcanie koloru	204
Przejścia — teczka z pomysłami	206
Wywoływanie przejść w JavaScriptcie	207
Transformaty	209
Przejścia wykorzystujące transformaty	212
Typografia w sieci	213
Formaty fontów	214
Font dla witryny	216
Pobieranie darmowych fontów z Font Squirrel	217
Przygotowanie fontu sieciowego	218
Korzystanie z fontów sieciowych Google	221
Wielokolumnowy tekst	223
Rozdział 7. Projektowanie elastycznych witryn w CSS3	227
Podstawy projektowania elastycznego	228
Płynny layout	228
Płynne obrazy	231
Płynna typografia	233
Widok strony: obsługa layoutu na smartfonach	236
Dostosowywanie layoutu przy użyciu zapytań medialnych	237
Zapytania medialne	238
Utworzenie prostego zapytania medialnego	240
Layout przyjazny dla urządzeń mobilnych	241
Zapytania medialne — wyższa szkoła jazdy	245
Zastępowanie całego arkusza stylów	246
Rozpoznawanie urządzeń mobilnych	246

Rozdział 8. Podstawy rysowania na elemencie canvas	249
Płótno — wprowadzenie	250
Linie proste	252
Ścieżki i figury	254
Krzywe	256
Transformaty	258
Przezroczystość	262
Kompozycje złożone	263
Tworzenie prostego programu graficznego	264
Przygotowanie narzędzi	265
Malowanie po płótnie	267
Zachowywanie płótna	268
Płótno i kompatybilność z przeglądarkami	271
Wypełnienie ExplorerCanvas	271
Wypełnienie FlashCanvas	272
Alternatywne płótna i wykrywanie obsługi	273
Rozdział 9. Więcej o płótnie	275
Inne własności płótna	275
Rysowanie obrazów	276
Wycinanie i zmienianie wielkości obrazu	277
Rysowanie tekstu	278
Cienie i inne ozdobniki	280
Dodawanie cieni	280
Wypełnianie figur deseniem	282
Wypełnianie figur gradientem	283
Składanie wszystkiego w całość: rysowanie wykresów	286
Interaktywne figury	291
Śledzenie rysowanych elementów	291
Współrzędne i lokalizowanie trafień	294
Animowanie płótna	297
Podstawowa animacja	297
Animowanie wielu obiektów	299
Praktyczny przykład: labirynt	303
Rysowanie labiryntu	304
Animowanie ikony	305
Lokalizowanie trafień a barwa pikseli	307
Cześć III. Konstruowanie aplikacji sieciowych przy użyciu komponentów desktopowych	311
Rozdział 10. Magazyn danych	313
Magazyn sieciowy — podstawy	314
Magazynowanie danych	315
Praktyczny przykład: zapisywanie stanu gry	317
Magazyn sieciowy a obsługa przeglądarek	319



Magazyn sieciowy — na głębszych wodach	319
Usuwanie wpisów	319
Listowanie wszystkich zachowanych wpisów	320
Zapisywanie liczb i dat	321
Zachowywanie obiektów	322
Reagowanie na zmiany w magazynie	323
Odczytywanie plików	325
Pobieranie pliku	326
Odczytywanie pliku tekstowego	326
Zastępowanie standardowej kontrolki ładowania plików	328
Odczytywanie wielu plików jednocześnie	329
Odczytywanie pliku graficznego	329
File API i obsługa przeglądarek	332
IndexedDB: silnik bazy danych w przeglądarce	333
Obiekt przechowujący dane	335
Tworzenie bazy danych i łączenie z nią	336
Tworzenie zapisów w bazie danych	338
Przeglądanie wszystkich zapisów tablicy	340
Przeszukiwanie pojedynczego zapisu tablicy	342
Usunięcie zapisu	343
Obsługa IndexedDB w przeglądarkach	344
Rozdział 11. Aplikacje sieciowe z trybem offline	345
Cache'owanie plików	346
Tworzenie manifestu	347
Korzystanie z manifestu	349
Przenoszenie manifestu na serwer	349
Uaktualnianie manifestu	352
Obsługa w przeglądarkach aplikacji w trybie offline	355
Praktyczne techniki cache'owania	356
Uzyskiwanie dostępu do cache'owanych plików	356
Tryb awaryjny	357
Sprawdzanie stanu połączenia	359
Wykrywanie uaktualniania przy użyciu JavaScriptu	360
Rozdział 12. Komunikacja z serwerem sieciowym	365
Wysyłanie wiadomości na serwer	366
Obiekt XMLHttpRequest	366
Wysyłanie zapytań na serwer	367
Pobieranie nowych treści	371
Zdarzenia przesyłane na serwer	375
Format wiadomości	376
Wysyłanie wiadomości za pomocą skryptu serwera	377
Przetwarzanie wiadomości na stronie	379
Polling a zdarzenia po stronie serwera	380
Technologia WebSocket	382
Serwer WebSocketów	383
Prosty klient w technologii WebSocket	384
Przykłady technologii WebSocket w sieci	385

Rozdział 13. Geolokalizacja, obiekt pracownika i zarządzanie historią	389
Geolokalizacja	390
Jak działa geolokalizacja?	390
Odnajdywanie współrzędnych użytkownika	392
Usuwanie błędów	394
Ustawienia geolokalizacji	396
Generowanie mapy	397
Monitorowanie ruchu użytkownika	399
Obsługa geolokalizacji w przeglądarkach	400
Obiekt pracownika	401
Czasochłonne zadanie	401
Wykonywanie zadań w tle	404
Obsługa błędów pracownika	407
Anulowanie zadania uruchomionego w tle	407
Przekazywanie bardziej złożonych wiadomości	407
Obsługa pracowników w przeglądarkach	410
Zarządzanie historią	411
Kwestia URL	411
Dawne rozwiązanie: znak kratki i adres URL	412
Rozwiązanie HTML5: historia sesji	413
Historia sesji i kompatybilność	417
Dodatki	419
Dodatek A. Podstawy CSS	421
Załączanie stylów do stron	421
Anatomia arkusza stylów	422
Własności CSS	423
Formatowanie elementów przy użyciu klas	423
Komentarze w arkuszach stylów	425
Odrobinę bardziej zaawansowane arkusze stylów	425
Konstruowanie struktury strony przy użyciu elementu <div>	425
Wiele selektorów	426
Selektory kontekstowe	427
Selektor identyfikatora	428
Selektory pseudoklas	428
Selektory atrybutów	429
Wycieczka po stylach	430
Dodatek B. JavaScript — mózg nowoczesnej witryny	435
W jaki sposób witryny korzystają z JavaScriptu?	436
Zagnieżdżanie kodu w dokumencie HTML	436
Używanie funkcji	437
Przenoszenie kodu JavaScript do oddzielnego pliku	439
Odpowiadanie na zdarzenia	440



Podstawy składni języka	441
Zmienne	441
Wartość null	443
Zakres zmiennych	443
Typy danych	444
Operacje	444
Instrukcje warunkowe	446
Pętle	447
Tablice	448
Funkcje — otrzymywanie i zwracanie danych	449
Obiekty	450
Literały obiektu	451
Interakcja ze stroną	452
Manipulowanie elementem	453
Dynamiczne łączenie ze zdarzeniem	454
Zdarzenia wplątane	457
Skorowidz	459

Projektowanie elastycznych witryn w CSS3

Webdesignerzy musieli stawić czoło poważnemu wyzwaniu już wtedy, kiedy tylko zaczęli umieszczać materiały na stronach internetowych. Projektanci materiałów drukowanych mogli polegać na pewnych założeniach dotyczących wyglądu swoich dokumentów na papierze i sposobu, w jaki odbiorcy się nimi posługiwali, jednak w sieci panowało pod tym względem zupełne bezprawie. Ta sama strona HTML, w zależności od przeglądarek i preferencji użytkowników, mogła zostać wtłoczona do małego okienka albo mogła wisieć pośrodku ogromnego okna, zatem tworzenie rozbudowanych layoutów było ryzykowne. Layout, który wyglądał idealnie w jednym oknie, mógł się całkowicie rozłożyć w oknie o innych wymiarach.

Dziś to zróżnicowanie jest jeszcze większe. Webdeweloperzy muszą brać pod uwagę nie tylko różne wymiary przeglądarek na komputerach, ale także rozmiary różnych *urządzeń*, takich jak tablety i smartfony. Layouty jednocześnie zwiększyły swoją złożoność — dziś na większości witryn stykamy się z różnorodnymi menu, pomocami nawigacyjnymi, paskami bocznymi i podobnymi elementami. Jeśli Twoim celem jest utworzenie jednej witryny, która ma wyglądać równie estetycznie w różnych warunkach, poradzenie sobie z takimi szczegółami będzie nie lada wyzwaniem.

Ponieważ webdesignerzy już dawno przenieśli kod odpowiedzialny za układ strony i formatowanie do plików CSS, można oczekiwać, że standard ten zapewni jakieś rozwiązanie omówionego powyżej problemu. Rzeczywiście, CSS3 obsługuje **zapytania medialne**, które pozwalają stronie na przełączanie stylów w zależności od wielkości okna lub urządzenia, na którym jest wyświetlana.

Zapytania medialne są praktycznie nieodzowne w projektowaniu stron na urządzenia mobilne. Jeśli nawet nie przypuszczasz, że ktokolwiek będzie odwiedzać Twoje strony na smartfonie, zapytania dadzą pewność, że layout będzie się dostosowywał do środowiska pracy użytkownika, np. usuwając dodatkowe kolumny, gdy zabraknie na nie miejsca, albo przenosząc linki nawigacyjne z góry strony

na bok. Taka elastyczność wiąże się z popularną filozofią webdesignu, noszącą nazwę projektowania elastycznego (ang. *responsive design*), którą omówimy w tym rozdziale.

Podstawy projektowania elastycznego

Od początku istnienia sieci zróżnicowanie wymiarów okien było przyczyną wielu kłopotów. Z biegiem lat webdesignerzy opracowali szereg mniej lub bardziej eleganckich, komplementarnych technik, pozwalających na sprostanie wymogom projektowania elastycznego.

Przed zapoznaniem się z zapytaniami medialnymi warto przyrzeć się ukazanym tutaj tradycyjnym metodom. Wszystkie są nadal ważne, ale — jak się przekonasz — nie stanowią pełnego rozwiązania. Znając ich ograniczenia, będziesz mógł zrozumieć, w jaki sposób CSS3 wypełnia wszystkie luki.

Płynny layout

Najprostszym sposobem zabezpieczenia strony przy zmianach wymiarów okna jest utworzenie layoutu **proporcjonalnego**, czyli takiego, który zajmuje całą dostępną powierzchnię, niezależnie od jej wielkości. Opracowanie proporcjonalnego layoutu jest — w teorii — całkiem proste. Podstawową zasadą jest podzielenie strony na kolumny o szerokościach określanych wartościami procentowymi, a nie pikselowymi. Powiedzmy, że Twoja strona ma następujący, dwukolumnowy układ:

```
<body>
  <div class="leftColumn">
    ...
  </div>

  <div class="rightColumn">
    ...
  </div>
</body>
```

Reguły stylów layoutu o stałych wymiarach mogą wyglądać następująco:

```
.leftColumn {
  width: 275px;
  float: left;
}

.rightColumn {
  width: 685px;
  float: left;
}

body {
  margin: 0px;
}
```

Tymczasem reguły stylów w zdefiniowanym proporcjonalnie układzie strony wyglądają tak:

```
.leftColumn {
  width: 28.6%;
  float: left;
}

.rightColumn {
  width: 71.4%;
  float: left;
}

body {
  margin: 0px;
}
```

Lewa kolumna ma tutaj szerokość 28,6%, wobec czego zajmuje 28,6% szerokości swojego kontenera, elementu <body>. W tym przykładzie element <body> pozabawiony jest marginesów, a zatem rozciąga się na całą szerokość okna przeglądarki, przy czym lewa kolumna obejmuje 28,6% tego obszaru.

Jak można się domyślić, wartości procentowe obydwu kolumn wynoszą łącznie równo 100%. Kolumny rozciągają się i kurczą, aby dopasować się do szerokości okna przeglądarki. Layouty proporcjonalne nazywane są **plynnymi**, ponieważ zamieszczona w nich treść gładko dostosowuje się do dostępnej przestrzeni.

Uwaga: W tym przykładzie szerokość lewej kolumny (28,6%) wyliczono, dzieląc stałą szerokość kolumny (275 pikseli) przez szerokość całego układu (wynoszącą 960 pikseli, co jest częstym standardem). Ponieważ projektowanie większości układów zaczyna się zwykle na ustalonych szerokościach, webdeveloperzy są przyzwyczajeni do wykonywania takich obliczeń przy tworzeniu płynnych layoutów.

Rzecz jasna, dostosowanie szerokości samych kolumn nie wystarczy. Musisz również wziąć pod uwagę marginesy, odstępy i obramowania. Początkujący webdeveloperzy często wprowadzają w swoich pierwszych płynnych layoutach ustalone marginesy i odstępy (o wartościach pikselowych), a jednocześnie proporcjonalnie określają wymiary kolumn. Wskutek tego kolumny zajmują jedynie przestrzeń pozostałą po odjęciu szerokości marginesów od całej szerokości strony. Procentowe wartości szerokości kolumn są jednak wyliczane na podstawie całej szerokości strony, bez uwzględnienia marginesów. Ta rozbieżność może powodować problemy z wyświetlaniem strony w wąskich oknach, bo marginesy o wartościach pikselowych zabierają miejsce proporcjonalnie zdefiniowanym kolumnom.

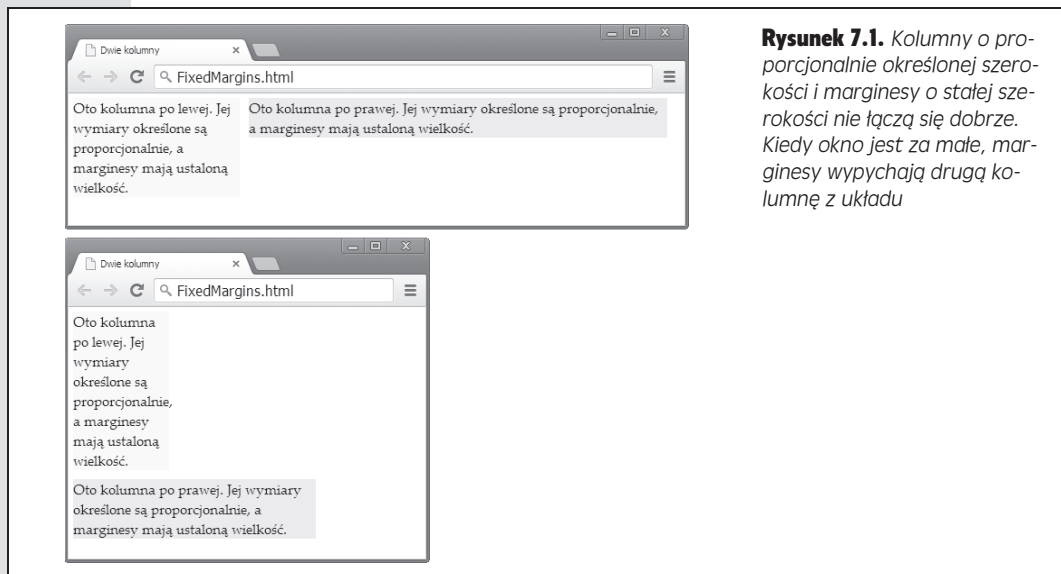
Zastanówmy się nad poniższymi stylami:

```
.leftColumn {
  width: 27%;
  margin: 5px;
  float: left;
}

.rightColumn {
  width: 68%;
  margin: 5px;
  float: left;
}
```

Te dwie kolumny zajmują łącznie 95% szerokości, pozostawiając dodatkowe 5% na marginesy. To wystarczy dla okien o średnich i dużych rozmiarach, ale w niewielkim oknie pozostałe 5% nie pomieści marginesów o ustalonej szerokości.

