

Zawiera
CSS3

CSS

Wydanie III

Witryny internetowe szyte na miarę

Charles Wyke-Smith

AUTORYTETY INFORMATYKI

New
Riders

Helion

Podziękowania

Moje podziękowania otrzymuje zespół Peachpit: Michael Nolan za zachęcenie mnie do napisania trzeciego wydania i dopuszczenie go do produkcji; moja redaktor Nancy Peterson za pokierowanie mną, jej spostrzeżenia i cierpliwość; wydawca Nancy Ruenzel za to, że od siedmiu lat publikuje moje książki.

Z zespołu produkcyjnego podziękowania otrzymuje korektor Darren Meiss za skrupulatne nadzorowanie mojej gramatyki, Katerina Malone za swoje umiejętności zarządzania oraz Karin Arrigoni za szczegółową pracę nad indeksem.

Wielkie podziękowania otrzymuje mój dobry przyjaciel i korektor merytoryczny tej książki Curtis Blanton za przekazywanie mi szczegółowych informacji zwrotnych oraz propozycji co do kodu wraz z objaśnieniami. Świetna robota, Curtis! Programiści Jeffrey Johnson i Isaac Shapira też zapracowali sobie na moją wdzięczność za sprawą swoich porad i wsparcia.

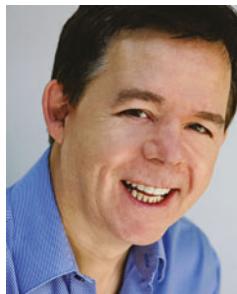
Wreszcie, szczególnie chciałbym podziękować mojej żonie, Beth Bast, która jako redaktor produkcyjny i operator DTP nieugięcie czytała i redagowała moje kolejne szkice, stworzyła oprawę graficzną książki i wykonała jej skład w InDesign. To było pięć miesięcy intensywnej pracy, a jej wysiłek i dbałość o szczegóły widać na każdej stronicy. Dziękuję, najdroższa. Nie udałoby się bez Ciebie.

Moje córki, Jemma i Lucy, które okazały wielką cierpliwość rodzinom pracującym nad książką, ściskam z całych sił. Kochamy Was.

— Charles Wyke-Smith

Charleston, Karolina Południowa, 24 września 2012

O autorze



Zdjęcie: Kelly Roper Photography, Charleston, Karolina Południowa

Zdjęcie: Kelly Roper Photography, Charleston, Karolina Południowa
Charles Wyke-Smith przez całe życie zawodowe był związany z produkcją multimediiów. W połowie lat 80. został współzałożycielem PRINTZ Electronic Design, jednej z pierwszych w pełni skomputeryzowanych agencji designerskich w San Francisco. Pracował na stanowiskach kierowniczych i jako konsultant dla Wells Fargo, ESPN Videogames oraz Benefitfocus, gdzie zajmował stanowisko dyrektora ds. UX. W roku 2009 został współzałożycielem PeopleMatter, platformy HR dla branży usługowej. Obecnie jest dyrektorem generalnym nowego start-upu, Publish — platformy umożliwiającej odkrywanie nowych książek.

Charles jest aktywnym muzykiem oraz autorem kilku książek o webdewelopingu, w tym *CSS. Witryny internetowe szyte na miarę*, *Codin' for the Web*, *Scriptin' with AJAX* i *Visual Stylin' with CSS3*. Mieszka z żoną i dwiema córkami w Charleston w Karolinie Południowej.

Spis treści

Podziękowania • iii

O autorze • iv

Spis treści • v

Wstęp • x

ROZDZIAŁ 1: KOD HTML I STRUKTURA DOKUMENTU • 1

Podstawy kodu HTML • 2

Znaczniki okalające — tekst • 2

Znaczniki nieokalające — treści wskazywane poprzez odniesienie • 3

Atrybuty • 4

Nagłówki i akapity • 5

Elementy złożone • 5

Zagnieżdżone znaczniki • 6

Budowa dokumentu HTML • 7

Szablon strony HTML • 7

Elementy blokowe i liniowe • 10

Elementy zagnieżdżone • 16

Obiektowy model dokumentu (DOM) • 20

Podsumowanie • 22

ROZDZIAŁ 2: PODSTAWY CSS • 23

Budowa reguły CSS • 24

Konwencje zapisu reguł CSS • 26

Selektory kontekstowe • 28

Wyspecjalizowane selektory kontekstowe • 32

Selektor dziecka > • 32

Selektor sąsiadującego brata + • 33

Ogólny selektor braci ~ • 33

Selektor uniwersalny * • 34

Identyfikatory i klasy • 35

Atrybut class • 35

Atrybut id • 38

Kiedy używać identyfikatorów, a kiedy klas? • 39
Identyfikatory i klasy — podsumowanie • 41
Selektory atrybutów • 41
Selektor nazwy atrybutu • 41
Selektor wartości atrybutu • 42
Selektory atrybutów — podsumowanie • 42
Pseudoklasy • 43
Pseudoklasy interfejsu • 43
Pseudoklasy strukturalne • 46
Pseudoelementy • 47
Dziedziczenie • 49
Kaskadowość • 50
Źródła stylów • 50
Zasady kaskadowości • 52
Obliczanie precyzji • 53
Deklaracje reguł • 55
Wartości słowne • 55
Wartości liczbowe • 56
Wartości kolorów • 57
Podsumowanie • 61

ROZDZIAŁ 3: POZYCJONOWANIE ELEMENTÓW • 62

Model polowy • 62
Obramowanie • 63
Dopełnienie • 66
Margines • 67
Scalanie marginesów • 68
Wybieranie jednostek miary marginesów • 69
Wielkość pola • 70
Elementy pływające i oczyszczające • 75
Właściwość float • 76
Trzy sposoby włączania pływających elementów do kontenerów • 78
Właściwość position • 86
Pozycjonowanie statyczne • 86
Pozycjonowanie względne • 87

Pozycjonowanie bezwzględne • 88
Pozycjonowanie stałe • 89
Kontekst pozycjonowania • 90
Właściwość display • 93
Tła • 93
Właściwości tła CSS • 94
Kolor tła • 95
Obraz tła • 95
Powtórzenia obrazu tła • 96
Położenie tła • 97
Wielkość tła • 99
Zaczepienie tła • 100
Właściwość zbiorcza tła • 101
Inne właściwości tła w CSS3 • 101
Większa liczba obrazów tła • 102
Gradienty tła • 104
Podsumowanie • 107

ROZDZIAŁ 4: STYLIZOWANIE FONTÓW I FORMATOWANIE TEKSTU • 108

Fonty • 108
Właściwość font-family • 109
Właściwość font-size • 112
Właściwość font-style • 115
Właściwość font-weight • 116
Właściwość font-variant • 116
Właściwość font • 117
Właściwości tekstu • 117
Właściwość text-indent • 118
Właściwość letter-spacing • 119
Właściwość word-spacing • 121
Właściwość text-decoration • 122
Właściwość text-align • 122
Właściwość line-height • 123
Właściwość text-transform • 124

Właściwość vertical-align • 125
Fonty internetowe • 126
Internetowe biblioteki fontów • 127
Gotowe zestawy @font-face • 128
Własne zestawy @font-face • 130
Stylizacja tekstu • 130
Podstawowy układ tekstu • 131
Stylizowanie tekstu w siatce • 135
Typografia klasyczna • 141
Podsumowanie • 150

ROZDZIAŁ 5: LAYOUTY • 151

Podstawy tworzenia layoutów • 151
Wysokość i szerokość layoutu • 152
Tworzenie kolumn • 153
Nadawanie kolumnom dopełnień i obramowań • 161
Trzykolumnowe layouty z płynną środkową kolumną • 172
Trzykolumnowy layout z płynną środkową kolumną i ujemnymi marginesami • 172
Trzykolumnowy layout z płynną środkową kolumną, oparty na właściwościach CSS3 table • 177
Layout wielorzędowy i wielokolumnowy • 179
Praktyczne selektory CSS • 182
Wewnętrzne elementy div w działaniu • 184
Podsumowanie • 185

ROZDZIAŁ 6: KOMPONENTY INTERFEJSU • 186

Tworzenie menu nawigacyjnych • 186
Pionowe menu • 186
Menu poziome • 189
Rozwijane menu • 191
Formularze • 201
Elementy HTML formularza • 201
Sposoby kodowania formularzy • 209
Stylizacja formularza • 210
Formularz wyszukiwania • 221

Chmurka • 224
Stosy i z-index • 227
Tworzenie trójkąta w CSS • 228
Podsumowanie • 230

ROZDZIAŁ 7: STRONA INTERNETOWA Z CSS3 • 231

Struktura strony • 231
Planowanie kodu HTML • 232
Stylizacja nagłówka • 236
Obszar tytułowy • 237
Formularz wyszukiwania • 239
Menu • 242
Obszar treści • 249
Stylizacja pola logowania • 253
Odnośniki do wpisów • 258
Obszar książek • 260
Stopka • 268
Podsumowanie • 271

ROZDZIAŁ 8: PROJEKTOWANIE SKALOWALNE • 272

Duże layouty na małych urządzeniach • 272
Zapytania medialne • 274
Reguła @media • 274
Atrybut media znacznika link • 277
Wartości graniczne • 277
Wartość viewport znacznika meta • 278
Optymalizacja layoutu na potrzeby tabletów • 278
Optymalizacja layoutu dla smartfonów • 282
Dostosowanie layoutu do orientacji pionowej • 285
Ostatnie detale • 287
Błąd ze skalowaniem w Safari Mobile • 287
Rozwijane menu na ekranach dotykowych • 287
Podsumowanie • 290

DODATEK • 291**SKOROWIDZ • 299**

Wstęp

Być dziś webdesignerem to fascynująca rzecz. Obecnie wszelkie treści medialne przyswajamy sobie przez internet. Telewizja kablowa oraz płyty CD i DVD są zastępowane przez oferujące treści na żądanie serwisy internetowe w rodzaju Hulu, Netflix, Pandora i Spotify.

W konsumpcji informacji pośredniczą różnorakie narzędzia: komputery stacjonarne, laptopy, tablety, smartfony czy też wielkie, 60-calowe ekrany.

Przekaz treści do wyżej wspomnianych urządzeń i mediów obsługuje ugruntowujący się standard technologiczny, bazujący na przeglądarkach wykorzystujących HTML5, CSS3 i JavaScript.

Kiedy niemal pięć lat temu napisałem pierwsze wydanie książki *CSS. Witryny internetowe sztyte na miarę*, standardem była sztywna, skomplikowana, oparta na XML wersja języka HTML — XHTML. Ponieważ XHTML był nieprzystosowany do wyzwolonego, pędzącego szybko świata webdewelopingu, Apple, Mozilla i Opera utworzyły wspólnie organizację Web Hypertext Application Technology Working Group. Jej celem było podjęcie na nowo prac rozwojowych nad językiem HTML, który World Wide Web Consortium porzuciło po wersji HTML 4 na rzecz XHTML. Odrodzony z popiołów feniks przyjął nazwę HTML5, a w ciągu ostatnich trzech lat nastąpiło błyskotliwe — i uzasadnione — przejście z XHTML do HTML5.

HTML5 jest przystosowany do współczesnego, multimedialnego internetu, oferując bogaty zbiór API (interfejsów programowania aplikacji, z ang. *Application Programming Interfaces*), które zapewniają wbudowaną obsługę plików audio i wideo, grafiki, geolokalizacji, magazynowania danych oraz wielu innych funkcji. HTML5 oferuje również wiele nowych elementów (`section`, `article`, `nav` itd.), pozwalających nadawać dokumentom lepszą strukturę. Wcześniej używano do tego celu semantycznie pozbawionych znaczenia elementów `div` opatrzonych atrybutami `class` i `id`, co zmniejszało uniwersalność kodu oraz jego czytelność.

Kiedy z HTML 4 przenoszono się na XHTML, a następnie na HTML5, obsługa CSS3 była systematycznie wdrażana we wszystkich przeglądarkach. CSS3, jako zestaw narzędzi graficznych, jest gigantycznym zbiorem zaleceń — tak wielkim, że podzielono go na wiele modułów, by osobne zespoły mogły je opracowywać niezależnie od siebie.

Dziś możesz wreszcie skorzystać z wyczekiwanych od dawna funkcji CSS3, takich jak obsługa gradientów, przejść, przekształceń, cieni i zaokrąglonych rogów, i mieć pewność, że owe elementy graficzne trafią do lwej części odbiorców we właściwej postaci. W przypadku starszych przeglądarek, które nie obsługują CSS3 w pełni, możesz dołączyć do swoich stron plik JavaScript o nazwie Modernizr, służący do wykrywania obsługi poszczególnych funkcji CSS3. Dzięki temu będziesz mógł zapewnić stronom kod zapasowy lub wprowadzić symulacje nieobsługiwanych funkcji CSS3 przy użyciu kodu JavaScript. Więcej na temat kodów zapasowych i symulacji wykorzystujących JavaScript przeczytasz w „Dodatku”.

Internet jest dziś o wiele przyjaźniejszy zarówno dla użytkowników, jak i dla deweloperów niż kiedykolwiek wcześniej. Najnowsze wydanie książki *CSS. Witryny internetowe sztyte na miarę* jest, jak zawsze, owocem setek godzin kodowania i pisania, spędzenia niezählonych nocy na pracy i wypicia niezählonych filiąnek herbaty. Jednocześnie jest dla mnie powodem do świętowania, ponieważ ta książka opisuje nowy stan internetu — ziszczenie od dawna żywej wizji. Wygląda na to, że dzięki pracy i wsparciu Jeffreya Zeldmana, Iana Hicksa oraz wielu innych wreszcie udało się ugruntować standardy internetowe. To uczucie, jakby po długiej wspinaczce nagle dostrzec, że jest się na szczycie góry. To, że nie muszę już pisać kodów pozwalających na obejście ograniczeń, jakie starsze przeglądarki nakładały nawet na najprostsze layouty (i to, że nie muszę marnować papieru, żeby nauczyć tego Ciebie), że cienie i zaokrąglone rogi mogą stworzyć przy użyciu jednej linijki kodu CSS, zamiast polegać na rozbudowanych obrazach i wielu warstwach elementów `div`, czy to, że każda współczesna przeglądarka potrafi wyświetlić moje strony spójnie i w pełni — jest znaczącym przełomem.

W tej książce spoglądam zatem w przeszłość. Zamiast, tak jak w przednich wydaniach, opisywać na kolejnych stronach, jak obejść problemy starych przeglądarek z kompatybilnością, koncentruję się na szerokich możliwościach, jakie oferują HTML5, CSS3 i nowoczesne przeglądarki. Internet Explorer 9 oraz kolejne wersje, Firefox, Chrome, Safari i Opera (które aktualizują się automatycznie) zachowują się niezwykle spójnie. Użytkowników starszych przeglądarek (zwłaszcza IE8 i poprzednich wersji) ubywa z dnia na dzień. Informacje o pracy ze starszymi przeglądarkami podaję wprawdzie w „Dodatku”, ale głównym tematem tej książki jest wykorzystanie CSS dziś i w przeszłości.

Kluczowe techniki

Nie musisz być wybitnym artystą czy programistą, by sprawnie korzystać z CSS, choć kompetencje z tych zakresów zdecydowanie mogą Ci się przydać. Musisz dysponować solidną znajomością HTML i CSS oraz знать kluczowe techniki i najlepsze zwyczaje. Zadaniem tej książki jest zapewnienie Ci owej znajomości, a także mocnych fundamentów pod dalszy rozwój Twoich umiejętności. CSS3 jest tak rozległy, że o niektórych jego funkcjach nawet w tej książce nie wspominam. Tak czy inaczej, uważam, że po przerobieniu przykładów z tej książki będziesz mógł szybko poszerzać swoją wiedzę i rozwijać umiejętności — a przynajmniej napisałem ją z myślą o tym celu.

Nie przepisuj mojego kodu, tylko go pobierz

Wszystkie przedstawione w tej książce kody można pobrać ze strony www.helion.pl/ksiazki/csswi3.htm. Zalecam, byś korzystał z elektronicznej wersji kodów, zamiast spisywać je z książki. Tak jest szybciej i łatwiej, a poza tym sam będę aktualizował kody, w razie gdyby pojawiły się jakieś błędy. Na towarzyszącej tej książce stronie www.stylinwithcss.com zamierzam również umieścić dział z ewentualnymi nieścisłościami, na jakie czytelnicy trafią. Znajdzie się na niej także nowy blog, który właśnie zakładam. Zapraszam Cię do czerpania z niego informacji, być może także inspiracji, oraz komentowania i zgłaszania tematów artykułów, jakie chciałbyś przeczytać. Dziękuję za kupienie tej książki. Mam nadzieję, że okaże się pomocna. Życzę powodzenia z pracą w sieci.

ROZDZIAŁ 1

Kod HTML i struktura dokumentu

CZYTASZ KSIĄŻKĘ O CSS, więc tematem pierwszego rozdziału musi być oczywiście język HTML — Hypertext Markup Language!

Zaczynam od HTML, ponieważ CSS służy do jego stylizacji.

Musisz wiedzieć, jak tworzyć kod HTML i nadawać mu strukturę, żeby móc go obstylować przy użyciu CSS. Każda strona internetowa rodzi się z kodu HTML, ponieważ pierwszą czynnością wykonywaną przy tworzeniu strony jest oznaczenie treści. Treść to wszystko, co chcesz dostarczyć odbiorcy: tekst, obrazki, a także pliki audio i wideo.

Znaczniki HTML pełnią funkcję semantyczną, czyli nadają treści znaczenie zrozumiałe dla klientów użytkownika — przeglądarki, czytników ekranowych i pajaków internetowych, które ją wyświetlają, odczytują bądź analizują. Najczęściej używane znaczniki wskazują nagłówki, akapity, odnośniki i obrazy. Obecnie istnieje w sumie 114 znaczników HTML, ale warto wziąć pod uwagę zasadę 80/20: mały zbiór 25 znaczników wystarczy do wykonania 80% czynności przy kodowaniu. Pełną listę znaczników HTML znajdziesz na stronie <http://www.w3schools.com/tags/default.asp>.

Po oznaczeniu treści możesz wykorzystać CSS do obstylowania znaczników na podstawie ich nazw, atrybutów w rodzaju **id** i **class** oraz relacji pomiędzy poszczególnymi znacznikami. Znaczniki HTML określają również hierarchię dokumentu, co pozwala na wykorzystanie CSS do utworzenia layoutu oraz obstylowanie każdego elementu w dowolny sposób.

HTML5, najnowsza wersja HTML, obsługuje nowy zbiór znaczników strukturalnych, służących do grupowania powiązanych zbiorów znaczników treści, co pozwala na lepsze określenie ogólnej struktury stron. Te znaczniki to m.in. **header**, **nav**, **article**, **section**, **aside** i **footer**. Znacznik **nav** służy na przykład do grupowania listy

odnośników służących do poruszania się po stronach wchodzących w skład witryny. W obrębie znacznika `article` możesz z kolei użyć nagłówki i akapity składające się na wpis blogowy.

Przed powstaniem HTML5 strukturę strony tworzyło się przy użyciu pozbawionych semantycznego znaczenia znaczników w rodzaju `div` i `span`. Dziś jednak istnieją znaczniki, które do tego konkretnie służą. Wszystkie te koncepcje zilustruję, oznaczając treści i pokazując, jak kod HTML tworzy hierarchię znaczników, na której operuje CSS i JavaScript.

Podstawy kodu HTML

Każdy element treści — nagłówek, akapit czy obraz — można oznaćzyć na jeden z dwóch sposobów: znacznikiem okalającym lub nieokalającym, w zależności od tego, czy znacznik ma zawierać tekst.