Aby zilustrować ten problem, nadaj obydwu kolumnom dwa różne kolory za pomocą własności `background`, a następnie pozmieniaj wymiary okna, tak jak na rysunku 7.1.



Rysunek 7.1. Kolumny o proporcjonalnie określonej szerokości i marginesy o stałej szerokości nie łączą się dobrze. Kiedy okno jest za małe, marginesy wypychają drugą kolumnę z układu

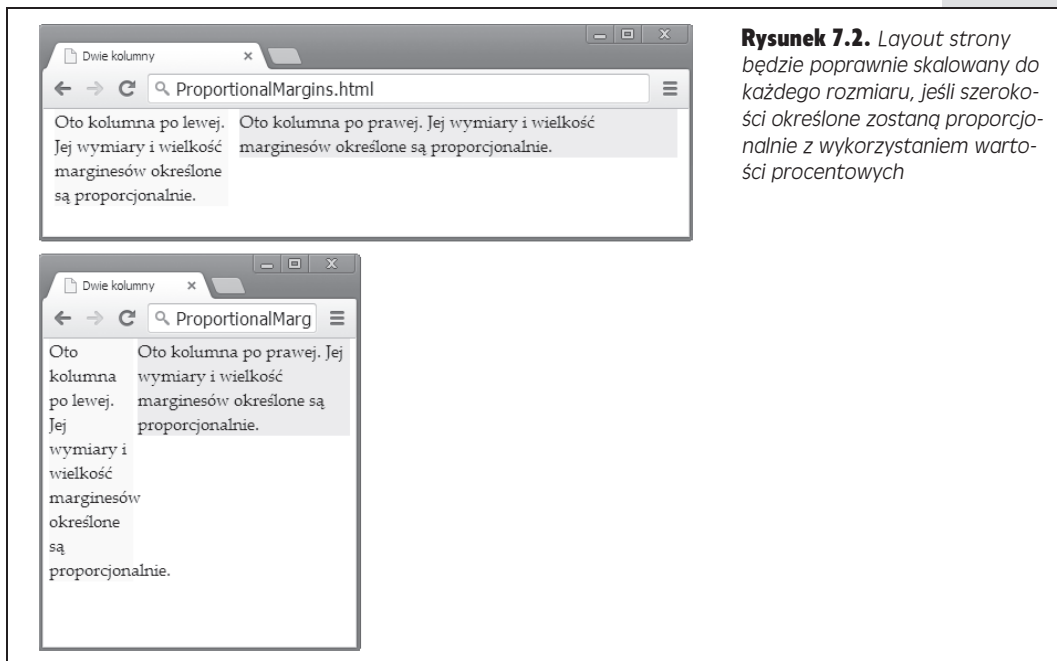
Rozwiązaniem tego problemu jest nadanie proporcjonalnej szerokości wszystkim marginesom, znajdującym się pomiędzy proporcjonalnymi kolumnami. Jeśli zatem kolumny pozostawiają 5% wolnego miejsca, możesz je przypisać marginesom. Podziel tę szerokość na trzy obszary o szerokości 1,66% — jeden na lewym krańcu okna, drugi na prawym i trzeci pomiędzy kolumnami.

```
.leftColumn {
    width: 27%;
    margin-left: 1.66%;
    margin-right: 1.66%;
    background: #FFFFCC;
    float: left;
}

.rightColumn {
    width: 68%;
    margin-right: 1.66%;
    background: #CCFFCC;
    float: left;
}
```

Na rysunku 7.2 widać efekt określenia szerokości zarówno kolumn, jak i marginesów proporcjonalnie przy użyciu wartości procentowych.

Proporcjonalne marginesy mogą Ci nie odpowiadać, jeśli potrzebujesz innego efektu. Jeżeli nie chcesz, by marginesy zmieniały wielkość stosownie do rozmiaru okna przeglądarki, możesz spróbować obejść to zachowanie. Możesz np. zamieścić dodatkowy element w jednej z proporcjonalnych kolumn i nadać mu własne marginesy lub odstępy o ustalonej szerokości. Ponieważ element ten umieszczony jest w obrębie zdefiniowanego już, w pełni proporcjonalnego układu, będzie się stale znajdował na swoim miejscu, niezależnie od wielkości okna.



Rysunek 7.2. Layout strony będzie poprawnie skalowany do każdego rozmiaru, jeśli szerokości określone zostaną proporcjonalnie z wykorzystaniem wartości procentowych

Obramowania sprawiają podobne problemy. Obramowanie nadane kolumnom zajmuje pewną ilość miejsca i może rozłożyć układ tak samo jak marginesy o ustalonej szerokości z rysunku 7.1. W takiej sytuacji problemu nie można rozwiązać poprzez wyliczenie proporcji, gdyż szerokości obramowania nie można określać w procentach. Najprostszym rozwiązaniem jest wykorzystanie techniki wspomnianej powyżej: osadź element `<div>` w kolumnie o procentowo określonej szerokości i nadaj mu obramowanie. Ta sprawdzona metoda wprowadzi nieco zaśmiewa kod (zmuszając do utworzenia dodatkowej warstwy layoutu), ale dzięki temu układ strony może działać sprawnie w oknie o dowolnym rozmiarze.

Płynne obrazy

Wielokolumnowy, proporcjonalny układ strony jest podstawą elastycznego projektu. Kiedy jednak przychodzi czas na zajęcie się samą zawartością kolumn na stronie, pojawia się wiele innych kwestii, które trzeba wziąć pod uwagę.

Jedną z nich są obrazy. Pola obrazów standardowo dopasowują się do swojej wartości, czyli dokładnie, co do piksela, przyjmują wymiary obrazu. Takie ustawienie może jednak wywoływać problemy w mniejszych oknach. Jeśli brakuje miejsca na wyświetlenie całego obrazu, wylewa się on z kolumny i rozchodzi na inne elementy, zakrywając je i wywołując niechłujne wrażenie.

Rozwiązanie tego problemu jest proste. Wystarczy nadać każdemu obrazowi maksymalną szerokość równą szerokości jego kontenera:

```
img {
  max-width: 100%;
}
```

WIZJE PRZYSZŁOŚCI

Własność `box-sizing` i funkcja `calc()`

Omówione w tym punkcie problemy z layoutami są do tego stopnia powszechne, że w CSS3 można znaleźć mnóstwo potencjalnych rozwiązań. Oto dwa z najbardziej obiecujących, choć nie bez wad.

- ◆ **Box-sizing.** Obramowania standardowo zamieszczone są po zewnętrznej stronie elementów, przez co trzeba brać pod uwagę ich obszar w obliczeniach proporcji layoutu. W CSS3 pojawiła się jednak właściwość `box-sizing`, której wystarczy nadać wartość `border-box`, aby obramowanie znalazło się w *obrębie* pola elementu. Obramowanie wygląda wtedy tak samo, ale obliczenia wielkości takich komponentów ulegają zmianie. Oznacza to tyle, że kolumna szeroka na 67% zajmuje 67% szerokości kontenera, niezależnie od szerokości swojego obramowania.
- ◆ **Funkcja `calc()`.** Jeśli musisz zestawić wymiary proporcjonalne z ustalonymi, przeprowadzenie stosownych obliczeń możesz powierzyć funkcji `calc()` w CSS3, a następnie wykorzystać ich wyniki w layoutcie. Wyobraź sobie, że musisz utworzyć kolumnę szeroką na 67% kontenera bez 5 pikseli,

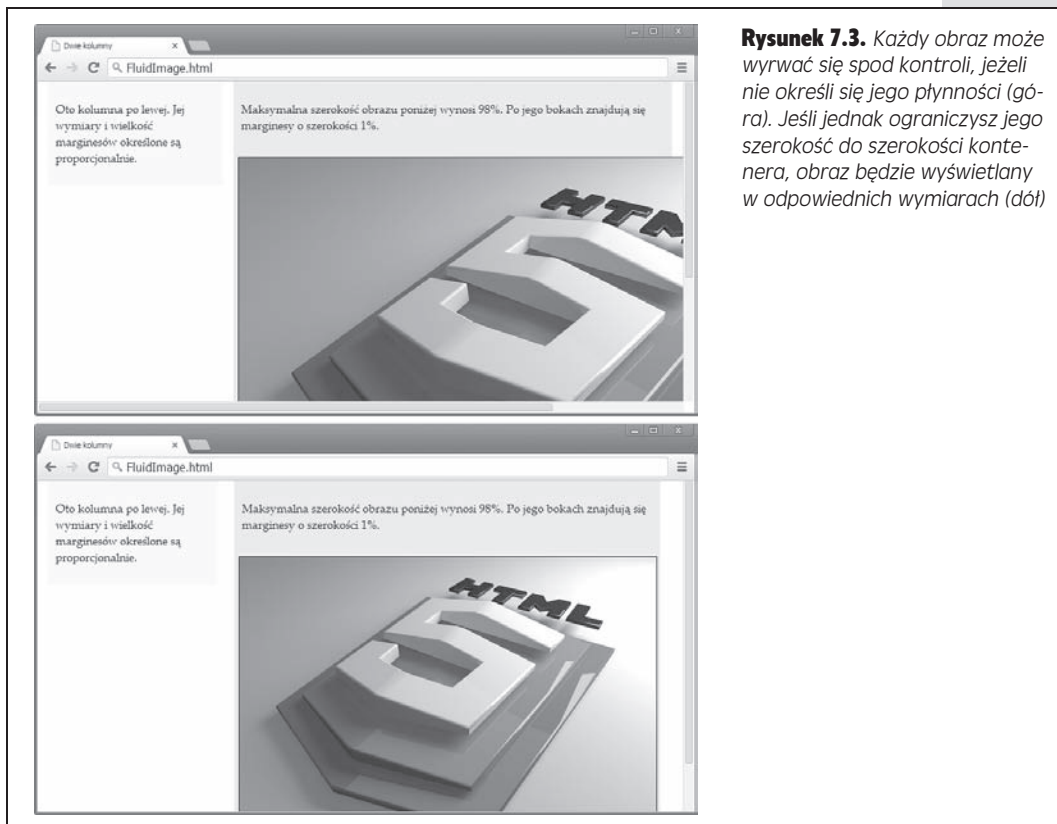
które mają być przeznaczone na marginesy. Nieuważny webdeveloper może ominąć ten problem, zwężając kolumnę do 65%, co prowadzi do widocznej na rysunku 7.1 niespójności w rozmieszczeniu elementów. W CSS3 można jednak nadać własności `width` wartość `calc(67%-5px)`, dzięki której kolumna ma taką szerokość, jaką powinna — i ani piksela więcej.

W obu przypadkach wprowadzone rozwiązanie może jednak pogorszyć sytuację. Ustawienie `box-sizing` nie działa w IE 7, a w Firefoksie wymaga użycia przedrostka `-moz-` (jak wspomniano w rozdziale 6., w punkcie „Style właściwe konkretnym przeglądarkom”). Funkcja `calc()` nie działa w IE 7, IE 8 i przeglądarce Androida sprzed Chrome, a starsze wersje Safari wymagają podania przedrostka `-webkit-`. Problemem z obsługą można wprawdzie zaradzić za pomocą wypełnień, ale na razie łatwiej unikać korzystania z tych funkcji w ogóle, dopóki więcej ludzi nie przeniesie się na nowocześniejsze przeglądarki.

Jak zwykle, wartość 100% odnosi się do kontenera elementu. W tym przypadku jest to kolumna, w której zawarty jest obraz, a nie cała strona. Twój obraz może teraz osiągnąć *albo* swój maksymalny rozmiar, *albo* rozmiar ograniczony wielkością swojego kontenera, w zależności od tego, która z tych wartości jest mniejsza (rysunek 7.3).

Wskazówka: Jeśli zdecydujesz się utworzyć margines wokół obrazu, dopilnuj, by wartości procentowe własności `margin-left`, `margin-right`, oraz `max-width` sumowały się do 100% (a już na pewno nie przekraczały tej wartości).

Jednym ograniczeniem płynnych obrazów jest to, że przeglądarka musi pobrać obraz w pełnej wielkości, niezależnie od wymiarów, w jakich ma być wyświetlany. Wiąże się to z niewielkim, choć i tak zbędnym zużyciem czasu i transferu, co jest niedogodne dla użytkowników urządzeń mobilnych. Niestety, CSS nie poradzi sobie z tym problemem samodzielnie. Istnieją jednak pewne rozwiązania, w ramach których wykorzystuje się połączenie kodów działających po stronie serwera, serwisów internetowych oraz bibliotek JavaScriptu. Jeśli zamierzasz dostarczać użytkownikom urządzeń mobilnych bogatą w pliki graficzne stronę internetową, powinieneś rozważyć skorzystanie z technik, których omówienie znajdziesz w artykule Smashing Magazine pod adresem <http://tinyurl.com/responsive-img>. (Trzeba jednak przyznać, że wielu developerów w ogóle nie zwraca sobie tym głowy). Na szczęście, istnieje również sposób na rozwiązanie analogicznego, choć dużo poważniejszego problemu z wymiarami filmów na urządzeniach mobilnych (o którym przeczytasz w ramce na końcu tego rozdziału).



Rysunek 7.3. Każdy obraz może wyrwać się spod kontroli, jeżeli nie określi się jego płynności (góra). Jeśli jednak ograniczysz jego szerokość do szerokości kontenera, obraz będzie wyświetlany w odpowiednich wymiarach (dół)

Płynna typografia

Skoro utworzyłeś już płynny layout z obrazami o właściwie określonych wymiarach, czas zająć się tekstem w kolumnach. Niedoświadczeni webdeweloperzy często wybierają dla swojego tekstu adekwatny, ustalony rozmiar (określany wartościami pikselowymi) i na tym poprzestają. Takie sztywno określone wielkości nie przystają jednak do modelu skalowalnego projektowania układów stron, ponieważ tekst, który wygląda dobrze na ekranie komputera, jest niemal niewidoczny na urządzeniach mobilnych. Choć użytkownik zawsze może przybliżyć widok i przeczytać tekst, celem projektowania skalowalnego jest utworzenie strony pasującej do każdego okna, która nie wymaga częstego przybliżania i przewijania w poziomie.

I znowu rozwiązanie polega na unikaniu ustalonych jednostek miary w rodzaju pikseli i punktów. Zamiast tego wielkość tekstu należy ustalać względnie, korzystając z wartości procentowych i firetów. Najpopularniejszym wyborem jest firet, znany również jako jednostka **em**, której nazwa odnosi się do wielkiej litery M.

Uwaga: Firet od dawna służy do określania szerokości w typografii drukowanej. Pojęcie *em dash* (z ang. pauza) pierwotnie odnosiło się do poziomego znaku o szerokości równej wielkiej literze M danego kroju.

Wartości procentowe pozwalają na uzyskanie takiego samego wyniku: wielkość tekstu określana jest względem domyślnej wielkości tekstu przeglądarki. Jeśli wybierzesz rozmiar tekstu równy 110% lub 1.1em, otrzymasz pismo o 10% większe od zwyczajnego, nieobstylowanego tekstu. Wybierz rozmiar tekstu równy 50% lub 0.5em, aby zmniejszyć znaki o połowę.

Choć nie jest istotne, czy skorzystasz z wartości procentowych czy firetów, najbardziej elastyczne layouty bazują na jednej, określonej konwencji. W większości z nich wielkość bazowa tekstu na stronie wynosi 100% (wskazując, że jest to punkt odniesienia dla pozostałych rozmiarów), a wielkość tekstu innych elementów jest dodatkowo zwiększana lub zmniejszana firetami:

```
body {
  font-size: 100%
}

p {
  font-size: 0.9em
}

h1 {
  font-size: 2em
}
```

Doświadczeni webdeveloperzy nie poprzestają na tym, lecz używają firetów przy definiowaniu wszystkich innych wielkości w swoich layoutach. Jeśli np. gdzieś w głębi Twojego layoutu znajduje się drobny margines lub odstęp, lepiej zdefiniować go przy użyciu firetów, a nie pikseli. Takiego rodzaju wymiary są wtedy dopasowane do wielkości tekstu. Jest to dość subtelny czynnik, ale pozwala na dokładniejsze wyczyszczenie strony.

Wyobraź sobie np., że utworzyłeś dwupoziomowy layout, w którym element <div> osadzony jest w lewej kolumnie. Służy on do utworzenia dodatkowych odstępów dookoła treści bez przełamывania proporcjonalnego układu kolumn:

```
<body>
  <div class="leftColumn">
    <div class="leftColumnContent">
      ...
    </div>
  </div>

  <div class="rightColumn">
    ...
  </div>
</body>
```

Możesz określić obramowanie, marginesy i odstępy elementu <div> w lewej kolumnie przy użyciu wartości pikselowych. Taki layout jest sprawny i też powinien być płynny. Jednak lepiej użyć firetów, jak w poniższym przykładzie:

```
.leftColumn {
  width: 28.6%;
  background: #FFFCC;
  float: left;
}

.rightColumn {
  width: 71.4%;
  background: #CCFFCC;
  float: left;
}
```

```

}

.leftColumnContent {
  border: 0.07em solid gray;
  margin: 0.3em;
  padding: 0.2em 0.3em 0.4em 0.4em;
}

```

Uwaga: Największą korzyścią z używania firetów do określania obramowań, marginesów i odstępów często jest to, że elementy te nie robią się za duże w małych oknach i nie dominują nad layoutem na urządzeniach mobilnych.

WIZJE PRZYSZŁOŚCI

CSS3: od em do rem

Gdy webdesignerzy używają tekstu w złożonych, skalowalnych layoutach, muszą się mierzyć z pewnym dziwactwem. Jednostki tekstowe o proporcjonalnych wartościach, takie jak firety i procenty, określają wielkość pisma w odniesieniu do kontenera. Nie sprawia to problemów w prostych przykładach ukazanych w punkcie „Płynna typografia”, ponieważ kontenerem jest element `<body>`, w którym zawarta jest cała strona, bądź inny element, dziedziczący jego ustawienia fontu. Bólu głowy można dostać dopiero przy określaniu proporcjonalnych wielkości na *wielu* poziomach layoutu.

Załóżmy np., że utworzyłeś element `<div>` i nadałeś jego tekstowi wielkość `1.1em`. Następnie zamieszczasz w elemencie nagłówek `<h1>` o wielkości `2em`. Pewnie spodziewasz się, że tekst nagłówek będzie dwukrotnie większy od tekstu domyślnego, ale w rzeczywistości jest on dwukrotnie większy od tekstu kontenera, którego wielkość wynosi `1.1em`. Zatem nagłówek jest `2,2` razy większy od tekstu domyślnego.

Aby zapobiec wystąpieniu efektu kumulacji, musisz zachować rygor przy doborze miejsc, w których nada-

jesz tekstowi wielkość. Najlepiej robić to tylko na jednym poziomie layoutu. W CSS3 pojawiła się nowa jednostka, która pozwala na ominięcie tego problemu. Jest to `rem` (skrót od „root em”, czyli em elementu głównego dokumentu). Jednostka `rem` jest w zasadzie taką samą jednostką względną jak `em`, z tym że — niezależnie od swojego położenia w arkuszu stylów — jej wartość zawsze jest obliczana w odniesieniu do wielkości tekstu elementu `<html>`, a nie kontenera. Wobec tego wartość `2rem` zawsze jest równa dwukrotności wielkości domyślnego tekstu, niezależnie od tego, gdzie się ją wprowadzi.

Jednostki `rem` są zdumiewająco dobrze obsługiwane — działają w każdej nowoczesnej przeglądarce. Problemy sprawiają jedynie dobrze znani maruderzy, czyli przeglądarki IE 8 i IE 7, które w ogóle nie rozumieją tych jednostek. Choć wypełnienie tej luki przy użyciu JavaScriptu jest możliwe (<http://tinyurl.com/rem-polyfill>), większość rozsądnych webdeweloperów woli unikać dodawania kolejnych skryptów, żeby tylko można było korzystać z nowych jednostek, i decyduje się dalej używać nieco niewygodnych jednostek `em`.

Oczywiście, typografia nie kończy się na definiowaniu wielkości tekstu. Aby utworzyć tekst, który pozostanie czytelny na różnorodnych wyświetlaczach, musisz też wziąć pod uwagę szerokość wiersza, marginesy, interlinię, a nawet rozmieszczenie tekstu w kolumnach (co przedstawiono w punkcie „Wielokolumnowy tekst” rozdziału 6.). Żadnym z tych elementów nie można się zająć, jeśli używa się zwyczajnych, płynnych layoutów i określa wielkości wartościami proporcjonalnymi. *Możesz* jednak utworzyć bardziej elastyczne arkusze stylów, precyzujące takie szczegóły za pomocą zapytań medialnych. Wkrótce dowiesz się więcej na ten temat, ale najpierw trzeba rozpatrzyć kwestię automatycznego skalowania stron na telefonach komórkowych.

Widok strony: obsługa layoutu na smartfonach

Strona z dwiema kolumnami, którą omówiliśmy wcześniej, teoretycznie mogłaby się zmieścić w oknie o dowolnym rozmiarze. W praktyce jednak trzeba wziąć pod uwagę inne utrudnienie związane z korzystaniem z małych urządzeń mobilnych: rozmiar **widoku strony** (ang. *viewport*).

Apple wprowadził kiedyś widoki strony po to, żeby iPhone'y sensownie wyświetlały zwyczajne strony internetowe, które nie były oparte na technikach elastycznego projektowania. Zamiast wyświetlać niewielki fragment dużej strony internetowej, przeglądarki mobilne, takie jak Safari, pokazują całą stronę w pomniejszeniu, by uchwycić na ekranie więcej treści. Taki oddalony obszar wyświetlania określa się właśnie mianem widoku strony.

Technika ta opiera się na kompromisie. Z jednej strony sprawia, że strona wygląda tak, jak wyglądałaby na komputerze, ale przez to jednocześnie większość zwyczajnego tekstu staje się nieczytelna. Nie trzeba przewijać ekranu, ale trzeba częściej przybliżać i oddalać widok. Technika ta ułatwia poruszanie się po stronie, ale utrudnia odczytywanie treści.

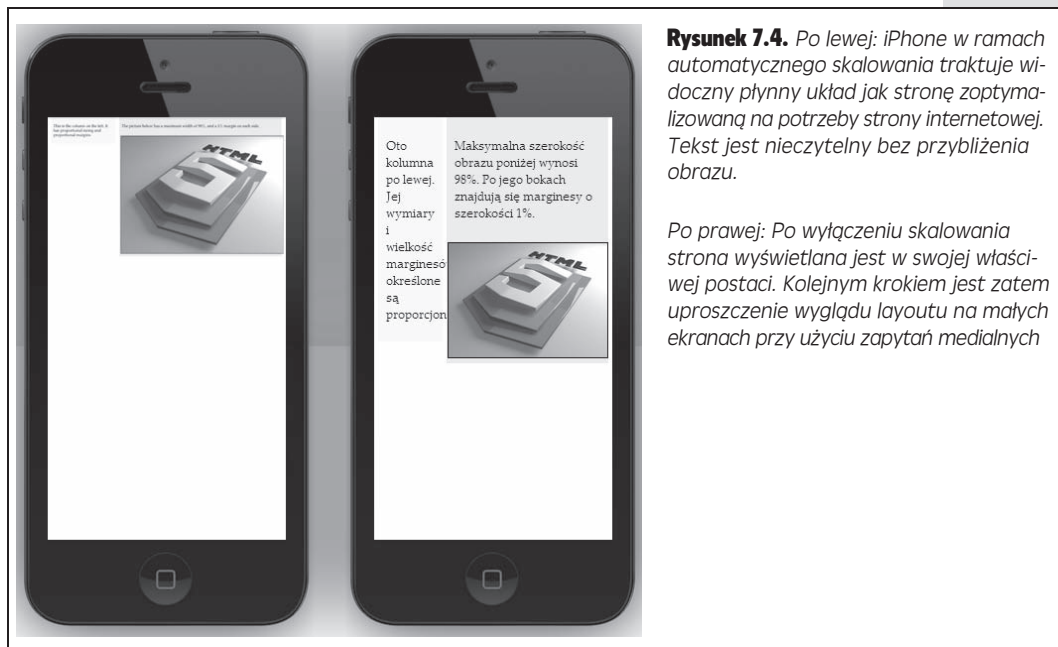
Uwaga: Choć widoki strony wprowadził Apple, obecnie korzystają z nich wszyscy inni producenci oprogramowania na urządzenia mobilne. Różnice dotyczą jedynie wielkości widoków i obszarów stron, jakie są włączane w ich obręb.

Jeśli tworzysz tradycyjną witrynę na przeglądarki komputerowe, nie musisz się przejmować ustawieniami widoków strony urządzeń. One same dopilnują, by Twoja strona wyglądała w miarę dobrze na małym ekranie telefonu (choć trzeba przyznać, że pomniejszona strona nie jest dla użytkowników szczególnie wygodna). Z drugiej strony, jeśli chcesz zaangażować się w projektowanie elastyczne i utworzyć witrynę przyjazną dla urządzeń mobilnych, musisz wprowadzić zmiany w opcjach widoków stron. Musisz polecić przeglądarkom mobilnym, by *nie skalowały* strony automatycznie. Można to wykonać, zamieszczając następujący element `<meta>` w sekcji `<head>` dokumentu:

```
<meta content="initial-scale=1.0" name="viewport">
```

Powyższy znacznik wskazuje urządzeniom mobilnym, by wyświetlały stronę w jej rzeczywistej skali, bez oddalania obrazu. Nowy iPhone umieści Twoją stronę w rzeczywistym rozmiarze w oknie o szerokości 320 pikseli. Bez takiego dostosowania skali iPhone przypisałby stronie 980 pikseli szerokości i wyświetlił ją w pomniejszeniu, by zmieściła się w oknie. Różnicę widać na rysunku 7.4.

Uwaga: Zapewne wiesz, że istnieje wiele symulatorów online, pozwalających na sprawdzenie wyglądu strony na różnych urządzeniach mobilnych. Na <http://mobiletest.com> można np. porównać wygląd strony na najnowszych iPhone'ach, iPadach i urządzeniach z Androidem. W większości z nich nie można jednak odtworzyć automatycznego skalowania. Innymi słowy, strona w symulatorze może działać tak, jakbyś przypisał wejściowej skali wartość 1.0 w ukazanym powyżej elemencie `<meta>`. Jeśli tego nie zrobiłeś, nie uzyskasz trafnego odwzorowania tego, co zobaczysz na samym urządzeniu, więc pozostań czujny na takie szczegóły.



Dostosowywanie layoutu przy użyciu zapytań medialnych

Wiesz już, jak utworzyć płynny layout, który rozszerza się i ścieśnia, aby dopasować się do okna przeglądarki o dowolnych wymiarach. Przyjmując takie podejście, zapewniasz, że strona będzie pasować do każdego okna. Nie zyskujesz jednak pewności, że w każdym oknie będzie wyglądać dobrze.

Skonstruowane w prosty sposób płynne layouty często psują się w skrajnych warunkach. W małym oknie liczne kolumny stają się żenująco wąskie, ścieśniając tekst i obrazy do nieczytelności. W bardzo dużym oknie kolumny stają się przygnębiająco szerokie — trudno nie zgubić się, próbując odczytywać rozciągnięte na pół strony wiersze.

Jednym sposobem na rozwiązanie tych problemów jest określenie ograniczeń rozciągania i ścieśniania layoutu. Wykonać to można przy użyciu własności `max-width` i `min-width`. Jeśli strona zostanie rozciągnięta ponad swój górny limit, po jej prawej pojawi się dodatkowy margines. Jeśli strona zostanie ścieśniona do rozmiaru mniejszego od dolnego limitu, kolumny zachowają swoją minimalną szerokość, a przeglądarka wyświetli paski przewijania. Ustawienia minimalnej i maksymalnej szerokości zapewniają podstawową ochronę przed wyświetlaniem ułomnych layoutów. Trzeba jednak przyznać, że jednocześnie zmniejszają elastyczność układu strony. Jeśli np. Twoja strona nie może zmniejszyć się do rozmiarów okna iPhone'a, nie będzie zbyt przydatna dla użytkowników korzystających z urządzeń mobilnych.

Lepszym rozwiązaniem jest subtelne zmodyfikowanie *struktury* layoutu przy zmianie wielkości strony. Do bardzo małego okna pasowałby przykładowo uproszczony layout bez pasków bocznych czy paneli z reklamami. Z kolei bardzo duże okno daje sposobność do powiększenia tekstu lub podzielenia go na wiele kolumn (punkt „Wielokolumnowy tekst” rozdziału 6.).

Odpowiedzią na to zapotrzebowanie są **zapytania medialne**. Ta własność CSS3 pozwala dynamicznie zmieniać formatowanie dla różnych urządzeń i ustawień wyświetlania. Użyta w odpowiedni sposób pozwoli Ci poprawnie wyświetlać strony na szerokich monitorach komputerów stacjonarnych i iPhone'ach — bez modyfikowania linijki kodu HTML.

Zapytania medialne

Zapytania medialne działają w oparciu o jedną z cech charakterystycznych urządzenia, które obsługują (takich jak wielkość ekranu, rozdzielczość, liczba wyświetlanych kolorów itd.). W oparciu o te dane możesz nałożyć różne style czy nawet inny ich arkusz.

Zapytanie medialne jest, najprościej rzecz ujmując, osobną sekcją arkusza stylów. Zaczyna się od terminu `@media`, po którym znajduje się warunek w nawiasie, a następnie seria powiązanych stylów w nawiasie klamrowym. Oto jego typowa struktura:

```
@media (media-feature-name: value) {
    /* Tutaj znajdują się nowe style. */
}
```

Zapytanie medialne przypomina instrukcję warunkową w JavaScriptcie. Jeśli bieżąca przeglądarka spełnia warunki określone w nawiasie, zawarte w zapytaniu style stają się obowiązujące. Jeżeli jednak przeglądarka nie spełnia tych warunków, style są ignorowane.