Znaczniki okalające — tekst

Znacznik okalający wygląda w swojej podstawowej formie następująco:

`<nazwaZnacznika>Jakaś treść tekstowa</nazwaZnacznika>`

Znacznik można opatrzyć atrybutami:

`<nazwaZnacznika atrybut_1="wartość"`

`atrybut_2="wartość">Jakaś treść tekstowa</nazwaZnacznika>`

Elementy tekstowe, w rodzaju nagłówków i akapitów, oznaczane są znacznikami okalającymi, czyli jednym otwierającym i jednym zamkającym:

`<h1>Pieskie życie</h1>`

`<p>Jestem samotnym psem włóczęgą.</p>`

Jak widzisz, znacznik składa się z nawiasów ostrokatnych, w których zawarta jest jego nazwa. Nazwa znacznika zwykle składa się z jednej litery lub skrótu odnoszącego się do właściwego mu rodzaju treści. Znacznik zamkający od otwierającego odróżnia się ukośnikiem, który poprzedza jego nazwę.

Pomiędzy znacznikiem otwierającym a zamkającym znajduje się treść, którą przeglądarka wyświetla; same znaczniki nie są przez nią wyświetlone. Znaczniki otwierające i zamkające jasno wskazują, gdzie zaczyna się i kończy tekst nagłówków i akapitów. Zauważ, że w znaczniku nagłówka znajduje się liczba `1`. Wynika to z tego, że nagłówki w HTML występują w sześciu poziomach, gdzie `h1` jest najwyższy w hierarchii.

Znaczniki nieokalające — treści wskazywane poprzez odniesienie

<nazwaZnacznika atrybut_1="wartość" atrybut_2="wartość" />

Treści nietekstowe wyświetlane są przy pomocy znaczników nieokalających. Różnica między znacznikami okalającymi a nieokalającymi polega na tym, że pomiędzy znacznikiem otwierającym i zamykającym elementu okalającego znajduje się treść, która ma być wyświetlona. Z kolei element nieokalający po prostu podaje przeglądarce *odnośnik* do treści, które mają być wyświetlone. W celu pozyskania treści podanych w nieokalających znacznikach przeglądarka podczas wczytywania strony HTML wysyła serwerowi dodatkowe żądania.

Oto obraz oznaczony znacznikiem nieokalającym:

Znaczniki zamykające

XHTML wymagał zamykania wszystkich znaczników, ale HTML5 pozwala na większą dowolność i pomijanie niektórych znaczników zamykających. Ze składnią HTML5 możesz się zapoznać na stronie <http://dev.w3.org/html5/html-author/#syntaxic-overview>.

Twój kod będzie zatwierdzany w toku walidacji jako HTML5 i wyświetlany poprawnie, jeśli na przykład otworzysz nowy znacznik akapitu bez zamknięcia poprzedniego. Sam jednak zawsze zamykam znaczniki, ponieważ dzięki temu struktura kodu jest bardziej przejrzysta, a ponadto zyskuję pewność, że zamknięte zostały wszystkie znaczniki, które tego rzeczywiście wymagają. Zalecam robić to w stosunku do wszystkich elementów, tak jak sam to robię w książce.

NieokalajĄce znaczniki musiały być zapisywane w XHTML następująco:

W HTML5 nie musisz jednak podawać zamykającego ukośnika:

Mimo to sam zapisuję znaczniki z zamykającym ukośnikiem po spacji.

HTML5 zwyczajnie go ignoruje, ale kiedy przeglądam strukturę kodu, mam od razu pewność, że znacznik został zamknięty i nie obejmuje kolejnego znacznika.



Używane przez niewidzących użytkowników czytniki ekranowe odczytują znaczniki `alt` na głos. Wobec tego koniecznie musisz oznaczać obrazy atrybutem `alt` z sensownym tekstem.

Atrybuty

Atrybuty podają przeglądarce dodatkowe informacje o znaczniku. Znacznik `img` w poprzednim przykładzie oznaczony jest dwoma atrybutami. Pierwszy, `src`, oznaczający źródło (z ang. *source*), ma wartość `cisco.jpg`. Wskazuje on, że źródłem obrazu jest plik o nazwie `ciscojpg`. Drugi atrybut, `alt`, określa alternatywny tekst, który ma zostać wyświetlony, jeśli obraz z jakiegoś powodu nie zostanie wczytany.

Każdy znacznik HTML może być oznaczony atrybutami. Jak sam zobaczyłeś w kolejnych przykładach, niektóre atrybuty — takie jak `class` i `id` — można dodać do każdego znacznika, podczas gdy inne — takie jak `src` — można dodawać jedynie do znaczników typu `img`, gdzie wskazują pliki źródłowe.



Pełny przegląd znaczników i atrybutów HTML znajdziesz na stronie [HTML Dog](http://htmldog.com/reference/htmltags) (<http://htmldog.com/reference/htmltags>).

Elementy HTML wykorzystane w tym rozdziale

Oto blokowe i liniowe znaczniki HTML, z których korzystam w tym rozdziale. Różnicę między elementami blokowymi a liniowymi omówię w dalszej części rozdziału.

BLOKOWE ELEMENTY TEKSTOWE

`h1, h2, h3, h4, h5, h6` (Sześć poziomów nagłówków, z czego `h1` jest najwyższy)

`p` akapit

`ol` uporządkowana lista

`li` pozycja w liście

`blockquote` samodzielny cytat

LINIOWE ELEMENTY TEKSTOWE

`a` odnośnik (kotwica)

`img` obraz

`em` kursywa

`strong` ważny tekst

`abbr` skrót

`cite` przypis

`q` cytat zawarty w tekście

Nagłówki i akapity

Do najczęściej używanych znaczników tekstowych na pewno należą nagłówki i akapity. Stronę zazwyczaj zaczyna się od nagłówka **h1**, z tekstem informującym czytelnika, czego ona dotyczy. Kolejny poziom treści opisuje się następnie nagłówkiem **h2**. Jeśli w obrębie tekstu opisanego nagłówkiem **h2** znajdują się dalsze podpunkty, należy skorzystać z nagłówka **h3** i kolejnych.

Nagłówek **h1** jest największy i najbardziej widoczny (chyba że zmienisz jego wygląd w CSS), a ponadto wyszukiwarki używają go w pierwszej kolejności po znaczniku **title** jako źródła słów kluczowych.

Akapity służą do oznaczania tekstu głównego i są główną formą przedstawiania wszelkich elementów tekstowych. Mówiąc krótko, jeśli jakiś tekst nie pasuje do innych znaczników tekstowych, zamieść go w akapicie.

Przejdźmy teraz do struktury dokumentu oraz elementów liniowych i blokowych. Przyjrzymy się, jak każdy element tworzy własne pole na stronie. Te właściwości kodu HTML pozwalają na szybką i sprawną stylizację dokumentu, utworzenie pożdanego layoutu i stworzenie odpowiedniej oprawy graficznej przy użyciu CSS.

Elementy złożone

HTML pozwala nie tylko na oznaczanie podstawowych treści w rodzaju nagłówków, obrazów i akapitów, ale również bardziej złożonych elementów interfejsu, takich jak listy, tabele i formularze. Służą do tego elementy złożone — zestawy znaczników, które ze sobą współdziałały.

Element oznaczający pozycję w liście **li** działa jedynie w obrębie dwóch z trzech znaczników list, mianowicie **ol** (oznaczającego uporządkowaną listę) i **ul** (oznaczającego nieuporządkowaną listę), ale nie w obrębie **dl** (czyli listy definicji). Oto prosta, uporządkowana lista **** z trzema elementami ****.

```
<ol>
    <li>Zapisz plik HTML</li>
    <li>Przenieś plik na serwer FTP</li>
    <li>Zobacz podgląd w przeglądarce</li>
</ol>
```

Na rysunku 1.1 widać tę listę w przeglądarce.

RYSUNEK 1.1. Prosta, uporządkowana lista. Przeglądarka automatycznie numeruje elementy uporządkowanej listy



Zwróć uwagę na dwie rzeczy. Po pierwsze, z pewnymi znacznikami, takimi jak `ol`, trzeba używać innych znaczników — w tym wypadku `li`. Po drugie, elementy listy `li` są „zagnieżdżone” w elemencie uporządkowanej listy `ol`, ponieważ zawarte są między jego znacznikiem otwierającym a zamkającym. Zagnieżdżanie znaczników jest bardzo ważnym zagadnieniem, które trzeba zrozumieć.

Zagnieżdżone znaczniki

W powyższym przykładzie znaczniki `li` są dziećmi znacznika `ol`, ponieważ są w nim zagnieżdżone. Z kolei znacznik `ol` jest rodzicem znaczników `li`, ponieważ je obejmuje.

Oto prosty przykład wykorzystania znacznika `em` (emfazy) do wyroźnienia wyrazu w akapicie. Niestety, znacznik dziecko `em` jest nieprawidłowo zagnieżdżony w znaczniku rodzicu `p`.

```
<p>Ten samochód jest <em>szybki</p>.</em>
```

Powinien być zapisany tak:

```
<p>Ten samochód jest <em>szybki</em>. </p>
```

Kiedy zagnieżdzasz znacznik, czyli otwierasz nowy przed zamknięciem poprzedniego, musisz go zamknąć przed zamknięciem tego, w którym jest zagnieżdżony. W pierwszym przykładzie powyżej zrobiono to niepoprawnie, ale już w drugim wszystko się zgadza.

Ogólna struktura dokumentu opiera się na tym, jak znaczniki są w sobie pozagnieżdżane, i na powstałych przez to relacjach między rodzicami a dziećmi. Zobaczysz to zjawisko w działaniu, kiedy przedstawię Ci strukturę dokumentu HTML.

Budowa dokumentu HTML



Ogólna struktura dokumentu HTML została znacznie uproszczona w HTML5. Ci, którzy mają choćby i kilka lat doświadczenia z web-dewelopingiem, z pewnością pamiętają różnorodność znaczników **DOCTYPE** dokumentów HTML i XHTML — wersje *transitional*, *strict* i oznaczenia metadanych. Zastąpiono je wszystkie prostszą składnią, z której możesz obecnie korzystać, a która odznacza się wsteczną kompatybilnością z wcześniejszymi wersjami HTML.