Uwaga: Style znajdujące się poza sekcją `@media` nadawane są zawsze, niezależnie od okoliczności. Warunkowe style zapytania medialnego są nadawane jako *dodatek* do innych stylów. Z tego względu style warunkowe często muszą nadpisywać pozostałe ustawienia, np. ukrywając coś, co było poprzednio widoczne, przenosząc sekcję do nowego miejsca, nadając tekstowi nową wielkość itp.

Aby skorzystać z zapytania medialnego, musisz się zorientować, jakiego rodzaju warunki można utworzyć. Standard zapytań pozwala na sprawdzenie różnych szczegółów, zwanych **właściwościami mediów** (ang. *media features*). Możesz np. sprawdzić szerokość obszaru wyświetlania strony, a następnie podmienić style, gdy stanie się on mniejszy od jakiejś wartości. W tabeli 7.1 znajduje się lista najczęściej używanych właściwości mediów. (Istnieją też autorskie, eksperymentalne właściwości mediów, które nie są powszechnie obsługiwane i z tego względu nie znalazły się w tabeli).

Tabela 7.1. Właściwości, które przydają się najbardziej przy tworzeniu zapytań medialnych

Nazwa właściwości	Wartość	Zwykle służy do...
width min-width max-width	Szerokość obszaru wyświetlania (lub drukowania).	zmiany layoutu, by dostosować go do bardzo wąskich (takich jak smartfon) albo bardzo szerokich wyświetlaczy.
height min-height max-height	Wysokość obszaru wyświetlania.	zmiany layoutu, by dostosować go do bardzo wysokich lub krótkich wyświetlaczy.
device-width min-device-width max-device-width	Pełna szerokość ekranu komputera lub innego urządzenia (bądź pełna szerokość zadrukowanego arkusza).	dostosowania layoutu tak, by reagował na konkretne urządzenia, takie jak smartfony.
device-height min-device-height max-device-height	Pełna wysokość ekranu lub arkusza.	dostosowania layoutu tak, by reagował na konkretne urządzenia, takie jak smartfony.
orientation	Jedna z dwóch wartości: landscape lub portrait.	zmiany layoutu na potrzeby różnych orientacji (poziomej i pionowej) ekranu.
device-aspect-ratio min-device-aspect-ratio max-device-aspect-ratio	Proporcje obszaru wyświetlania, np. format obrazu 1/1 jest idealnym kwadratem.	dostosowania stylów, by pasowały do ekranów o różnych wymiarach (choć dość szybko wynikają z tego dalsze komplikacje).
color min-color max-color	Liczba bitów koloru. 1-bitowa głębia koloru odpowiada obrazowi czarno-białemu, podczas gdy nowoczesne wyświetlacze standardowo mają 24-bitową głębię koloru, która przekłada się na miliony barw.	określenia poziomu obsługi kolorów (np. w wersji strony internetowej do druku).

KĄCIK WSPOMNIENI

Typy mediów CSS

Ciekawe jest to, że już w edycji CSS 2.1 twórcy standardu poczynili pierwszą próbę rozwiązania problemu urządzeń mobilnych — wprowadzili tzw. typy mediów. Być może zdarzyło Ci się wcześniej z nich korzystać w celu dodania oddzielnego arkusza stylów dla druku stron.

```
<head>
...
<!-- Używa tego arkusza do wyświetlania strony
na ekranie -->
<link rel="stylesheet" media="screen"
href="styles.css">
<!-- Używa tego arkusza do wydrukowania
strony. -->
```

```
<link rel="stylesheet" media="print"
href="print_styles.css">
</head>
```

Atrybut `media` przyjmuje również wartość `handheld` — co dotyczy małych ekranów telefonów komórkowych z ograniczonym dostępem do Internetu. Większość mobilnych urządzeń zwraca uwagę na ten atrybut i używa odrębnego arkusza — jeśli istnieje. Niestety, atrybut `media` nie funkcjonuje do końca sprawnie i nie wystarcza, by poradzić sobie z wyborem i różnorodnością urządzeń, które obecnie istnieją. Metoda ta działa wciąż wyśmienicie, jeśli chodzi o wydruki.

Utworzenie prostego zapytania medialnego

Większość właściwości mediów występuje w kilku wersjach i pozwala na określenie górnych i dolnych limitów. Limity te są ważne, ponieważ większość zapytań medialnych odnosi się do sprecyzowanych zakresów wartości.

Przed wykorzystaniem zapytań medialnych wybierz własność, którą będziesz analizował. Jeśli np. chcesz utworzyć nowy zestaw stylów, który ma obowiązywać dla wąskich okien, powinieneś wybrać właściwość `max-width` i określić stosowny limit. W omawianym przykładzie nowy zestaw stylów wchodzi w życie, jeśli szerokość okna przeglądarki jest mniejsza niż 480 pikseli:

```
@media (max-width: 480px) {
  ...
}
```

Wskazówka: Aktualnie najpopularniejszymi zapytaniami są `max-device-width` (używane do formatowania stron na urządzenia przenośne), `max-width` (wykorzystywane do zmiany układu w zależności od rozmiaru okna) oraz `orientation` (stosowane w celu zmiany wyglądu strony w zależności od ułożenia smartfona w pionie lub poziomie).

Aby przeprowadzić prosty test, użyj zapytania medialnego do wprowadzenia widocznej zmiany. Poniższy kod zmienia kolor tła kolumny:

```
@media (max-width: 480px) {
  .leftColumn {
    background: lime;
  }
}
```

Sprawdź, czy zapytanie medialne działa. Zmniejsz powoli okno swojej przeglądarki. Gdy obszar wyświetlania stanie się węższy niż 480 pikseli, nowy styl uaktywni się, a kolumna zmieni barwę na limonkową. Wszystkie pozostałe reguły stylów określone dla klasy `leftColumn` (np. jej rozmiar i położenie) wciąż obowiązują, ponieważ zapytanie mediów ich nie nadpisuje.

Uwaga: Przeglądarki, które nie rozpoznają zapytań medialnych (np. Internet Explorer 8), po prostu ignorują nowe style i będą nakładać stare, niezależnie od tego, jak zmieni się wielkość okna.

Jeżeli chcesz, możesz dodać pomocniczą sekcję zapytań, która zniesie te nowsze reguły po jeszcze bardziej radykalnym zmniejszeniu wielkości okna. Przykładowo sekcja ta użyłaby nowego formatowania, gdy szerokość przeglądarki spadła poniżej 250 pikseli.

```
@media (max-width: 250px) {
  ...
}
```

Uważaj na to, że nowe style unieważniają wcześniej nałożone — czyli wszystkie właściwości ustawione w zwykłej deklaracji i w zapytaniu medialnym dla sekcji mniejszych niż 450 pikseli. Jeśli wydaje Ci się to zbyt skomplikowane, nie martw się. W dalszej części rozdziału dowiesz się, jak pracować z bardziej złożonymi medialnymi zapytaniami. Najpierw jednak zajmiemy się praktyczniejszym przykładem.

Layout przyjazny dla urządzeń mobilnych

Zapytania medialne są fundamentem witryny wyglądającej równie przyzwoicie w przeglądarce na komórce, jak w przeglądarce komputerowej. Wystarczy po nie zwyczajnie sięgnąć.

Na rysunku 7.5 widnieje podrasowana wersja strony *ApocalypseSite.html*, którą pamiętasz z rozdziału 2. Pierwotna strona oparta była na sztywnym layoutcie o ustalonych szerokościach kolumn. Do utworzenia zmodyfikowanej wersji zastosowano wszystkie przedstawione w tym rozdziale techniki. Zbudowana jest na płynnym layoutcie z elementami o proporcjonalnie określonych wymiarach (punkt „Płynny layout”), który dopasowuje się do każdej szerokości okna. Marginesy, odstępy, obramowania i wielkości tekstu określone są w nim firetami, dzięki czemu wszystkie te komponenty zachowują spójne wymiary na różnych urządzeniach (punkt „Płynna typografia”). Obraz nagłówka strony rozszerza się i kurczy, dopasowując się do dostępnego obszaru, a obraz reklamy w pasku bocznym, dzięki zastosowaniu techniki płynnych obrazów, nigdy nie wychodzi poza swoje pole (punkt „Płynne obrazy”). Użyto tu również metody z elementem `<meta>`, by zapobiec oddalaniu obrazu w przeglądarkach mobilnych (punkt „Widok strony: obsługa layoutu na smartfonach”).

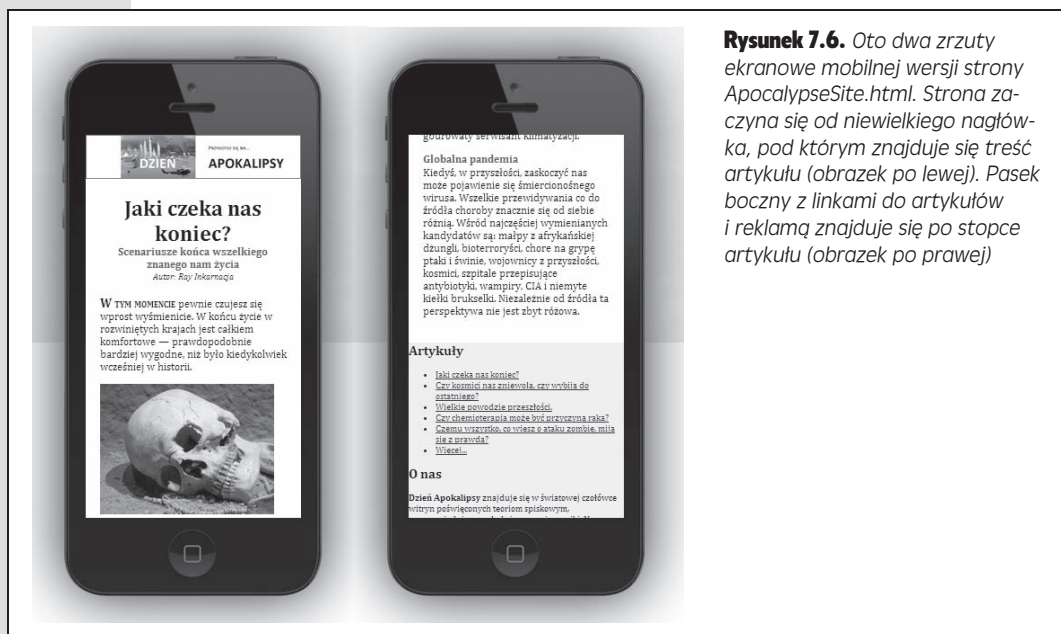


Rysunek 7.5. Widoczna tutaj strona oparta jest na najlepszych technikach elastycznego projektowania. Leżący u jej podstaw kod CSS możesz pobrać ze strony helion.pl/ksiazki/h5m2.htm

Mówiąc krótko, nowa strona *ApocalypseSite.html* jest przystosowana do przeglądania na komórkach. Jej layout wciąż jednak nie jest przyjazny urządzeniom przenośnym. Jest tak, ponieważ bez względu na to, jak mocno dwie umieszczone obok siebie kolumny się ścieśniają, nie zmieszczą się schludnie w małym oknie. Musisz użyć zapytania medialnego, żeby zarządzić temu efektowi.

Zanim zaczniesz modyfikować swój arkusz stylów, musisz zastanowić się, jak mobilna wersja Twojej strony ma właściwie wyglądać. Strony na komórki zwykle zwiężają się do pojedynczej kolumny. Paski boczne są wtedy albo ukrywane

zupnie, albo przenoszone nad lub pod treść główną. Na rysunku 7.6 widnieje uporządkowana wersja *ApocalypseSite.html* na iPhone'a.



Rysunek 7.6. Oto dwa zrzuty ekranowe mobilnej wersji strony *ApocalypseSite.html*. Strona zaczyna się od niewielkiego nagłówka, pod którym znajduje się treść artykułu (obrazek po lewej). Pasek boczny z linkami do artykułów i reklamą znajduje się po stopce artykułu (obrazek po prawej)

Utworzenie mobilnej wersji *ApocalypseSite.html* jest zaskakująco proste. Zmniejszenie nagłówka witryny i dopasowanie tekstu następuje automatycznie za sprawą użycia płynnych obrazów i jednostek em. Pozostaje jedynie sprawić, by zapytanie medialne przestawiło kolumny.

Wygląd obydwu kolumn był pierwotnie zdefiniowany następującymi regułami:

```
.NavSidebar {
  float: left;
  width: 22%;
  font-size: small;
}
.Content {
  float: left;
  width: 78%;
}
```

Pasek boczny sływa na lewo i ma szerokość 22%. Pole z treścią główną znajduje się zaraz obok niego i ma szerokość 78%.

Ponieważ layout przestaje działać sprawnie na wąskich ekranach, warto skorzystać z popularnej właściwości mediów `max-width`. Jak dowiedziałeś się w poprzednim punkcie, `max-width` sprawdza bieżącą szerokość strony w oknie przeglądarki. Jeśli wartość ta jest niewielka, wyświetlanie dwóch kolumn jest nieodpowiednim rozwiązaniem.

Oto zapytanie medialne, za sprawą którego pola przestają pływać, a kolumny przyjmują całą dostępną szerokość okna:

```
@media (max-width: 568px) {
  .NavSidebar {
    float: none;
    width: auto;
  }

  .Content {
    float: none;
    width: auto;
  }
}
```

Powyższe style nadawane są jako dodatek do zdefiniowanych już stylów. Z tego względu konieczne może być przywrócenie zmodyfikowanym własnościom ich domyślnych wartości. W tym przykładzie style zapytań medialnych przywracają wartość `none` własności `float` i wartość `auto` własności `width` (choć równie dobrze można by jej nadać wartość `100%`). Są to wartości domyślne, ale podane pierwotnie reguły stylów paska bocznego zmodyfikowały je. Zauważ też, że pierwotne reguły stylów `NavSidebar` określały wielkość tekstu. Zapytanie medialne nie zmienia tej reguły, wobec czego ona nadal obowiązuje.

To zapytanie medialne, mówiąc ściśle, określa style odnoszące się do każdego wąskiego okna, niezależnie od tego, czy jest to okno przeglądarki na komórce, czy pomniejszone okno przeglądarki komputerowej. To zupełnie sensowne rozwiązanie, ale style można dodatkowo uszczegółowić i utworzyć osobne reguły dla małych okien na komputerach oraz osobne dla urządzeń mobilnych. Warunki dla małych okien określa się właściwością `max-width`, a do wykrywania urządzeń mobilnych używa się właściwości `max-device-width` (o których wspomniano w tabeli 7.1).

Wskazówka: Określenie punktu, w którym następuje przełączenie na uproszczony layout, należy do Ciebie, ale warto zdecydować się na granicę 568 pikseli. Wynika to z tego, że 568 pikseli to szerokość strony na zorientowanym poziomo iPhone'ie. Dotyczy to również urządzeń działających na systemie Android (tabela 7.2).