Znacznik komentarzy HTML pozwala na dodawanie notatek dla siebie samego i innych, którzy mogą w przyszłości pracować nad stroną. Komentarze HTML zaczynają się znacznikiem **<!--**, a kończą znacznikiem **-->**, między którymi zawiera się tekst. Komentarze są ignorowane przy wczytywaniu strony i nie są wyświetlane w przeglądarce.

W każdym dokumencie HTML, czyli na stronie internetowej, muszą się znaleźć pewne elementy. Te elementy tworzą szkielet, w którym możesz ująć treść. Te wymagane elementy możesz uznać za szablon, z którego tworzysz stronę. Wiele aplikacji do kodowania stron, jak Adobe Dreamweaver, oferuje możliwość automatycznego generowania tego szablonu dla każdej nowej strony.

Szablon strony HTML

Podstawowy szablon współczesnej strony HTML, oparty na HTML5, wygląda następująco:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Szablon strony HTML</title>
  </head>
  <body>
    <!-- tutaj umieść treść strony -->
  </body>
</html>
```

W pierwszym wierszu znajduje się nowy, uproszczony **DOCTYPE**, który zwyczajnie deklaruje, że plik jest dokumentem HTML. Zauważ, że tego znacznika nie trzeba zamkniąć.

Następny jest znacznik **html**, który nazywa się znacznikiem głównym, ponieważ wszystkie pozostałe znaczniki strony są w nim zagnieżdżone. Jego znacznik zamkijający znajduje się na samym końcu strony. Może on mieć jedynie dwójkę dzieci: **head** i **body**.

Znacznik title

Przeglądarki zwracają szczególną uwagę na znacznik **title**, a konkretnie na jego zawartość. Służy on również jako tytuł rezultatów wyszukiwania. Nie marnuj go na teksty w rodzaju „Witaj na mojej stronie!”. Koniecznie zamieśc w nim krótki opis i słowa kluczowe, których odbiorcy używają w poszukiwaniu treści i usług oferowanych przez Twoją stronę.

W znaczniku **head** zawarte są znaczniki, które pomagają przeglądarce zrozumieć, jak należy wyświetlać stronę. W tym prostym przykładzie znajdują się tam tylko dwa znaczniki: **meta** z atrybutem **charset**, który wskazuje przeglądarce, by użyć kodowania znaków UTF-8, oraz **title**, którego tekst pojawia się w pasku okna przeglądarki, gdy strona jest wyświetlana. W znaczniku **title** zamieściłem tekst „Szablon strony HTML”.

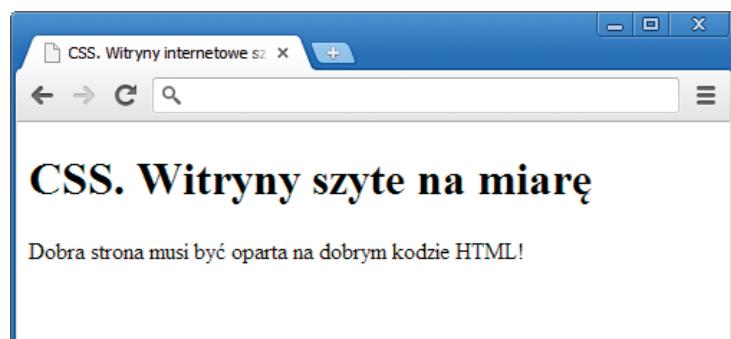
W znaczniku **body** zawarte są wszystkie elementy HTML składające się na treść. Zaczniemy od zastąpienia komentarza kilkoma znacznikami treści, a następnie rzućmy okiem na stronę w przeglądarce.

W znaczniku **body** powyższego szablonu zamieściłem następującą treść:

```
<body>
  <h1>CSS. Witryny internetowe szyte na miarę</h1>
  <p>Dobra strona musi być oparta na dobrym kodzie HTML!</p>
</body>
```

Na **rysunku 1.2** widać, jak strona wygląda w przeglądarce.

RYSUNEK 1.2. Prosta strona internetowa z nagłówkiem i akapitem



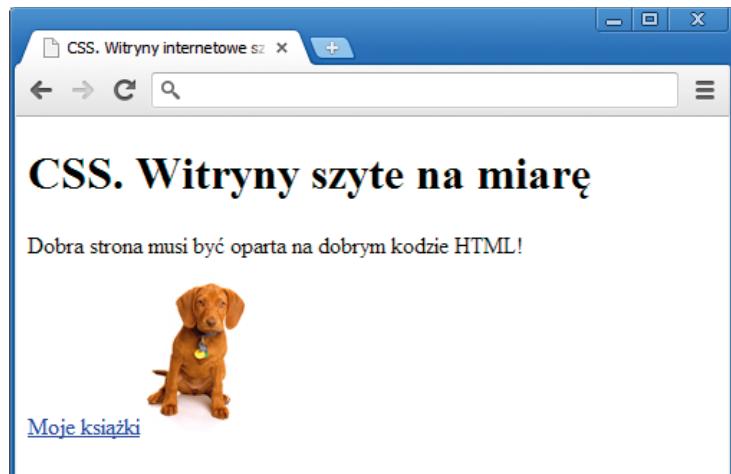
Przeglądarka rozkłada wszystkie elementy jeden po drugim, zaczynając od lewego górnego rogu. Zauważ, że nagłówek i akapit zapisane są krojem Times New Roman o różnych wielkościach, a ponadto oddzielone są odstępem. Widać wyraźnie, że już są w jakiś podstawowy sposób obstylowane. Ta domyślna stylizacja oparta jest na wbudowanym w przeglądarkę arkuszu stylów CSS i służy graficznemu odwzorowaniu wszystkich elementów HTML w sposób adekwatny, choć raczej nieciekawy.

Dodajmy teraz jeszcze dwa typowe elementy HTML: odnośnik i obraz.

```
<body>
  <h1>CSS. Witryny internetowe szyte na miarę</h1>
  <p>Dobra strona musi być oparta na dobrym kodzie HTML!</p>
  <a href="http://www.helion.pl">Moje książki</a>
  
</body>
```

Na **rysunku 1.3** widać, jak strona teraz wygląda.

RYSUNEK 1.3. Na stronie znajdują się teraz cztery elementy: nagłówek, akapit, odnośnik i obraz



Odnośnik tworzy się znacznikiem , który musi być opatrzony atrybutem `href` z adresem URL strony, do której link prowadzi.

Widać tu także przedstawiony wcześniej znacznik `img` w działaniu. Używa on atrybutu `src` zamiast `href`, ale jego wartość również jest adresem URL, który z kolei wskazuje plik z obrazem.

Zauważ, że nagłówek i akapit rozłożone są w pionie, natomiast odnośnik i obraz rozmieszczone są obok siebie. Wynika to z tego, że nagłówek i akapit są elementami blokowymi, a odnośnik i obraz elementami liniowymi.

Elementy blokowe i liniowe

Na poprzednim rysunku widać zobrazowanie wizualnej struktury dokumentu, czyli tego, co określa sposób przedstawiania na stronie kolejnych elementów HTML w kolejności zdefiniowanej w kodzie. Wizualna struktura dokumentu, zdefiniowana w arkuszu stylów przeglądarki, ma na celu zapewnienie, by dokument — opatrzony zwyczajnie poprawnym kodem HTML — został wyświetlony w prosty, ale czytelny sposób. Sztuka posługiwania się kodem CSS polega na przekształceniu tejże praktycznej, domyślnej stylizacji kodu HTML w zachęcającą, intuicyjną oprawę graficzną strony.

Niemal wszystkie elementy HTML mają właściwość `display` o wartości `block` lub `inline`. Do najbardziej wyrazistych wyjątków należą elementy tabelowe, które mają swoje własne wartości `display`.

Elementy blokowe w rodzaju nagłówków i akapitów rozmieszczane są pionowo — każdy wchodzi w skład nowego wiersza. Elementy liniowe, takie jak odnośniki i obrazy, rozmieszczone są obok siebie, a przenoszą się do kolejnego wiersza tylko wtedy, kiedy brakuje miejsca w poziomie.

Kiedy masz styczność z dowolnym elementem HTML, powinieneś sobie przede wszystkim zadać pytanie, czy masz do czynienia z elementem blokowym czy liniowym. Pozwoli Ci to przewidzieć, jak element będzie początkowo rozmieszczany, oraz zaplanować zmianę jego położenia przy użyciu CSS.

UTWORZENIE STRONY PRZY UŻYCIU ELEMENTÓW BLOKOWYCH I LINIOWYCH

Oto strona zbudowana w całości z nagłówków i akapitów. Akapity są bardzo krótkie, żeby zapewnić zrzutom i kodowi przejrzystość.

Oto kod:

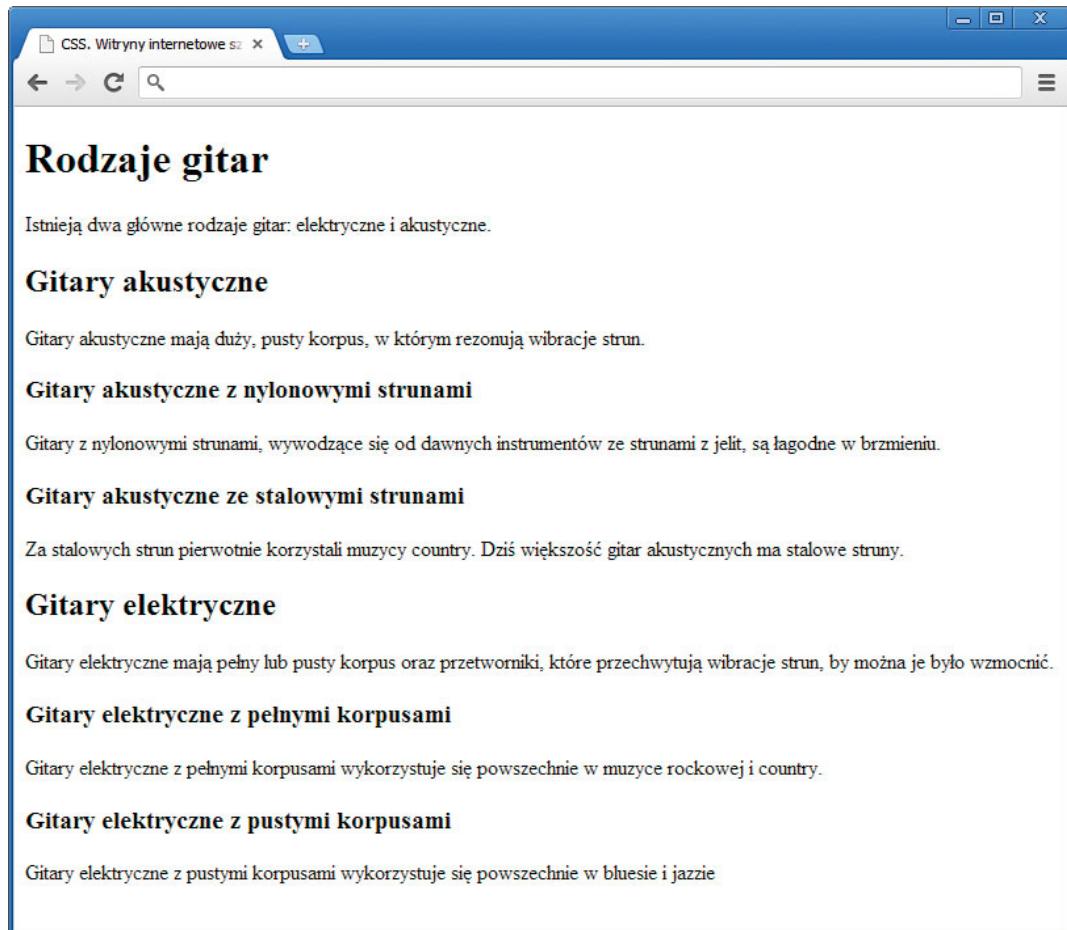
```
<!DOCTYPE html>
<html>

<head>
    <meta charset ="utf-8" />
    <title>Elementy blokowe i liniowe</title>
</head>
<body>
    <h1>Rodzaje gitar</h1>
    <p>Istnieją dwa główne rodzaje gitar: elektryczne i akustyczne.</p>
    <h2>Gitary akustyczne</h2>
    <p>Gitary akustyczne mają duży, pusty korpus, w którym rezonują wibracje strun.</p>
    <h3>Gitary akustyczne z nylonowymi strunami</h3>
    <p>Gitary z nylonowymi strunami, wywodzące się od dawnych instrumentów ze strunami z jelit, są łagodne w brzmieniu.</p>
    <h3>Gitary akustyczne ze stalowymi strunami</h3>
    <p>Ze stalowych strun pierwotnie korzystali muzycy country. Dziś większość gitar akustycznych ma stalowe struny.</p>
    <h2>Gitary elektryczne</h2>
    <p>Gitary elektryczne mają pełny lub pusty korpus oraz przetworniki, które przechwytyują wibracje strun, by można je było wzmacnić.</p>
    <h3>Gitary elektryczne z pełnymi korpusami</h3>
    <p> Gitary elektryczne z pełnymi korpusami wykorzystuje się powszechnie w muzyce rockowej i country.</p>
    <h3>Gitary elektryczne z pustymi korpusami</h3>
    <p>Gitary elektryczne z pustymi korpusami wykorzystuje się powszechnie w bluesie i jazzie</p>
</body>

</html>
```

Zawartość tej strony przedstawiona jest w postaci nagłówków i apertów. Używane standardowo znaczniki w rodzaju `article` pominąłem w celu zachowania przejrzystości.

Na rysunku 1.4 widać, jak ten kod wyświetla przeglądarka.



RYSUNEK 1.4. Strona składa się z nagłówków i akapitów

W kodzie i na rysunku widać trzy poziomy nagłówków. Przeglądarka wyświetla każdy z nich w innym rozmiarze, aby można było zobaczyć hierarchię treści na stronie. Każdy element tworzy nowy wiersz, ponieważ nagłówki i akapity są elementami blokowymi.

Zauważ też, że stronę otacza marginesy, aby tekst nie dotykał krawędzi okna przeglądarki. Światło znajduje się także między wierszami. Zanim przyjrzymy się tym odstępom dokładniej, dodajmy jeszcze kilka obrazów, które widać na **rysunku 1.5**. Jak już wiesz, obrazy są elementami liniowymi.

RYSUNEK 1.5. Dodałem do strony dwa elementy liniowe



To, że znaczniki `img` znajdują się na osobnych wierszach kodu, nie wpływa na sposób, w jaki te elementy są wyświetlane. Są to elementy liniowe, więc umieszczone są obok siebie. Znajdujące się pomiędzy znacznikami białe znaki (tabulatory, znaki łamania wiersza i spacje) są ignorowane, toteż możesz do woli rozmieszczać kod po różnych wierszach w celu zwiększenia jego czytelności. Dobrym przyzwyczajeniem jest wcinanie znaczników dzieci względem ich rodziców.

The screenshot shows a web browser window with a blue title bar and a white content area. The title bar says "CSS. Witryny internetowe Sz x". The content area has a header "Rodzaje gitar". Below it is a paragraph of text: "Istnieją dwa główne rodzaje gitar: elektryczne i akustyczne." Underneath the text is a section header "Gitary akustyczne". Below the header are two images of acoustic guitars: one orange classical guitar on the left and one red and black acoustic guitar on the right. At the bottom of the content area is another paragraph: "Gitary akustyczne mają duży, pusty korpus, w którym rezonują vibracje strun." A scroll bar is visible on the right side of the browser window.

Poniżej znajduje się kod zilustrowanej części strony, w którym widać, że dodano dwa obrazy. Te obrazy umieszczone są obok siebie, ponieważ są elementami liniowymi.

```
<body>
  <h1>Rodzaje gitar</h1>
  <p>Istnieją dwa główne rodzaje gitar: elektryczne i akustyczne.</p>
```

```
<h2>Gitary akustyczne</h2>


<p>Gitary akustyczne mają duży, pusty korpus, w którym
rezonują wibracje strun.</p>
</body>
```

Zapoznajmy się teraz bliżej z elementami blokowymi i liniowymi. W tym celu zbadamy stronę przy użyciu jednego z moich narzędzi deweloperskich, rozszerzenia Web Developer. Jest to dodatek do Firefoksa, który dodaje do przeglądarki menu narzędzi przydających się do analizowania kodu HTML, CSS i JavaScript.

RYSUNEK 1.6. Wybieram tutaj opcję z menu Wyróżnianie paska narzędzi Web Developer

 Aby pobrać i zainstalować dodatek Firefoksa Web Developer, kliknij zakładkę Pobierz dodatki w zakładce Dodatki, wyszukaj jego nazwę w pasku, a następnie kliknij przycisk Zainstaluj przy jego nazwie w liście.



Na rysunku 1.6 pod paskiem adresu widnieje pasek narzędzi Web Developer. Służy on między innymi do jasnego wskazania rozmieszczenia elementów i relacji pomiędzy nimi. Wybieram tu opcję *Wyróżnij elementy blokowe* z menu *Wyróżnianie*, która wyświetla obrysów elementów blokowych.

RYSUNEK 1.7. Po kliknięciu opcji Wyróżnij elementy blokowe w przeglądarce można zobaczyć rzeczywisty rozmiar pól elementów i odstępy między nimi. Elementy liniowe nie zostają obrysowane



Jak widać z obrysów elementów blokowych na rysunku 1.7, pola elementów są dużo większe od tekstu, który zawierają. Choć wysokość każdego z elementów jest zaledwie odrobinę większa od wysokości treści, są one szerokie na całe okno przeglądarki!

Pole elementu blokowego dostosowuje swoją szerokość do szerokości elementu rodzica.

W tym przypadku rodzicem wszystkich tych elementów jest element `body`, który domyślnie jest szeroki na całe okno przeglądarki, pomijając drobny margines. Wszystkie te elementy zatem również mają szerokość okna przeglądarki. Teraz już wiesz, dlaczego elementy blokowe zawsze tworzą nowy wiersz: zajmują całą szerokość okna, wobec czego nie pozostawiają miejsca na nic obok.

Web Developer nie umożliwia zaznaczenia wszystkich elementów liniowych na stronie tak jak blokowych, ale do tego celu można użyć opcji *Wyróżnianie niestandardowe* z menu *Wyróżnianie*. Pola elementów liniowych zachowują się odwrotnie do pól elementów blokowych.

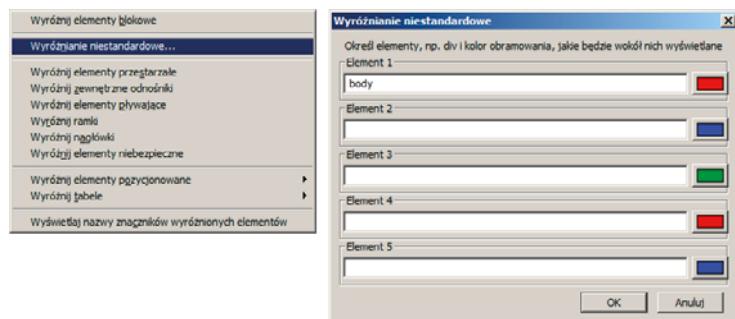
Pole elementu liniowego „kurczy się” wokół treści i okala ją możliwie ciasno.

Teraz już wiesz, dlaczego elementy liniowe umieszczane są obok siebie, a blokowe zawsze pojawiają się w nowym wierszu.

Elementy zagnieżdżone

Zobaczmy teraz, jak na ekranie przedstawiane są zagnieżdżone elementy HTML. Wszystkie elementy w poprzednim przykładzie miały wspólnego rodzica — element `body`. Wymagany element `body` zawsze jest w dokumencie obecny, więc Web Developer standardowo go nie wyświetla, aczkolwiek obramowanie jego pola możesz wyświetlić przy pomocy opcji *Wyróżnianie niestandardowe* z menu *Wyróżnianie*, tak jak na **rysunku 1.8**.

RYSUNEK 1.8. Wybierz opcję *Wyróżnianie niestandardowe*, a następnie w oknie wybierz elementy do wyświetlenia i przypisz im kolory



Na **rysunku 1.9** widać, że pole elementu rodzica `body` (zaznaczone na niebiesko) okala wszystkie swoje dzieci (zaznaczone na czerwono). Te elementy blokowe są rozszerzone tak, by dostosować się do elementu `body`, którego domyślna szerokość równa się szerokości okna przeglądarki (poza drobnym marginesem).

Kiedy zagnieżdzasz w kodzie znaczniki HTML, zagnieżdzasz na ekranie pola elementów.

RYSUNEK 1.9. Element rodzić **body** okala swoje dzieci

The screenshot shows a Firefox browser window with the title bar "Firefox" and the address bar "CSS. Witryny internetowe sztyte na miarę - ...". Below the address bar is a toolbar with various icons. The main content area displays a page titled "Rodzaje gitar".
Gitary akustyczne
Two acoustic guitars are shown side-by-side: one orange and one red.
Text: "Istnieją dwa główne rodzaje gitar: elektryczne i akustyczne."
Section: "Gitary akustyczne"
Text: "Gitary akustyczne mają duży, pusty корпус, w którym rezonują vibracje strun."
Section: "Gitary akustyczne z nylonowymi strunami"
Text: "Gitary z nylonowymi strunami, wywodzące się od dawnych instrumentów ze strunami z jelit, są łagodne w brzmieniu."
Section: "Gitary akustyczne ze stalowymi strunami"
Text: "Za stalowych strun pierwotnie korzystali muzycy country. Dziś większość gitar akustycznych ma stalowe strony."
Gitary elektryczne
Two electric guitars are shown side-by-side: one black and one orange.
Text: "Gitary elektryczne mają pełny lub pusty корпус oraz przetworniki, które przechwytyują vibracje strun, by można je było wzmacnić."
Section: "Gitary elektryczne z pełnymi korpusami"
Text: "Gitary elektryczne z pełnymi korpusami wykorzystuje się powszechnie w muzyce rockowej i country."
Section: "Gitary elektryczne z pustymi korpusami"
Text: "Gitary elektryczne z pustymi korpusami wykorzystuje się powszechnie w bluesie i jazzie"

Na stronach z wieloma elementami zagnieżdżanie bywa wielopoziomowe, wobec czego uporządkowanie kodu HTML tak, by można było z niego odczytać strukturę kodu, pozwala upewnić się co do poprawnego zagnieżdżenia znaczników. Zagnieżdżanie wskazuje wcięciami. Każdy poziom warto wcinać czterema spacjami, które tutaj przedstawiam jako kropki.



Niektóre edytory HTML, takie jak Dreamweaver, dodają cztery spacje, gdy naciskasz klawisz Tab, co zapewnia spójność, a ponadto oszczędza stukania w klawisze.