W tym przykładzie trzeba wprowadzić jeszcze jedną poprawkę. W pierwotnej wersji strony sekcja `NavSidebar` zamieszczona była w kodzie HTML przed sekcją `Content`. Dzięki temu obydwie mogły spływać w lewo, a `NavSidebar` znajdował się przy lewej krawędzi strony. Po usunięciu przypisanych wartości `float` wersja mobilna witryny musi wyświetlać sekcje w takiej kolejności, w jakiej są umieszczone w pliku, przez co `NavSidebar` znajduje się na górze strony, a dopiero pod nim wyświetlana jest sekcja `Content`. Taki layout może zniechęcać użytkowników komórek, ponieważ przewijanie linków i reklam przed dotarciem do właściwej treści strony nie jest szczególnie wygodne.

Rozwiązaniem jest opracowanie kodu spełniającego wymogi mobilnej wersji strony, aby dopiero *potem* zdefiniować dodatkowe reguły CSS, tworzące bardziej rozbudowany, wielokolumnowy layout. Ta wzorcowa technika jest zgodna z projektowaniem **ukierunkowanym na urządzenia przenośne** (ang. *mobile-first*).

W omawianym przykładzie zastosowanie jej wiąże się z zamieszczeniem sekcji `Content` przed sekcją `NavSidebar`. Rozwiązuje to problem z mobilną wersją przeglądarki, ale jednocześnie w pełnej wersji strony spycha pasek boczny do prawej kolumny. Aby naprawić tę usterkę, postaraj się, by sekcja `Content` spływała w prawo:

```
.Content {
  float: right;
  width: 78%;
}
```

Sekcja Content powraca teraz na swoje miejsce po prawej, a NavSidebar trzyma się lewej strony, dzięki czemu przywrócony zostaje pełnowymiarowy layout z rysunku 7.5.

PORADNIK INŻYNIERA

Ukrywanie i zastępowanie sekcji

Jeśli jesteś szczególnie ambitny, możesz wprowadzić o wiele więcej zmian, aby odróżnić wersję strony na urządzenia mobilne od jej pełnego wydania. Możesz np. skorzystać z własności CSS `display` do ukrywania i wyświetlania całych sekcji strony.

Najpierw jednak musisz wziąć pod uwagę wady takiego rozwiązania. Jeśli chcesz podmienić duże fragmenty strony, prawdopodobnie uzyskasz rozbudowany kod, o którego utrzymanie, spójność i przetestowanie na różnych urządzeniach będziesz musiał później zadbać. Co więcej, przeglądarki pobierają obrazy zamieszczone w ukrytych sekcjach, choćby i miały ich nigdy nie wyświetlić. W przypadku urządzeń mobilnych takie zachowanie może poważnie obniżyć wydajność i niepotrzebnie obciążać łącze.

Istnieje jednak sytuacja, w której podmiana sekcji jest wskazana: dzieje się tak, kiedy pojawia się konieczność zastąpienia rozbudowanego systemu nawigacyjnego skromniejszą, prostszą wersją na urządzenia przenośne. Dostarczanie użytkownikom mobilnym rozwijanych list nawigacyjnych zamiast nieporęcznych drzewek linków jest standardową praktyką. (Istnieje nawet sprytna technika, pozwalająca na przekształcenie rzędu odnośników w rozwijaną listę; znajdziesz ją na stronie <http://css-tricks.com/convert-menu-to-dropdown>).

Czasami proste sztuczki i drobne zmiany nie wystarczają. Konieczne może się okazać bardziej radykalne przekształcenie mobilnej wersji witryny. Masz do wyboru szereg rozwiązań o różnych stopniach złożoności i zaawansowania. Możesz utworzyć całkowicie osobną witrynę na urządzenia przenośne i umieścić ją w innej domenie („New York Times” tak właśnie robi ze swoją wersją na urządzenia mobilne <http://mobile.nytimes.com>). Wymaga to dużego nakładu pracy i uruchomienia na serwerze systemu zarządzania treścią, żeby zasoby dostępne na stronie mobilnej pokrywały się z zasobami standardowej witryny. Innym sposobem jest napisanie działającego po stronie serwera kodu, który sprawdza każde żądanie, identyfikuje przeglądarkę, z której użytkownik korzysta, i przesyła odpowiednie treści. To rozwiązanie jest idealne, jeśli dysponujesz odpowiednim czasem i umiejętnościami.

Skromniejsza metoda polega na zastosowaniu narzędzia JavaScriptu, pozwalającego dynamicznie modyfikować stronę, w zależności od wyświetlacza. Jednym z nich jest Modernizr; za pomocą jego metody `Modernizr.mq()` można testować zawarte w kodzie zapytania medialne (<http://modernizr.com/docs>). Korzystanie z takiego narzędzia jest efektywniejsze od używania zapytań medialnych, ale prowadzi do zwiększenia złożoności układu strony.

Na tym etapie możesz rozważyć dodanie kolejnego zapytania medialnego, aby dostosować style także do bardzo szerokich okien, np. dzieląc tekst na wiele kolumn (przy użyciu własności CSS opisanych w rozdziale 6., w punkcie „Wielokolumnowy tekst”) dla zachowania jego czytelności.

Wskazówka: Szukasz inspiracji? Skorzystaj z gotowych, elastycznych szablonów, których jest w sieci pełno. Zacznij od przejrzenia <http://html5up.net>, www.typeandgrids.com lub <http://responsify.it>.

Zapytania medialne — wyższa szkoła jazdy

Czasem do nałożenia konkretnych stylów powinno dojść po spełnieniu kilku odmiennych warunków. Taki scenariusz przedstawiono w poniższym przykładzie:

```
@media (min-width: 400px) and (max-width: 700px) {
  /* Nakłada formatowanie dla okien szerszych niż 400 pikseli i węższych od 700 pikseli. */
}
```

Takie zapytanie sprawdza się, gdy trzeba wprowadzić kilka zestawów wzajemnie wykluczających się stylów, lecz nie chce się mieć kłopotów z nakładaniem warstw reguł. Rzuć okiem na ten przykład:

```
/* Style uniwersalne */
@media (min-width: 600px) and (max-width: 700px) {
  /* Nakłada formatowanie dla okien szerszych niż 600 pikseli i węższych od 700 pikseli. */
}
@media (min-width: 400px) and (max-width: 599.99px) {
  /* Nakłada formatowanie dla okien szerszych niż 400 pikseli i węższych od 600 pikseli. */
}
@media (max-width: 399.99px) {
  /* Nakłada formatowanie dla okien węższych niż 400 pikseli. */
}
```

Od tej pory, gdy okno przeglądarki zmaleje do 380 pikseli, uruchomione zostaną dwa zestawy reguł: style standardowe i style z końcowego bloku @media. To, czy takie rozwiązanie upraszcza, czy komplikuje Ci życie, zależy od tego, co pragniesz osiągnąć. Jeżeli korzystasz z wielu stylów i często je zmieniasz, zaprezentowane tu podejście znacznie ułatwi pracę.

Pamiętaj jednak, że nawet tutaj musisz zwracać uwagę, które własności się nakładają. Jeżeli np. w dwóch miejscach ustawisz maksymalną i minimalną szerokość elementu na 400 pikseli, otrzymasz punkt, w którym oba style będą na siebie nachodzić. Trochę niezręcznym rozwiązaniem jest użycie ułamków, np. 399,99 pikseli, jak to miało miejsce w poprzednim przykładzie.

Innym pomysłem jest wykorzystanie słowa klucza `not`. Między obiema metodami nie ma żadnej widocznej różnicy, ale być może podejście w tym przykładzie będzie Ci lepiej odpowiadać:

```
/* Style standardowe */
@media (not max-width: 600px) and (max-width: 700px) {
  /* Nakłada formatowanie dla okien szerszych niż 600 pikseli i węższych od 700 pikseli. */
}
@media (not max-width: 400px) and (max-width: 600px) {
  /* Nakłada formatowanie dla okien szerszych niż 400 pikseli i węższych od 600 pikseli. */
}
@media (max-width: 400px) {
  /* Nakłada formatowanie dla okien węższych niż 400 pikseli. */
}
```

Istnieje jeszcze jeden sposób na zniesienie formatowania, którego nie podano w przeanalizowanych przykładach. Każdą sekcję zapytań medialnych (@media) otwierają standardowe, uniwersalne reguły. W zależności od wielu czynników, może się okazać, że lepiej będzie całkowicie rozdzielić segmenty arkusza (tak aby np. urządzenia mobilne korzystały z oddzielnego arkusza niezależnych stylów). W tym celu zapytań medialnych należy użyć w zewnętrznych arkuszach. Temu zagadnieniu poświęcono następną część.

Zastępowanie całego arkusza stylów

Jeśli w grę wchodzi jedynie drobne usprawnienia strony, blok @media okaże się przydatny w grupowaniu oddzielnych zestawów stylów w jednym pliku. Jeżeli jednak chcesz wprowadzić znaczne zmiany, może się okazać, że łatwiej będzie Ci zarządzać formatowaniem, kiedy utworzysz odrębny arkusz stylów. Odwołując się do niego, możesz zadeklarować zapytanie medialne.

```
<head>
  <link rel="stylesheet" href="standard.css">
  <link rel="stylesheet" media="(max-width: 568px)"
href="small_styles.css">
  ...
</head>
```

Przeglądarka ściągnie drugi arkusz (*small_styles.css*), lecz nie użyje go dopóty, dopóki szerokość jej okna nie spadnie poniżej maksymalnej wartości parametru width.

W poprzednim przykładzie nowe style usunęły starsze, już nałożone. W niektórych przypadkach lepiej utworzyć całkowicie oddzielne, niezależne arkusze. W tym celu dodaj do standardowego arkusza zapytanie medialne, upewniając się w ten sposób, że zostanie uruchomione tylko dla okien o dużych rozmiarach.

```
<link rel="stylesheet" media="(min-width: 568.01px)" href="standard.css">
<link rel="stylesheet" media="(max-width: 568px)"
href="small_styles.css">
```

Problem z tym rozwiązaniem polega na tym, że przeglądarki, które nie obsługują zapytań medialnych, zignorują *oba* arkusze. W starszych wersjach Internet Explorera usuniesz ten problem, dodając standardowy plik CSS raz jeszcze, wewnątrz specjalnego komentarza.

```
<link rel="stylesheet" media="(min-width: 568.01px)" href="standard.css">
<link rel="stylesheet" media="(max-width: 568px)"
href="small_styles.css">
<!--[if lt IE 9]>
  <link rel="stylesheet" href="standard.css">
<![endif]-->
```

Nie jest to rozwiązanie doskonałe. Starsze wersje Firefoksa (wcześniejsze niż 3.5) nie rozpoznają zapytań medialnych, komentarz warunkowy ich nie dotyczy. Problem ten da się rozwiązać po rozpoznaniu przeglądarki i „podmienieniu” strony przy użyciu kodu JavaScript, lecz nie jest to zbyt eleganckie wyjście. Na szczęście, starsze wersje Firefoksa są coraz rzadziej używane.

Notabene, możesz połączyć medialne zapytania z typami opisanymi w ramce „Typy mediów CSS”. Zacznij od przypisania atrybutowi media typu, ale nie wpisuj go w nawiasach. Oto sposób na zadeklarowanie druku strony o ustalonej wielkości.

```
<link rel="stylesheet" media="print and (min-width: 25cm)"
href="NormalPrintStyles.css" >
<link rel="stylesheet" media="print and (not min-width: 25cm)"
href="NarrowPrintStyles.css" >
```

Rozpoznawanie urządzeń mobilnych

Jak już wiesz, komputery stacjonarne od urządzeń mobilnych odróżnisz, stosując zapytanie medialne z własnością max-device-width. Rodzi się pytanie, jakich szerokości w tym celu użyć.

Telefony komórkowe wykryjesz, sprawdzając, czy parametr `max-device-width` jest mniejszy niż 568 pikseli. Jest to najlepsza, najlepiej potwierdzona zasada. Reguła ta rozpoznaje iPhone'y i współczesne telefony działające w oparciu o system Android:

```
<link rel="stylesheet" media="(max-device-width: 568px)"
href="mobile_styles.css">
```

Jeśli jesteś pasjonatem techniki, w Twoim mózgu powinna zapalić się czerwona lampka. W końcu, aktualna generacja urządzeń mobilnych używa małych ekranów o wysokiej rozdzielczości. Przykładowo iPhone 5 wyświetla siatkę 640×1136 pikseli na raz. Dlatego też może Ci się wydać, że dla niego własność `max-device-width` powinna być większa. Co zaskakujące, tak nie jest.

Rozważ jako przykład urządzenie iPhone 5. Informuje ono przeglądarkę, że szerokość jego ekranu wynosi 320 pikseli w orientacji pionowej, pomimo że jego ekran w rzeczywistości składa się z dwukrotnie większej liczby pikseli fizycznych. Dzięki temu strony internetowe nie przesyłają swoich pełnych, komputerowych wersji na wyświetlacze iPhone'a o szerokości 640 pikseli. Choć iPhone niewątpliwie może wyświetlić taką wersję strony, niewielki rozmiar jego pikseli uniemożliwiłaby użytkownikowi jej odczytanie.

Większość urządzeń w dalszym ciągu tak się zachowuje. Dodają one współczynnik korekcji, tzw. **stosunek pikseli**. W iPhone (w wersji 4. i nowszych) każdy piksel z perspektywy CSS jest równy dwóm pikselom na ekranie, a więc stosunek pikseli jest równy 2. W zasadzie łatwo utworzyć zapytanie medialne, które wykryje iPhone'a 4, ale zignoruje starsze wersje telefonu:

```
<link rel="stylesheet"
media="(max-device-width: 480px) and (-webkit-min-device-pixel-ratio: 2)"
href="iphone4.css">
```

W tabeli 7.2 widnieją szerokości niektórych z najpopularniejszych urządzeń. Pamiętaj, że często trzeba się liczyć z korekcją stosunku pikseli. Tak jest w przypadku iPada, którego wszystkie wersje zgłaszają szerokość 768 pikseli, pomimo że liczba pikseli fizycznych uległa podwojeniu w iPadzie 3.

Tabela 7.2. Standardowe szerokości urządzeń

Urządzenie	Szerokość urządzenia (w orientacji pionowej)	Szerokość urządzenia (w orientacji poziomej)
Apple iPhone 4	320	480
Apple iPhone 5	320	568
Apple iPad	768	1024
Samsung Galaxy S4	360	640
Google Nexus 4	384	640
Kindle Fire	600	1024

Wskazówka: Na rynek stale wychodzą nowe urządzenia. Aktualne informacje o nich znajdziesz na serwisach w rodzaju www.mobitest.me/devices.

Zidentyfikowanie iPada jest wyzwaniem innego typu. Użytkownicy mogą go obracać, co powoduje wyświetlenie treści w pionie lub w poziomie. Choć zmienia to parametr `max-width`, obrócenie urządzenia nie wpływa na własność `max-device-width`. W obu orientacjach — horyzontalnej i wertykalnej — iPad komunikuje, że jego szerokość wynosi 768 pikseli. Na szczęście, nie ma przeciwwskazań, by połączyć właściwości `max-device-width` i `orientation` — pozwoli to dynamicznie zmienić obstylowanie dokumentu w zależności od pozycji urządzenia.

```
<link rel="stylesheet"
  media="(max-device-width: 768px) and (orientation: portrait)"
  href="iPad_portrait.css">

<link rel="stylesheet"
  media="(max-device-width: 768px) and (orientation: landscape)"
  href="iPad_landscape.css">
```

Oczywiście, sposób ten sprawdza się nie tylko w iPadach. Gadżety wyposażone w wyświetlacz o podobnej szerokości (w tym przypadku 768 pikseli lub mniej) otrzymają ten sam zestaw reguł.

Uwaga: Zapytania medialne nie zmieniają zwykłej strony w przyjazną dla użytkowników komórek. Należy wziąć pod uwagę takie czynniki jak wrażenia użytkownika. Dobrym pomysłem może okazać się rozbić treści na mniejsze części (w rezultacie nie trzeba będzie tak często przewijać strony). Zaleca się również unikanie efektów i elementów interakcji, z których trudno skorzystać w interfejsie dotykowym (np. wyskakujących menu).

NIEOSZLIFOWANY DIAMENT

Zapytania medialne i wideo

Najbardziej czytliwą różnicą między witrynami dla komputerów osobistych a tworzonymi z myślą o urządzeniach przenośnych jest sposób wykorzystania nagrań wideo. Witryna dla telefonu komórkowego będzie wyświetlać plik wideo w mniejszym oknie, odczytując mniejszy plik. Powód takiego zachowania powinien być jasny — telefony komórkowe mają zarówno wolniejsze i bardziej kosztowne połączenie z siecią, jak i znacznie mniejszą moc obliczeniową potrzebną do odtworzenia nagrania.

Korzystając z omówionych wcześniej technik zapytań medialnych, możesz łatwo zmienić wielkość elementu `<video>`, tak by dopasował się do mobilnego ekranu. Podanie odrębnego linku do „odchudzonej” wersji nagrania wideo stanowi o wiele większy problem.

HTML5 podaje rozwiązanie: dołącza atrybut `media` bezpośrednio do elementu `<source>`. Jak dowiedziałeś się w rozdziale 5., element `<source>` służy do zadeklarowania pliku multimediów, który ma zostać odtworzony przez komponent `<video>`. Dodając atrybut `media`, możesz przypisać określony typ pliku do rodzaju urządzenia.

W niżej przedstawionym przykładzie plik *butterfly_mobile.mp4* jest przekazywany sprzętom wyposażonym w mały ekran. Inne urządzenia pobierają plik *butterfly.mp4* lub *butterfly.ogv*, w zależności od obsługiwanego formatu wideo.

```
<video controls width="400" height="300">
  <source src="butterfly_mobile.mp4"
    type="video/mp4"
    media="(max-device-width: 480px)">
  <source src="butterfly.mp4"
    type="video/mp4">
  <source src="butterfly.ogv"
    type="video/ogg">
</video>
```

Oczywiście, do Ciebie należy zakodowanie oddzielnej kopii nagrania dla użytkowników telefonów. Narzędzia do zapisywania multimediów są na ogół wyposażone w profile, które ułatwiają cały proces. Mogą np. zawierać opcje kodowania filmu w profilu „iPad video”. W Twojej gestii leży też upewnienie się, że używasz formatu plików właściwego dla danego urządzenia (zwykle jest to format H.264), oraz przygotowanie kilku wariantów pliku dla każdej przeglądarki.

Skorowidz

- \$_GET, 368
- \$_POST, 369
- .appcache, 349
- .NET, 387
- @font-face, 214
- @media, 238, 245
- <a>, 47
- <address>, 46
- <article>, 66, 87, 94
- <aside>, 70, 76, 83, 87, 94
- <audio>, 159
 - obsługa, 169
- , 46, 107
- <body>, 33, 39
-
, 38
- <button>, 329
- <canvas>, 250, 275
 - height, 277
 - width, 277
- <cite>, 46
- <command>, 149
- <content>, 66
- <datalist>, 144
- , 98
- <details>, 80
- <div>, 59, 63, 65, 425
- , 46
- <embed>, 47, 157
- <fieldset>, 122
- <figcaption>, 68, 94
- <figure>, 67, 94
- <footer>, 60, 66, 82, 94
- <form>, 97, 121
- <h1>, 75, 77
- <h2>, 75
- <head>, 33, 39
- <header>, 66, 74, 94
- <hr>, 38, 45
- <html>, 33, 39
- <i>, 46
- <iframe>, 44, 152
- , 38, 276
- <input>, 123, 137, 402
 - kompatybilność przeglądark, 139
 - multiple, 329
 - pobieranie pliku, 326
 - ukryty, 326
- <ins>, 98
- <legend>, 122
- , 78
- <link>, 36, 422
- <main>, 83, 94
- <mark>, 98
- <menu>, 149
- <meta>, 236
- <meter>, 146
- <nav>, 60, 76, 87, 94
- <nobr>, 47
- <object>, 173
- , 47
- <option>, 145
- <output>, 96
- <progress>, 146
- <s>, 45
- <script>, 436, 440
- <section>, 78, 79, 87, 94
- <select>, 123
- <small>, 45
- <source>, 169
 - media, 248
- , 63, 96, 107, 426
- , 46
- <summary>, 80
- <textarea>, 123
- <time>, 60, 95
 - pubdate, 96
- <title>, 33
- <track>, 184, 185
- , 78
- <video>, 49, 159, 161, 279
 - obsługa, 169
- <wbr>, 47
- 3D Walker, 309

A

- accuracy, 395
- action, 121
- adaptacyjny streaming
 - wideo, 159
- adresy
 - danych, 332
 - e-mail, 139
 - IP, 390
 - URL, 104, 140, 269
 - część fragmentaryczna, 412
- affiliation, 108
- agregacja, 90
- Ajax, 366
- akceleracja sprzętowa, 300
- all, 206
- alt, 181
- alternatywna treść, 170, 171, 273
- animowanie
 - ikony, 305
 - płótna, 297
 - wielu obiektów, 299
- aplikacja graficzna, 264
 - malowanie po płótnie, 267
 - przygotowanie narzędzi, 265

- aplikacja graficzna
 - zachowywanie plótina, 268
 - aplikacje offline, 345, 347
 - cache'owanie plików, 346
 - manifest, 347
 - nie działa poza siecią, 352
 - obsługa w przeglądarkach, 355
 - problemy, 352
 - samowystarczalne, 334
 - aplikacje sieciowe, bogate, 249
 - applicationCache, 361
 - status, 361
 - arc(), 256
 - argumenty, 449
 - ARIA, 100
 - aria-invalid, 100
 - aria-required, 100
 - arkusze stylów, 36, 421
 - budowa, 422
 - komentarze, 425
 - zaawansowane, 425
 - zastępowanie całego arkusza, 246
 - at, 203
 - atrybuty, 38
 - cudzysłów, 39
 - zdarzenia, 454, 455
 - zmiennych, 441
 - Audacity, 170
 - audio, 159
 - dynamicznie tworzone i modyfikowane, 159
 - obsługa, 167
 - autocapitalize, 136
 - autocomplete, 136
 - autocorrect, 136
 - autofocus, 126
 - autoIncrement, 338
 - automatyczne przywracanie połączenia, 381
 - autoplay, 161
- B**
- background, 230
 - background-color, 432
 - background-image, 198
 - background-position, 198
 - background-repeat, 198, 331
 - background-size, 331
 - baner, 75
- C**
- bazy danych, 333
 - lokalne, 333
 - na serwerach i u klienta, 333
 - nazywanie, 336
 - przeglądanie wszystkich zapisów, 340
 - przeszukiwanie pojedynczego zapisu, 342
 - sprawdzanie wersji, 337
 - tradycyjne programowanie, 337
 - tworzenie, 336
 - tworzenie zapisów, 338
 - usuwanie zapisu, 343
 - байд praktyczny, 31
 - bevel, 255
 - biblioteka
 - graficzna, 260
 - modułarna, 136
 - prefix-free, 195
 - binary large object, 328
 - blob, 328
 - body, 431
 - bookmarklet, 85
 - border, 432
 - border-box, 232
 - border-colors, 190
 - border-radius, 189, 196
 - border-thickness, 190
 - box-shadow, 199, 206
 - box-sizing, 232
 - brak wsparcia przeglądarek, 158
 - brukuj ścieżki, 30
 - butt, 253
- D**
- dane
 - długowieczne, 314
 - duży blok, 328
 - krótkotrwałe, 314
 - magazynowanie, 315, 334
 - procedury konwersji, 321
 - typy, 444
 - usuwanie, 319
 - w adresie URL, 268
 - dane semantyczne
 - ekstrakcja w przeglądarce, 109
 - ignorowane przez Google, 114
 - data, 95, 409
 - formaty, 143
 - database, 336
 - data-url, 342
 - datetime, 95
 - debugowanie, 445
 - descriptions, 184
 - designMode, 150, 152
 - destination, 263
 - cienie, 199
 - dodawanie, 280
 - inne ozdobniki, 280
 - Cisco, 157
 - class, 423
 - className, 455
 - code, 395
 - color, 239, 423, 432
 - column-count, 224
 - column-rule, 224
 - column-span, 224
 - all, 224
 - column-width, 224
 - Content, 243
 - contentEditable, 150
 - controls, 159, 162
 - cookies, 313, 319
 - coords, 395
 - CSS, 36, 187, 421
 - podstawy, 421
 - CSS3, 187
 - cubic-bezier, 210
 - currentTime, 179
 - cursor.value, 341
 - cytaty, 70
 - czas, 95, 142
 - serwera, 377
 - czyszczenie pamięci podręcznej, 354

device-aspect-ratio, 239
device-height, 239
device-width, 239
display, 244
 block, 72
 none, 75, 107
doctype, 34
document, 453
Document Object Model, 31
dokument HTML5, zmiana
 na XHTML5, 42
dostępność, 60, 181
 płótno, 274
drobny druk, 45
dropBox, 330
durationBar, 179
dziedziczenie, 426

E

ease-in, 210
ease-in-out, 210
ease-out, 210
echo, 369
edytory
 audio, 170
 HTML na stronie, 149
edytowanie HTML-u, 150,
 152
efekty
 dźwiękowe, 174
 przejścia, 204
eksploracja danych, 99
elementy
 blokowe, 426
 puste, 38
 sekcji, 87
 standaryzowane, 47
 warstwy prezentacji, 43
elementy semantyczne, 60,
 65, 94
 dodawanie do strony, 73
 kompatybilność
 z przeglądarkami, 71
 prawidłowe
 wykorzystanie, 83
 służące do tworzenia
 struktury strony, 94
 stylizacja, 72
 zastępowanie
 komponentu <div>,
 426
elipsa, 63, 257
else, 447
em, 224, 233, 235

em dash, 233
enableHighAccuracy, 396
EOT, 214, 215
event, 456
Event-Machine, 387
ExplorerCanvas, 271

F

Fabric.js, 260
FALLBACK, 357
figury, 254
File API, 325, 355
 obsługa przeglądarek,
 332
filename, 407
fillStyle, 254, 282
Firefogg, 170
firety, 233, 235
Flash, 32, 158, 170
 z mechanizmem
 awaryjnego ładowania
 filmu, 173
FlashCanvas, 272
FlashCanvas Pro, 273
float, 243
Flowplayer, 171
Flowplayer HTML5, 174
focus, 125, 126
Font Squirrel, 217
font-family, 430
fonty
 awaryjne, 430
 formaty, 214
 wielkość, 432
fonty dla witryny, 216
 konwersja na font
 sieciowy, 216
 pobieranie z Font
 Squirrel, 217
 przygotowanie, 218
 zasady licencjonowania,
 219
fonty sieciowe, 188
 bezpieczne, 430
 Google, 221
for, 447
formatowanie, 45
 elementów przy użyciu
 klas, 423
 komendy, 151
 kontrolki walidacyjnych,
 130
 nagłówków, 64, 66
 panelu bocznego, 79

 pojedynczego
 komponentu, 428
 rysunków, 69
 stopki, 82
 zawartości dokumentu,
 63
 znaków wodnych, 125
formnovalidate, 129
formularze
 nowe elementy, 144
 sieciowe, 120
 uzupełnianie kodu, 135
 wykrywanie błędów, 127
formularze HTML, 119
 modernizacja, 121
 ograniczenia, 124
from, 201, 203
funkcje, 437
 addLink(), 339
 addValue(), 324
 alert(), 436
 askServer(), 369
 calc(), 232
 canvasClick(), 295
 changeColor(), 266
 changeThickness(), 266
 checkForCollision(),
 306, 308
 clearCanvas(), 268
 clearInterval(), 297
 connect(), 386
 definiowania obiektu,
 450
 draw(), 267
 drawFrame(), 298, 300
 drawImage(), 277
 drop(), 331
 flush(), 378
 goToNewSlide(), 373,
 415
 goToNextSlide(), 417
 gradient(), 200
 ignoreDrag(), 331
 importScripts(), 406
 linear-gradient(), 201
 matrix(), 211
 nazwy, 441
 nextSlide(), 373
 Number(), 319, 321
 obsługujące, 441
 open(), 336
 otrzymywanie danych,
 449
 plotScore(), 289
 previousSlide(), 373

- funkcje
- processFiles(), 326, 331
 - processKey(), 305
 - radial-gradient(), 202, 203
 - repeating-linear-gradient(), 203
 - repeating-radial-gradient(), 203
 - rgb(), 253, 432
 - rgba(), 195, 196, 262
 - rotate(), 210, 211
 - scale(), 211
 - scaleX(), 211
 - scaleY(), 211
 - setInterval(), 297, 375
 - setTimeout(), 297, 375
 - showLinks(), 336
 - showMessage(), 441
 - skew(), 211
 - skewX(), 211
 - skewY(), 211
 - sleep(), 378
 - startDrawing(), 267
 - stopDrawing(), 267
 - storageChanged(), 325
 - time(), 378
 - transaction.objectStore(), 340
 - translate(), 211
 - translateX(), 211
 - translateY(), 211
 - webkit-linear-gradient(), 201
 - wpłatane, 457
 - zwracanie danych, 449
 - zwrotne, 393
- G**
- geolokalizacja, 389, 390
- adres IP, 390
 - Bluetooth, 392
 - działanie, 390
 - generowanie mapy, 397
 - GPS, 391
 - Manual Geolocation, 392
 - monitorowanie ruchu użytkownika, 399
 - obsługa
 - w przeglądarkach, 400
 - odnajdywanie współrzędnych użytkownika, 392
 - określanie dokładności współrzędnych, 395
 - plik cookie, 392
 - RFID, 392
 - smartfon, 391
 - ustawienia, 396
 - usuwanie błędów, 394
 - wykorzystanie, 392
- GET, 370
- GitHub, 272
- globalAlpha, 263
- GlobalStats, 51
- główna treść, 83
- Goldwave, 170
- Google Analytics, 53
- Google Location Services, 390
- Google Web Fonts, 221
- GPS, 391
- gradienty, 200
- liniowy, 200, 201
 - powtarzalny, 203
 - promienisty, 200, 202
- grafika, 78
- dołączanie, 67
 - zapisywanie, 270
- grupy treści, 80
- H**
- H.264, 163, 165, 167
- licencjonowanie formatu, 168
- h5o, 85
- HandBrake, 170
- handlery zdarzeń, 336, 341, 441
- hCalendar, 102
- hCard, 101
- height, 162, 239, 250
- historia sesji, 389, 411
- dawne rozwiązanie, 412
 - kompatybilność, 417
 - kwestia URL, 411
 - rozwiązanie HTML5, 413
 - znaki #, 413
- history, 411, 414
- HTML, 29
- HTML 5.1, 19
- HTML5, 19, 27, 187
- elementy
 - standaryzowane, 47
 - elementy zaadaptowane, 45
 - historia, 25
 - kategorie właściwości, 28
- komponenty usunięte ze specyfikacji, 43
- nowe znaczniki, 44
- pryncypia, 29
- rozluźnione reguły, 38
- składnia, 32, 38
- zawartość, 28
- zmodyfikowane znaczniki, 46
- HTML5 Outliner, 85
- HTML5 Shiv, 72
- HTML5Forms, 135, 142
- I**
- id, 250, 376, 428
- identyfikatory, 428
- numeryczne, 338
- if, 447
- indeksery, 316
- indeksy, 333
- IndexedDB, 333, 355
- łączenie z bazą danych, 336
 - manipulacja danymi, 338
 - obsługa, 344
 - przechowywanie baz danych, 338
 - przechowywanie obiektów, 335
 - zamknięcie tablic, 339
 - zastosowanie, 333
- Infinity, 397
- InfoWindow, 399
- innerHTML, 455
- inset, 200, 206
- instrukcja timera, 297
- instrukcje warunkowe, 446
- interaktywne figury, 291
- Internet, 157
- iPaint, 271
- isDrawing, 267
- ISO 8859-2, 35
- itemprop, 103, 107
- itemReviewed, 113, 114
- itemscope, 103, 107
- itemtype, 103, 107
- J**
- JavaScript, 36, 435
- analityczny składni XML, 328
 - atrybut id, 428