```
<nav id="spis">
....<ol>
.....<li><a href="#">Wstęp</a></li>
.....<li><a href="#">Rozdział 1</a></li>
.....<li><a href="#">Rozdział 2</a></li>
.....<li><a href="#">Rozdział 3</a></li>
....</ol>
</nav> <!--koniec spisu treści -->
```

DWA PRZYKŁADY ZAGNIEŻDZANIA W KODZIE

Spójrzmy na kolejny przykład zagnieżdżania, tym razem z wykorzystaniem znacznika **blockquote**. Element **blockquote** służy do zaznaczania cytatów będących osobnymi elementami wizualnymi na stronie. Zwróć uwagę na wykorzystanie encji HTML do utworzenia prawidłowych cudzysłów.

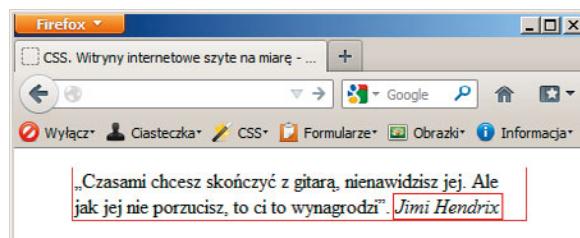
```
<blockquote>&bdquo;Czasami chcesz skończyć z gitarą,  
nienawidzisz jej. Ale jak jej nie  
porzucisz, to ci to wynagrodzi&rdquo;.
```

przypisem **cite** zaznaczam ————— imię i nazwisko autora cytatu

<cite>Jimi Hendrix</cite>
</blockquote>

Na rysunku 1.10 widać, jak to wygląda na ekranie. Obydwa elementy są obrysowane.

RYSUNEK 1.10. Element **blockquote** jest domyślnie wcięty



Encje HTML

Encje HTML służą zwykle do generowania znaków typograficznych, których nie ma na klawiaturze, np.: ™, †, © i innych. Encje HTML zaczynają się od etki, kończą średnikiem, a zawierają ciąg znaków, które reprezentują daną encję. W powyższym przykładzie widnieją encje z nazwami cudzysłowu otwierającego i zamknięjącego.

Moja koleżanka Elizabeth Castro, też autorka książek wydawanych przez Peachpit (które zresztą bardzo polecam), podaje listę powszechnie używanych encji HTML na stronie <http://www.elizabethcastro.com/html/extras/entities.html>.

Zauważ, że ponieważ encje zaczynają się od znaku etki, to ona sama ograniczona jest do pełnienia tej funkcji. Jeżeli chcesz zamieścić samą etkę w treściach zawartych w znacznikach HTML, musisz użyć encji &#amp;.

Na przykład **Kowalski & Kowalski** wyświetlany jest jako **Kowalski & Kowalski**.

Znacznik **blockquote** jest elementem blokowym. Trudno się dziwić, skoro ma on być osobnym elementem na stronie.

W elemencie **blockquote** (oznaczony na czerwono), po zacytowanym tekście, zagnieźdzony jest przypis (oznaczony na zielono) w postaci znacznika liniowego **cite**. Nie brak mu miejsca, by znaleźć się obok tekstu akapitu. Jak widzisz, znacznik **cite** domyślnie stylizowany jest kursywą.

Także w tym przykładzie widnieją dwie encje HTML: „ i ”, które tworzą typograficznie poprawne cudzysłowy.

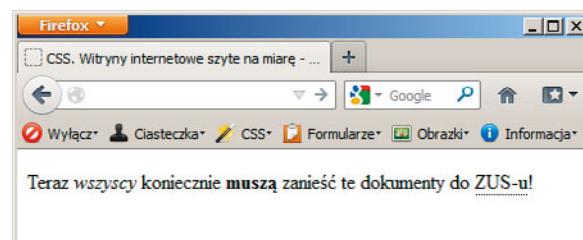
Wykorzystanie tych encji zamiast stawiania prostych cudzysłówów klawiszami *Shift+”* dodaje oprawie graficznej nieco bardziej profesjonalnego charakteru.

Spójrzmy na drugi przykład z trzema elementami liniowymi zawartymi w elemencie blokowym.

```
<p>Teraz <em>wszyscy</em> koniecznie <strong>muszą</strong>  
zanieść te dokumenty do <abbr title="Zakładu Ubezpieczeń  
Społecznych">ZUS-u</abbr>!</p>
```

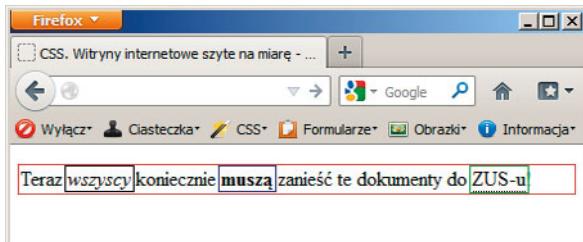
Na **rysunku 1.11** widać, jak ten kod wygląda w Firefoksie.

RYSUNEK 1.11. Akapit zawiera znaczniki, które wyróżniają tekst, zwracając szczególną uwagę na jego ważność oraz wskazując skrót



Na rysunku 1.12 widnieje ten sam tekst z wyświetlonymi polami elementów.

RYSUNEK 1.12. Trzy elementy liniowe zawarte w elemencie blokowym



Poza poczuciem, że masz coś ważnego do zrobienia, z powyższego przykładu możesz wynieść jeszcze kilka rzeczy.

- Tekst oznaczony jest jako akapit i zawiera trzy znaczniki liniowe.
- Znacznik **strong** wskazuje coś ważnego i domyślnie wyświetlany jest pogrubionym tekstem.
- Znacznik **em** wyróżnia tekst i domyślnie wyświetlany jest kursywą.
- Znacznik **abbr** wskazuje skrót i — w Firefoksie — domyślnie występuje jako tekst podkreślony kropkami.

Widziałeś już, jak kod HTML tworzy na stronie pola. Wiesz także, że zagnieźdzając znaczniki, zagnieźdzasz pola na ekranie. Przejedźmy teraz do oglądu kodu HTML — DOM, czyli obiektowego modelu dokumentu (z ang. *Document Object Model*).

Obiektowy model dokumentu (DOM)

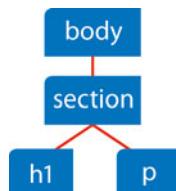
Ostatnim zagadnieniem z zakresu HTML, które musimy omówić przed przejściem do CSS, jest to, jak struktura kodu HTML tworzy obiektowy model dokumentu, do którego będę się odtąd odnosił skrótem DOM. DOM jest oględem elementów na stronie według przeglądarki. Odzwierciedla on stan właściwości wszystkich elementów, na podstawie którego przeglądarka może określić relacje panujące w „drzewie rodzinnym” elementów. Odnosząc się w kodzie CSS do konkretnej lokalizacji w modelu DOM, możesz wybrać element HTML i zmodyfikować jego właściwości stylu.

Z DOM możemy się teraz zaznajomić z pomocą poniższego przykładu.

w poprawnie wciętym kodzie HTML widoczna jest hierarchia znaczników