- dynamiczne łączenie ze zdarzeniem, 454
 - identyfikowanie błędów w kodzie, 445
 - interakcja ze stroną, 452
 - Internet Explorer, 437
 - korzystanie, 436
 - manipulowanie elementem, 453
 - obchodzenie mechanizmu przesyłania danych, 121
 - obliczenia, 96, 121
 - odpowiadanie na zdarzenia, 440
 - przeniesienie kodu do oddzielnego pliku, 439
 - składnia, 441
 - sterowanie odtwarzaniem, 174
 - używanie funkcji, 437
 - w przeglądarce, 436
 - wykrywanie uaktualnienia, 360
 - zagnieżdżanie kodu, 436
 - zastępczy kod, 49
 - język, 35
 - jPlayer, 180
 - jQuery, 374, 455
 - JSON, 323
- K**
- Kaazing, 387
 - kanały, 110
 - alfa, 195, 262
 - karetka, 150
 - kind, 184
 - KineticJS, 260
 - klasy, 450
 - applicationCache, 360
 - ArticleTitle, 425
 - Content, 433
 - FileReader, 327, 332
 - LeadIn, 433
 - mikroformatu, 102
 - klient, 313
 - WebSocket, 384
 - klucze, 315
 - główne, 338
 - nazwy, 318
 - kod
 - języka, 184
 - po stronie serwera, 365
 - przeniesienie z części głównej dokumentu, 439
 - semantyczny, standardy, 99
 - w oddzielnym pliku, 439
 - wielokrotne użycie, 439
 - zagnieżdżanie, 436
 - zestaw funkcji, 438
 - kodeki
 - audio, 164
 - profilu podstawowego, 167
 - wideo, 164
 - kodowanie
 - base-64, 269
 - mediów, 170
 - znaków, 35
 - kolorы, 144, 253
 - kolumny, 228, 426
 - komentarz
 - JavaScript, 437
 - w arkuszach stylów, 425
 - kompatybilność, 50
 - wsteczna, 31, 56
 - komponenty sekcji, 87
 - kompozycje, 263
 - złożone, 263
 - kommunikacja z serwerem, 365
 - historia rozwoju, 366
 - operacje w tle, 385
 - utrata łączności, 376
 - konflikty nazw, 318
 - konспект strony, 85
 - podstawowy, 86
 - problemy z tworzeniem, 89
 - włączenie, 85
 - kontekst graficzny, dwuwymiarowy, 251
 - kontener, 194, 426
 - multimedialny, 164, 177
 - kontrolki, 97, 121, 123
 - adresy e-mail, 139
 - adresy URL, 140
 - aktywowanie, 126
 - atrybuty, 136
 - dat, 142
 - kolorów, 144
 - liczb, 140
 - ładowania plików, zastępowanie, 328
 - poza obrębem formularza, 122
 - telefonów, 140
 - walidacyjne, 130
 - konwersja
 - daty na tekst, 321
 - liczb na tekst, 321
 - własnych obiektów na tekst, 322
 - korzenie sekcji, 89
 - krzywa Béziera, 257
 - krzywe, 256
 - kumulacja, 235
 - kursor bazy danych, 340
 - kursywa, 45
 - kwerenda, 340

L

 - label, 184
 - labirynt, 303
 - rysowanie, 304
 - zapis stanu gry, 317
 - lang, 36
 - LatLng, 399
 - layout, 227
 - dostosowywanie, 237
 - firety, 234
 - modyfikacja struktury, 238
 - obsługa na smartfonach, 236
 - ograniczenie rozciągania, 237
 - płynny, 228, 229
 - proporcjonalność na wielu poziomach, 235
 - proporcjonalny, 228
 - przyjazny dla urządzeń mobilnych, 241
 - skalowany, 231
 - left, 201
 - length, 448
 - licencjonowana treść, 159
 - liczby, 140
 - linear, 210
 - lineCap, 253
 - line-height, 433
 - lineJoin, 255
 - lineno, 407
 - lineWidth, 253, 266, 280
 - linie proste, 252
 - linki, 78, 338
 - lista, 123
 - dostępnych opcji, 144
 - kroków, 268
 - rozwijana, 244
 - literały obiektu, 451

local(), 220
localStorage, 314, 317, 321
logika kodu, dodatkowa, 327
lokalizowanie trafień, 291
 a barwa pikseli, 307
 współrzędne, 294
long-lived, 314
loop, 161

Ł

łańcuch
 zapytań, 368
 znaków, 442
łatwość
 edytowania, 60
 utrzymania, 60
łączenie dokumentów, 90
łuki, 257

M

magazyn danych, 317
magazyn lokalny, 314
 pobieranie wpisów, 320
 przeglądarki, 317
 przekazywanie danych,
 322
 tworzenie elementów,
 315
 usprawnianie, 334
magazyn sesji, 314
 tworzenie elementów,
 316
magazyn sieciowy, 314, 319
 listowanie wszystkich
 zachowanych wpisów,
 320
 obsługa przeglądarek, 319
 podstawy, 314
 problemy, 317
 reagowanie na zmiany,
 323
 składnie, 316
 usuwanie wpisów, 319
 zachowywanie
 obiektów, 322
 zapisywanie liczb i dat,
 321
malowanie, 267
małe bloki treści, 80
manifest, 346
 komentarze, 348
 korzystanie, 349
 przenoszenie na serwer,
 349
 tryb awaryjny, 357
 tworzenie, 347
 uaktualnianie, 352, 353
manipulacja danymi, 338
mapa wiedzy wikipedii, 309
mapy obrazów, 274
margin, 207, 432
marginesy, 229
markupToInsert, 341
matrix, 260
max-color, 239
max-device-aspect-ratio, 239
max-device-height, 239
max-device-width, 239,
 240, 243, 246
max-height, 239
maximumAge, 396
max-width, 237, 239, 240
media, 246
 handheld, 239
mediagroup, 163
menu, 149
 rozwijalne, 80
message, 395
messageLog, 379
messageType, 408, 409
meta, 35
metadane, wykorzystanie
 przez wyszukiwarki, 110
metadata, 184
metody, 453, 454
 abort(), 332
 addColorStop(), 285, 286
 addEventListener(), 324
 asynchroniczne, 327
 back(), 411
 beginPath(), 254
 boing(), 176
 canPlayType(), 177
 clear(), 319
 clearRect(), 268, 294
 clearWatch(), 392, 400
 click(), 326, 329
 close(), 380, 384, 407
 closePath(), 254
 createImageData(), 276,
 307
 createLinearGradient(),
 283
 createObjectStore(), 337
 createPattern(), 283
 createRadialGradient(),
 283
 cursor.continue(), 342
 database.transaction(),
 339
 delete(), 339, 343
 document.getElementById
 ↳ ById(), 250, 453
 doSearch(), 405
 drawImage(), 276
 fill(), 254, 291
 fillEllipse(), 260
 fillRect(), 256
 fillStroke(), 262
 fillStyle(), 262
 fillText(), 279, 280
 focus(), 126
 forward(), 411
 get(), 342
 getContext(), 251
 getCurrentLocation(),
 394
 getCurrentPosition(),
 392, 393, 400
 getImageData(), 268,
 307
 getItem(), 316
 go(), 411
 jQuery.ajax(), 374
 JSON.parse(), 323
 JSON.stringify(), 323
 key(), 320
 lineTo(), 252
 localStorage.setItem(),
 315
 measureText(), 279
 Modernizr.mq(), 244
 moveTo(), 252
 open(), 370
 openCursor(), 341
 pause(), 175
 play(), 175
 postMessage(), 404
 push(), 448
 pushState(), 411, 414
 put(), 339, 340
 putImageData(), 307
 readAsArrayBuffer(), 328
 readAsBinaryString(),
 328
 readAsDataURL(), 328,
 329, 331
 readAsText(), 327
 removeItem(), 319
 replaceState(), 416
 restore(), 261
 save(), 261

send(), 370, 384
 setCenter(), 399
 setCustomValidity(), 132
 setInterval(), 401
 setItem(), 316
 setTimeout(), 299, 401
 showFileInput(), 329
 składu obiektów, 339
 startEdit(), 153
 stopEdit(), 153
 stroke(), 252, 254, 291
 strokeRect(), 256
 strokeText(), 280
 swapCache(), 363
 terminate(), 407
 toDataURL(), 268
 update(), 363
 validateComments(), 132
 watchPosition(), 392, 400
 window.indexedDB.
 ↪deleteDatabase(), 338
 window.location.reload(), 362
 miarka, 148
 Microdata Tool, 109
 miernik, 146
 mikrodane, 95, 102
 a mikroformaty, 103
 generowanie kodu, 109
 ignorowane przez Google, 114
 nowa sekcja, 108
 mikroformaty, 101
 MIME, 350
 min-color, 239
 min-device-aspect-ratio, 239
 min-device-height, 239
 min-device-width, 239
 min-height, 239
 min-width, 237, 239
 Miro Video Converter, 170
 mitre, 255
 mobile-first, 243
 model asynchroniczny, 334
 modernizacja strony
 formularze HTML, 121
 tradycyjnej, 61
 uzupełnienie danymi semantycznymi, 105
 Modernizr, 53, 190, 244
 dodawanie awaryjnych mechanizmów, 190
 działanie własności rysowania, 273
 testowanie obsługi mechanizmu walidacji, 134
 znaczniki semantyczne, 73
 moduły CSS, 187
 MP3, 163, 165, 167
 multimedia
 formaty, 163
 grupy, 163
 kodowanie, 170
 obsługa w przeglądarkach, 165
 odtwarzanie, 159
 wczytywanie po załadowaniu strony, 160
 zamiana formatów, 168
 multiple, 136, 139, 327, 329
 muted, 162

N

nagłówek kontroli pamięci podręcznej, 347
 nagłówki, 74, 77, 432
 nagrywanie dźwięku i obrazu, 159
 nakładanie stylów, 72
 name, 327
 napisy, 181
 narzędzia do rysowania, 260
 navigator.onLine, 359
 NavSidebar, 243
 NETWORK, 358
 new, 292
 nie psuj sieci, 30
 no-cache, 350
 node.JS, 387
 not, 245
 novalidate, 129
 null, 443
 numery telefonów, 140

O

obiekty, 450
 EventSource, 379
 FileReader, 329
 LinkRecord, 335
 literały, 451
 navigator.geolocation, 392
 przechowujące dane, 335
 szablony, 450
 WebSocket, 384
 window.indexedDB, 336
 XMLHttpRequest, 121, 366, 369, 382, 415
 obliczenia, 96, 121, 444
 możliwe tylko na serwerze, 367
 obramowania, 231
 obrazy
 bitmapowe, 260
 płynne, 231
 przenoszenie, 332
 wycinanie i zmienianie wielkości, 277
 obsługa
 błędów, 31
 canvas, 271
 File API, 332
 HTML5, 15, 17, 48, 50
 IndexedDB, 344
 layoutu, 236
 magazynu sieciowego, 319
 mechanizmu walidacji, 133
 Modernizr, 138
 multimediów, 165, 169
 obrazów, 330
 podpisów, 185
 różnych formatów, 169
 WebSocket, 382
 własności, 53
 własności serwerowych zdarzeń, 375
 wykrywanie, 273
 zdarzeń po stronie serwera, 379
 odczytywanie plików, 325
 graficznych, 329
 tekstowych, 326
 wielu, jednocześnie, 329
 odnośniki, 47, 76
 odstępy, 234
 odświeżenie, pełne, 373
 odtwarzacz
 filmów, 177
 Flash, 171
 JavaScript, 180
 odtwarzanie audio, 159, 175
 dynamiczne, 161
 zapętlone, 161
 Ogg Theora, 165, 167

- Ogg Vorbis, 165, 167
 - okrąg, 257
 - onclick, 266
 - opacity, 196, 206
 - operacje, 444
 - operatory
 - arytmetyczne, 444
 - dodawania, 444
 - identyczności, 447
 - logiczne, 446
 - options, 396
 - optymalizacja
 - wyszukiwarek, 61, 110
 - orientation, 239, 240
 - OTF, 214
 - overflow, 403
 - overflow-x, 403
- P**
- padding, 207, 432, 433
 - pamięć podręczna, obejmście
 - w trybie online, 359
 - panel boczny, 76
 - parametry, 449
 - parentElement, 455
 - pasek narzędzi, 149
 - pasek postępu, 179
 - nieoznaczony, 148
 - pasek stanu, 146, 179
 - pattern, 132, 134
 - PersonalityScore, 322
 - pętle, 447
 - PHP, 368
 - tworzenie skryptu, 368
 - WebSocket, 387
 - PHP_EOL, 378
 - piaskownica, 452
 - pisanie kodu, 17
 - placeholder, 125
 - playbackRate, 178
 - pliki
 - manifestu, 346
 - ścieżek, 185
 - plótno, 249
 - alternatywne, 273
 - dostępność, 274
 - kompatybilność
 - z przeglądawkami, 271
 - praktyczne
 - wykorzystanie, 309
 - wprowadzenie, 250
 - współrzędne, 252
 - zachowywanie, 268
 - pobieranie
 - danych z serwera, 367, 452
 - nowych treści, 371
 - pliku, 326
 - podpisy, 181, 185
 - dodawanie, 184
 - pogrubienie, 45
 - poła
 - formularza, 121, 125
 - tekstowe, 123
 - wyboru, 123
 - wymagane, 128
 - wyszukiwania, 140
 - polling, 375
 - a zdarzenia po stronie
 - serwera, 380
 - polyfill, 56
 - pomocnicza sekcja zapytań, 240
 - positionBar, 179
 - POST, 370
 - poster, 162
 - potokowanie, 381
 - pozycjonowanie
 - stałe, 81
 - stron, 110
 - półokrąg, 256
 - pracownicy, 389, 401
 - anulowanie zadania
 - uruchomionego w tle, 407
 - bezpieczeństwo, 402
 - czasochłonne zadanie, 401
 - inne zastosowania, 409
 - obsługa błędów, 407
 - obsługa w
 - przeglądarkach, 410
 - przekazywanie bardziej
 - złożonych
 - wiadomości, 407
 - przykłady wykorzystania, 409
 - wykonywanie zadań
 - w tle, 404
 - wykorzystanie offline, 406
 - preferencje aplikacji, 318
 - prefix-free, 195
 - preload, 160
 - programy graficzne, 271
 - projektowanie elastyczne, 228
 - projektowanie stron, 187
 - HTML a CSS, 422
 - mobilna wersja, 242
 - technologie, 76
 - projektowanie
 - ukierunkowane na
 - urządzenia przenośne, 243
 - przeciągnij-i-upuść, 326, 330
 - przedrostek autorski, 192
 - automatyczne
 - dodawanie, 195
 - moz-, 193
 - ms-, 193
 - o-, 193
 - webkit-, 193
 - przegląd tablicy, 340
 - przeglądarki, 48
 - dostawcy lokalizacji, 390, 391
 - ekstrakcja danych
 - semantycznych, 109
 - statystyki poziomu
 - przyjęcia, 51
 - wsparcie obsługi
 - HTML5, 50
 - przejścia, 206
 - cień, 206
 - gradienty, 206
 - płynne, 374
 - przezroczystość, 206
 - transformaty, 207
 - wykorzystujące
 - transformaty, 212
 - wywoływanie
 - w JavaScriptcie, 207
 - przekształcanie koloru, 204
 - przeplływowość, 169
 - przestrzenie nazw
 - Organization, 108
 - Person, 105, 108
 - XML, 103, 104
 - przesyłanie danych, 121
 - na serwer, 369
 - w formacie JSON, 376
 - prześwitujące tło, 81
 - przezroczystość, 188, 195, 262
 - przyciski
 - formularza, 123
 - przesyłania omijające
 - walidację, 129
 - radiowe, 123
 - wyboru napisów, 185
 - zamykania, 81
 - pseudoklasy, 204, 428
 - active, 429
 - focus, 130, 204

hover, 194, 204, 429
in-range, 130
invalid, 130
link, 428
optional, 130
out-of-range, 130
required, 130
valid, 130
visited, 428
punkty kontrolne, 257
Python, 387

R

radial-gradient, 192, 193
ramki, 44
 redaktorskie, 70
RDFa, 100
rdzeń, 28
readonly, 340
readwrite, 340
readyState, 370, 386
Recipe View, 115
reguły, 422
rejestr linków, 334
rem, 235
repeat, 283
repeat-x, 283
repeat-y, 283
Request.QueryString, 369
request.result, 337
required, 127
responseText, 371
responsive design, 228
retry, 376
return, 449
reversed, 47
Rich Snippets Testing Tool,
 111
roboty, 61
role, 100
root em, 235
rotate, 260
round, 253, 255
rozchodzenie się cienia, 200
Ruby, 387
rysowanie, 249
 klatki filmowej, 279
 kompozycje złożone, 263
 na płótnie dla
 matematycznych
 alergików, 260
 obrazów, 276
 rozpoczęcie nowego
 segmentu, 254

tekstu, 278
wykresów, 286
rysunek, 67

S

samoistne treści, 80
sandbox, 452
sans-serif, 430
scale, 260
seekable, 179
sekcje, 79
 FALLBACK, 357
 NETWORK, 358, 359
 SETTINGS, 359
 strony, 76
 ukrywanie, 244
 zapytań medialnych, 245
 zastępowanie, 244
selektory, 422, 425
 .FatFooter, 83
 .FloatFigure, 68
 atrybutów, 429
 identyfikatora, 428
 klasy, 427
 kontekstowe, 427
 pseudoklas, 428
 wiele, 426
Semantic Inspector, 109
semantyka, 93
 tekstu, 94
SEO, 110
serif, 430
serwery
 dedykowany, 383
 proxy, 378, 385
 sieciowy, 313, 317
 czas, 377
 WebSocketów, 383, 387
 wywoływanie, 369
sessionStorage, 314, 316,
 317, 321
SETTINGS, 359
sformatowane fragmenty,
 110
shadowBlur, 281
shadowColor, 281
shadowOffsetX, 281
shadowOffsetY, 281
short-lived, 314
SIL Open Font, 218
Silverlight, 272
size, 327
skalowanie automatyczne,
 236

Sketchpad, 271
skład obiektów, 339
 tworzenie, 340
skrypt serwera, 377
source, 263
source-atop, 263
source-over, 263
spellcheck, 136
sprawdzanie
 obsługi, 53
 stanu połączenia, 359
square, 253
src, 454
srcLang, 184
stan aplikacji, 313, 318
 zapisywanie, 317
standard
 audio i video, 164
 HTML5, 18
status, 370
step, 141
sterowanie odtwarzaniem,
 174
stopień zamazania, 199
stopka, 81
 rozwijana, 81
stosunek pikseli, 247
strefa czasowa, 96
strokeStyle, 253, 280, 282
Structured Data Testing
 Tool, 111
struktura strony
 klasyczna, 62
 konstruowanie przy
 użyciu elementu
 <div>, 425
 nagłówki, 77
 w HTML5, 65
struktura zagnieżdżona, 107
style, 454, 455
 blokowy, 72
 warunkowe, 238
 załączanie do stron, 421
subtitles, 184
sugerowane odpowiedzi, 144
suwaki, 141
SVG, 215, 272
symulatory wyglądu
 strony, 236
syndykalizacja, 90
system tworzenia
 konspektów, 91
szablony, 450
 stron, 76
szpachlówka, 56

Ś

- ścieżki, 254
 - kluczowe, 337
 - napisów, 181
- śledzenie rysowanych elementów, 291
- średnik, 436
- środowiska działające po stronie serwera, 76

T

- tablica, 448
 - przeglądanie zapisów, 340
 - przeszukiwanie pojedynczego zapisu, 342
 - tworzenie, 337
 - usunięcie zapisu, 343
- TagName, 455
- taktowne degradowanie strony, 49
- technika poślizgowych drzwi, 198
- tekst
 - emfaza, 46
 - płynny, 233
 - rozdzielanie wyrazu, 47
 - ważny, 46
 - zaznaczanie, 98
- testowanie wsparcia dla własności, 55
- text-align, 423
- text-shadow, 199, 200
- this, 292, 451
- timeDisplay, 379
- timeout, 396
- timestamp, 395
- title, 126
- tło, 197
- to, 201, 203
- top, 201
- transakcje, 338
 - rodzaj, 340
 - tworzenie, 339
- transform, 209
- transformacja strony w witrynę, 76
- transformaty, 209, 258
 - macierzy, 260
 - obrotów, 260
 - skali, 260
 - tłumaczeniowe, 260

- transform-origin, 212
- transition, 205, 210
- translate, 260
- treści awaryjne, 168
- tryby
 - dziwactw, 34
 - offline, 345
 - standardów, 34
 - XHTML, 43
- TTF, 214, 215
- TTML, 183
- tworzenie kolekcji fontów, 223
- type, 327
 - color, 144
 - date, 142, 143
 - datetime, 143
 - datetime-local, 143
 - email, 137, 139
 - file, 326
 - month, 143
 - number, 140
 - range, 141
 - search, 140
 - tel, 140
 - text, 429
 - time, 143
 - url, 140
 - week, 143
- typografia
 - płynna, 233
 - w sieci, 213
- typy
 - danych, 444
 - mediów, 239
 - MIME, 165, 166
 - treści, 166
 - tytuły sekcji, 78

U

- uaktualnianie strony, 452
- układ strony
 - dwukolumnowy, 228
 - proporcjonalny, 229
- układ współrzędnych domyślny, 261
- urządzenia mobilne rozpoznawanie, 246
 - standardowe szerokości, 247
- user_name, 315
- ustawienia użytkownika, 313
- UTF-8, 35
- uzupełnianie braków, 55

V

- var, 441, 443
- vcard, 101
- vevent, 102
- VideoJS, 180
- viewport, 236
- VML, 271

W

- W3C, 16, 25
- WAI, 61
- WAI-ARIA, 100
- walidacja, 39, 127
 - obsługa mechanizmu, 133
 - po stronie klienta, 128
 - po stronie serwera, 128
 - proces w HTML5, 127
 - pustych pól, 132
 - własne reguły, 132
 - wyłączenie, 129
 - wyrażeń regularnych, 131
- walidator
 - W3C, 40
 - XHTML5, 42
- wartości, 423
 - barw, 432
 - dziedziczone, 426
 - stopujące, 201
- warunek, 446
- WAV, 165, 167
- Web Applications 1.0, 28
- Web Forms 2.0, 28
- web storage, 314
- WebM, 165, 167
- WebSocket, 365, 382
 - klient, 384
 - serwery, 383, 387
 - w sieci, 385
 - wykonywane zadania, 383
- WebVTT, 181
- WHATWG, 28
- white-space, 47
- wiadomości, 376
 - przetwarzanie na stronie, 379
 - wysyłanie, 377
- wideo, 157
 - na urządzeniach przenośnych, 248
 - obsługa, 167
 - w HTML5, 159
- widok strony, 236

width, 162, 239, 243, 250
wielki obiekt binarny, 328
wielokolumnowy tekst, 223
wielokropek, 63
wizytówka, 101
własności
 wypełnienia, 433
 zaplanowane, 61
własności CSS, 423
 kategorie, 424
własny obiekt, 291
właściwości, 316, 453, 454
 mediów, 238
 limity, 240
WOFF, 215
word-wrap, 188
Worker, 404
ws, 384
wskaźnik, 148
współczynnik korekcji, 247
współrzędne, 294
wss, 384
wtyczki, 32, 158, 170
wydajność
 animacji, 300
 zwiększanie, 334
wydawanie
 asynchronicznych żądań,
 336
wykrywanie
 luk w obsłudze, 55
 obsługi własności, 53
wypełnianie, 55
 deseniem, 282
 gradientem, 283
wypełnienia, 49, 55
 ExplorerCanvas, 271
 Flash, 171
 FlashCanvas, 272
 FlashCanvas Pro, 264
 formatu, 168
 HTML5Forms, 142
 technologiczne, 168
wyrażenia regularne, 131
wysyłanie danych na
 serwer, 366, 453
wyszukiwanie
 dynamiczne, 121
 lepsze wyniki, 111
 sugerowane hasła, 121
wyszukiwarki
 przepisów, 114
 wykorzystanie
 metadanych, 110
wywołanie funkcji, 438

X

XForms, 119
XHTML, 26, 42
XHTML 2, 27
XHTML5, 42
XML, 328
xor, 263

Y

YouTube, 171

Z

zadania asynchroniczne, 457
zamykający ukośnik, 38
zaokrąglone rogi, 196
zapożyczanie treści, 90
zapytania medialne, 170,
 227, 238
 i wideo, 248
 wiele stylów, 245
zapytania w tle, 370
zapytania wysyłane
 na serwer, 367
 wywoływanie serwera,
 369
zasady dobrego stylu
 w HTML5, 39
zastępczy kod, 49
zdarzenia, 439, 440
 dynamiczne łączenie, 454
 event.oldVersion, 337
 findAllItems(), 320
 nazwy, 456
 onBeforeUnload, 317
 onBlur, 442
 onCached, 361
 onChange, 442
 onChecking, 361
 onClick, 295, 442
 onClose, 384
 onDownloading, 361
 onDragEnter, 331
 onDragOver, 331
 onDrop, 331
 onError, 332, 337, 339,
 361, 384, 407, 442
 onFocus, 442
 onInput, 132
 onKeyDown, 442
 onKeyUp, 442
 onLoad, 276, 282, 327,
 442
 onLoadEnd, 332
 onMessage, 384, 404
 onMouseDown, 267
 onMouseMove, 267, 296
 onMouseOut, 267, 442
 onMouseOver, 440, 442
 onMouseUp, 267
 onNoUpdate, 361
 onObsolete, 361
 onOpen, 384
 onPopState, 414
 onProgress, 179, 332, 361
 onReadyStateChange, 370
 onSelect, 442
 onStorage, 319, 323
 onSuccess, 336, 337, 339
 onUnload, 442
 onUpdateReady, 361
 onUpgradeNeeded, 337
 po stronie serwera, 380
 receiveMessage, 379
 request.onError, 337
 window.onload, 251, 253
 window.onStorage, 323,
 324
 wplatane, 457
zdarzenia przesyłane
 na serwer, 365, 375
 format wiadomości, 376
 przetwarzanie
 wiadomości
 na stronie, 379
 wysyłanie wiadomości,
 377
Zencoder, 170
zerwanie połączenia, 381
zmiennne, 441
 globalne, 443
 lokalne, 443
 nazwy, 442
 operacje na zmiennych,
 444
 zakres, 443
znaczniki
 blokowe, 72
 liniowe, 426
 małe litery, 39
 nowe, 43
 dodawanie do strony,
 73
 przestarzałe, 31
 rozpoznawanie, 72
 usunięte ze specyfikacji,
 43
 zamykający, 32
 zmodyfikowane, 46

znak wodny, 124
 prawidłowy, 126
znaki sieci, 37, 73, 437

ż

żądania asynchroniczne, 367
żywy język, 29

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 na dobre zadomowił się w sieci. Z jego dobrodziejstw garściami czerpią projektanci witryn — i zaskakują użytkowników coraz bardziej pomysłowymi aplikacjami. HTML5 zapewnia zaawansowaną integrację ze środowiskiem przeglądarki internetowej, usługi geolokalizacyjne oraz doskonałe wsparcie dla materiałów multimedialnych czy aplikacji offline. Jeszcze całkiem niedawno nikt nie marzył o rzeczach, na jakie obecnie ten język programowania pozwala twórcom stron!

Książka, którą trzymasz w rękach, należy do cenionej serii „Nieoficjalny podręcznik”. Jej kolejne wydanie zostało poprawione, zaktualizowane i rozszerzone o mnóstwo nowych informacji. Dzięki tej publikacji nie musisz odkrywać tajników HTML5 na własną rękę. Liczne przykłady oraz szczegółowe omówienie najróżniejszych zagadnień sprawiają, że w mig opanujesz niuanse tego języka. W trakcie lektury dowiesz się, jak wykorzystać nowy element canvas i jak bez trudu zlokalizować użytkownika, oraz zobaczysz, jak przechować kluczowe dane w bazie przeglądarki. Sięgnij po tę książkę i przekonaj się, jakie możliwości daje Ci HTML5!

HTML5 to:

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

Poznaj możliwości HTML5 i zacznij korzystać z nich już dziś!

helion.pl
księgarnia internetowa

Nr katalogowy: 23314



Księgarnia internetowa
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900
0 601 339900

O'REILLY



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. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po WIECEJ



KOD KORZYŚCI

ISBN 978-83-246-9251-4



Cena: 77,00 zł

Informatyka w najlepszym wydaniu