```
<body>
  <section>
    <h1>Obiektowy model dokumentu (DOM)</h1>
    <p>Struktura kodu HTML strony określa DOM.</p>
  </section>
</body>
```

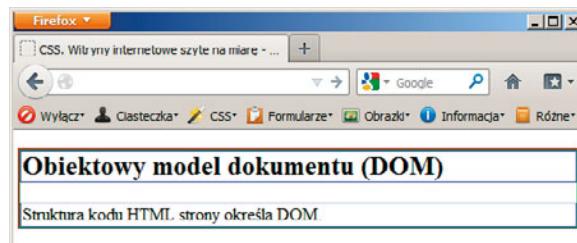
RYSUNEK 1.13. Proste przedstawienie kodu HTML jako hierarchii DOM



Powyższy kod i **rysunek 1.13** przedstawiają typową strukturę strony, gdzie znacznik strukturalny (w tym wypadku `section`) zawiera znaczniki dzieci (tutaj `h1` i `p`). Wcięcie kodu wskazuje na relację w obrębie hierarchii rodzinnej. Tę hierarchię można też ująć w pionowym wyobrażeniu, zupełnie jak drzewo rodzinne.

Na **rysunku 1.14** widać kod w przeglądarce, z zaznaczonymi obramowaniemi elementów.

RYSUNEK 1.14. Na rysunku widać, że element `body` (zaznaczony na czerwono) obejmuje element `section` (zaznaczony na zielono), który z kolei obejmuje dwa elementy dzieci, nagłówek i akapit (zaznaczone na niebiesko)



Oto, co możemy powiedzieć o tej hierarchii DOM:

- `section` jest **rodzicem** `h1` i `p` — ich bezpośrednim przodkiem;
- `h1` i `p` są **dziećmi** `section` — jego bezpośredniimi potomkami;
- `h1` i `p` są **braćmi** — mają wspólnego rodzica, `section`;

- `section`, `h1` i `p` są **potomkami** `body` — są w nim gdzieś zagnieżdżone;
- `section` i `body` są **przodkami** `h1` i `p` — na jakimś poziomie zawierają je.

W kolejnych rozdziałach często będę się posługiwał pojęciami dzieci, rodziców, braci, przodków i potomków, więc warto mieć pewność, do czego się odnoszą, a także w pełni rozumieć, jak zagnieżdżanie elementów HTML kształtuje hierarchię DOM.

CSS posługuje się DOM, wybierając najpierw element lub zbiór elementów, których właściwości następnie modyfikuje. Kiedy używasz CSS do zmodyfikowania właściwości elementu takiej jak szerokość albo umieszczasz w kodzie pseudoelement, zmiany są wprowadzane do DOM, który aktualizuje wygląd strony.

Mówiąc krótko, DOM tworzysz kodem HTML, a następnie możesz go zmodyfikować kodem CSS, który jest wykorzystywany przy wczytywaniu strony oraz w toku interakcji użytkownika ze stroną.

Podsumowanie

W tym rozdziale zobaczyłeś, jak znaczniki HTML nadają treści strukturę. Dowiedziałeś się też, że każdy element tworzy na ekranie własne pole. Poznałeś różnicę między elementami blokowymi i liniowymi. Dowiedziałeś się również, że w wyniku zagnieżdżania elementów powstaje między nimi hierarchiczna relacja oraz że zagnieżdżanie elementów HTML w kodzie tworzy na ekranie zagnieżdżone pola. Wreszcie, dowiedziałeś się, że DOM jest oględem dokumentu z perspektywy przeglądarki i że kodem CSS można zmienić właściwości stylu elementów w DOM, tym samym zmieniając layout i wygląd samej strony. Ta wiedza jest podstawą dla udanego stylizowania kodu HTML przy użyciu CSS. W rozdziale 2. omówię zasady działania CSS oraz sposób, w jaki wykorzystują znaczniki HTML.

ROZDZIAŁ 2

Podstawy CSS

W ROZDZIALE 1. DOWIEDZIAŁEŚ SIĘ, jak struktura dokumentu powstaje na podstawie kodu HTML. W tym rozdziale pokażę Ci, jak wykorzystywać reguły CSS do stylizowania kodu HTML, oraz wytłumaczę, na czym polega działanie kaskadowości, czyli zasady, według której CSS decyduje o pierwszeństwie, gdy do właściwości jakiegoś elementu odnosi się więcej niż jeden styl.

Każdy element HTML określany jest właściwościami stylu, które można zdefiniować przy użyciu CSS. Te właściwości odnoszą się do różnych aspektów wizualnych elementu, np. jego położenia na ekranie, szerokości jego obramowania, jego wielkości, koloru, kroju tekstu i innych. CSS jest mechanizmem wybierania elementów HTML i definiowania ich właściwości. Wybór elementu i powiązanie z nim stylu nazywamy regułą CSS.

Przyjrzyjmy się zatem regułom CSS i przypisywaniu ich elementom HTML. Sam możesz dodawać przedstawione tu przykłady do poniższego prostego szablonu HTML5 albo po prostu skorzystać z przykładów do pobrania ze strony <http://www.helion.pl/ksiazki/csswi3.htm>.

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Szablon HTML5</title>
<style>
/* Reguły CSS umieszcza się między znacznikami style */
</style>
</head>
<body>
```



W tym przykładzie widać różnicę między formatowaniem komentarzy w CSS a w HTML.

```
<!-- Elementy HTML umieszcza się między znacznikami body -->  
</body>  
</html>
```

Widać, że zastosowano tutaj znacznik HTML **style**. Znacznik ten umożliwia dodawanie (czy też, mówiąc ściślej, osadzanie) stylów CSS bezpośrednio w dokumencie. Przeglądarka nadaje style zawarte w znaczniku **style** elementom HTML znajdującym się w znaczniku **body**.

Budowa reguły CSS

Reguła to termin, który odnosi się do pełnej instrukcji CSS. Reguła deklaruje element do zmodyfikowania oraz style, które mają mu być przypisane.

Oto przykład reguły CSS, która nadaje tekstowi akapitu kolor czerwony.

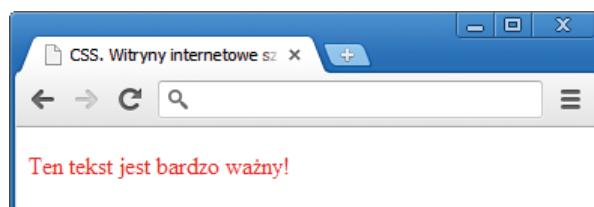
```
p {color: red;}
```

Po przypisaniu jej poniższemu kodowi HTML

```
<p>Ten tekst jest bardzo ważny!</p>
```

tekst elementu zmieni kolor na czerwony, tak jak widać na rysunku 2.1.

RYSUNEK 2.1. Prosty selektor znaczników CSS został wykorzystany do obstylowania elementu HTML



 Ten szablon HTML i wszystkie przykłady kodów z tej książki znajdziesz na stronie <http://www.helion.pl/ksiazki/csswi3.htm>.

Oto jak ten kod wygląda w obrębie szablonu HTML5.

```
<!DOCTYPE html>  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8" />  
<title>Szablon HTML5</title>  
<style>  
Ciąg dalszy
```

Trzy sposoby dołączania stylów do dokumentu

Kod CSS można dołączyć do strony na trzy sposoby: lokalnie, osadzając go i linkując do niego.

STYLE LOKALNE

Lokalne deklaracje stylów dołączają się do znaczników przy użyciu atrybutu HTML `style`:

```
<p>Ten akapit oznaczony jest domyślnym stylem akapitów przeglądarki.</p>
<p style="font-size: 12px; font-weight:bold; font-style:italic; color:red;">Dodając lokalnie kod
CSS do akapitu, przesyłasz domyślne style.</p>
```

Zakres działania stylów lokalnych jest bardzo ograniczony. Styl lokalny wpływa tylko na znacznik, do którego jest dołączony, i zawsze przesyła style osadzone oraz te, do których odnośnik znajduje się w dokumencie.

STYLE OSADZONE

Osadzone style CSS umieszcza się w nagłówku dokumentu HTML:

```
<head>
<!-- tutaj umieszcza się inne elementy nagłówka (np. znaczniki meta, title) -->
<style type="text/css">
  h1 {font-size:16px;}
  p {color:blue;}
</style>
</head>
```

Zakres działania osadzonych stylów ograniczony jest do strony. Style stron przesyłają style z arkuszy, ale pierwszeństwo przed nimi mają style lokalne. Style osadzone bywają użyteczne przy tworzeniu stylów w ramach produkcji poszczególnych komponentów takich jak menu (co robię w najwcześniejjszych przykładach w tej książce), aby mieć kod HTML i CSS na jednej stronie. Kiedy już jednak CSS jest w pełni działający i sprawdzony, warto takie style przenieść do arkusza, żeby mogły z nich korzystać także inne strony.

STYLE ZEWNĘTRZNE

Przy tworzeniu witryny składającej się z więcej niż jednej strony warto zamieścić style w osobnym dokumencie, zwany arkuszem stylów. Arkusz stylów jest po prostu plikiem tekstowym z rozszerzeniem .css. Arkusz stylów możesz podłączyć do dowolnej liczby stron HTML, zamieszczając w ich nagłówkach jedną linijkę kodu:

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

Zakres działania stylów zewnętrznych obejmuje całą witrynę. Wystarczy użyć znacznika HTML `link`, aby podłączyć arkusz stylów do każdej strony, która ma z niego korzystać. Zmiany stylów w arkuszu wpływają na wygląd wskazanego elementu, kiedy tylko taki pojawi się na stronie. Dzięki temu można uzyskać spójną stylizację w obrębie całej witryny oraz z łatwością aktualizować jej wygląd w całości.

Arkusze stylów można również podłączać do innych arkuszy przy użyciu reguły `@import`:

```
@import url(css/styles2.css)
```

Zauważ, że reguły `@import` muszą znajdować się przed wszelkimi innymi regułami CSS w arkuszu, gdyż inaczej podane w nich style się nie załadują.

Kod CSS zamieszczony jest w znaczniku **style** w nagłówku strony

Ciąg dalszy

```
p {color: red;}  
</style>  
</head>  
<body>  
<p>Ten tekst jest bardzo ważny!</p>  
</body>  
</html>
```

Kod HTML zamieszczony jest w znaczniku **body**



W arkuszach stylów nie należy umieszczać znaczników **style**, gdyż arkusze się wtedy nie wczytują.

Należy koniecznie zwrócić uwagę na to, że style CSS osadzone są w stronie przy użyciu znacznika **style**. Kiedy przeglądarka trafia na otwierający znacznik **style**, zamiast interpretować kod jako HTML, zaczyna go interpretować jako CSS. Po dojściu do znacznika zamkajającego ponownie zaczyna interpretować kod jako HTML.

Użyj tego szablonu do wypróbowania kolejnych podanych w tym rozdziale przykładów. Zamień powyższy szablon w pliku tekstowym, zapisz go z rozszerzeniem *.html*, a następnie powklejaj kody CSS i HTML z każdego przykładu do szablonu tak, jak to pokazałem. Zapisz, a następnie otwórz te pliki w przeglądarce, aby zobaczyć rezultat.

Konwencje zapisu reguł CSS

RYSUNEK 2.2. Reguła CSS składa się z dwóch głównych elementów: selektora i deklaracji. Deklaracja składa się z dwóch kolejnych elementów: właściwości i wartości. Deklaracje zawarte są między nawiasami klamrowymi



Reguła CSS składa się z dwóch części: selektora, który wskazuje element wybrany przez regułę — w tym przypadku akapit — oraz deklaracji. Deklaracja składa się z dwóch części: właściwości, która określa, jaki aspekt graficzny elementu ma zostać zmodyfikowany — w tym przypadku kolor tekstu — oraz wartości, która określa nowy stan właściwości — w tym przypadku jest to kolor czerwony.

Zauważ, że po selektorze znajduje się nawias otwierający. Właściwość oddzielona jest od wartości dwukropkiem, a deklaracja kończy się średnikiem. Cała reguła natomiast kończy się nawiasem zamkającym.



W tej książce zapisuję deklaracje w jednym wierszu, żeby zaoszczędzić miejsca. Sam, w celu zachowania przejrzystości, możesz w kodzie CSS zamieścić kolejne deklaracje w osobnych wierszach. Zauważ, że CSS ignoruje białe znaki między deklaracjami, więc możesz formatać swój kod wedle upodobania, używając enterów, spacji i tabulatorów.

Warto, byś się uważnie przyjrzał rysunkowi 2.2, żeby zdobyć całkowitą jasność co do tych pojęć. Będziemy się nimi często posługiwać. Podstawową strukturę reguły można rozszerzyć na trzy sposoby:

1. W regule można zawrzeć więcej niż jedną deklarację.

```
p {color:red; font-size:12px; font-weight:bold;}
```

Tekst akapitu jest teraz czerwony, ma wielkość 12 pikseli i jest pogrubiony.

Zauważ, że każda deklaracja oddzielona jest od kolejnej średnikiem. Ostatni średnik przed nawiasem zamykającym jest opcjonalny, ale zawsze go stawiam, żeby nie myśleć o tym, w razie gdybym chciał dodać kolejną.

Być może zastanawiasz się, jakich innych wartości mogą używać właściwości w rodzaju `font-size` i `color`. Warto wiedzieć, że można określić kolor modelem RGB (czyli kolorem czerwonym, zielonym i niebieskim, z ang. *red, green, blue*), zamiast podawać jego nazwę. Najpierw jednak pokażę Ci, jak selektory działają, a dopiero w dalszej części rozdziału omówię deklaracje reguł.

2. Selektory można zgrupować.

Jeżeli na przykład chcesz, by tekst znaczników `h1`, `h2` i `h3` był niebieski i pogrubiony, to możesz podać poniższy kod w całości:

```
h1 {color:blue; font-weight:bold;}  
h2 {color:blue; font-weight:bold;}  
h3 {color:blue; font-weight:bold;}
```

Możesz jednak uniknąć takiego powtarzania się, grupując selektory w pojedynczej regule:

```
h1, h2, h3 {color:blue; font-weight:bold;}
```

Koniecznie zamieść przecinek po każdym selektorze poza ostatnim. Spacji między selektorami nie trzeba stawiać, ale dzięki nim kod jest bardziej czytelny.

3. Do jednego selektora można przypisać więcej niż jedną regułę.

Jeśli po napisaniu poprzedniej reguły zdecydujesz, że chciałbyś, żeby znacznik `h3` był także zapisywany kursywą, to możesz napisać osobną regułę dla `h3`:

```
h1, h2, h3 {color:blue; font-weight:bold;}  
h3 {font-style:italic;}
```

Te trzy struktury reguł są podstawą bardziej złożonych wyborów, których możesz dokonywać. Przy tworzeniu kodu CSS prawdopodobnie zechcesz, by jakiś konkretny element, jak na przykład akapit, wyglądał inaczej w pasku bocznym niż w artykule. Tymczasem jak na razie mieliśmy do czynienia z regułami, które wpływały na wszystkie znaczniki określonego rodzaju w dokumencie. Zobaczmy teraz, jak pisać reguły odnoszące się do konkretnych elementów w kodzie.

Takie szczegółowe selektory można zaklasyfikować do trzech głównych grup:

- **Selektory kontekstowe** — wybierają element na podstawie jego przodka lub brata.
- **Selektory identyfikatora i klasy** — wybierają element na podstawie atrybutów `id` i `class`.
- **Selektory atrybutów** — wybierają element na podstawie informacji zawartych w ich atrybutach.

Selektory kontekstowe

Wyobraź sobie, że chcesz, aby tekstu akapitu był wyświetlany w jednym rozmiarze w obrębie artykułu w głównym obszarze treści na stronie, a w innym, gdy znajduje się w pasku bocznym. Takie „lokalizacyjne” zróżnicowanie stylów dla wybranego znacznika można uzyskać przy pomocy selektorów kontekstowych, o których zaraz dowieš się więcej.

```
znacznik1 znacznik2 {deklaracje}
```

`znacznik2` wybierany jest tylko, jeśli jego przodkiem jest `znacznik1`.

Selektory kontekstowe, ściśle nazywane *selektorami potomka*, deklarują sekwencję oddzielonych spacjami nazw znaczników. Służą do wybierania znaczników będących potomkami wybranych przodków.

```
article p {font-weight: bold;}
```

W tym przykładzie selektora kontekstowego reguła odnosi się jedynie do znaczników `p`, które są potomkami znaczników `article`. Innymi słowy, wybrany w powyższej regule znacznik jest znacznikiem `p` zawartym w jakimś stopniu w znaczniku `article`.

Przyjrzyjmy się temu bardziej szczegółowo, wykorzystując poniższy przykład:

```
<body>
  <article>
    <h1>Selektory kontekstowe są <em>bardzo</em>
    szczegółowe</h1>
    <p>W tym przykładzie widnieje wybór <em>konkretnego</em>
    znacznika.</p>
  </article>
  <aside>
    <p>Selektory kontekstowe są <em>bardzo</em> przydatne!</p>
  </aside>
</body>
```

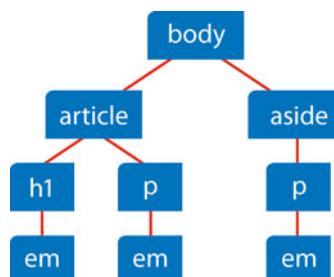
Ten kod, bez stylizacji w CSS, wyświetlany jest w przeglądarce tak jak na **rysunku 2.3**.

RYSUNEK 2.3. Kod obstyłowany domyślnym stylem przeglądarki



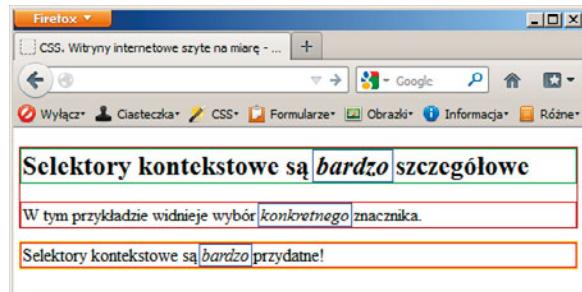
Kod tworzy DOM, którego wyobrażenie widać w diagramie hierarchii na **rysunku 2.4**.

RYSUNEK 2.4. Diagram hierarchii DOM powyższego kodu



Na **rysunku 2.5** widnieje struktura pól wytworzona na tej stronie przez hierarchię DOM.

RYSUNEK 2.5. Każdy poziom hierarchii tworzy pole okalające elementy dzieci



 Obrys pól wyświetlikiem, wybierając opcje Wyróżnij elementy blokowe i Wyróżnianie niestandardowe (dla których podałem nazwy elementów liniowych) w dodatku Developer Toolbar w Firefoksie.

RYSUNEK 2.6. Prosta, wybierająca jeden znacznik reguła zmienia kolor tekstu wszystkich elementów `em` na zielony

Powyższy diagram przyda Ci się, gdy będę demonstrował reguły CSS wybierające konkretne elementy w kodzie tego przykładu.

Zacznę od prostego selektora jednego znacznika.

`em {color:green;}`

Ta reguła wybiera wszystkie znaczniki `em` na stronie, zmieniając kolor wszystkich elementów `em` na zielony (**rysunek 2.6**).



 Pamiętaj, że w odróżnieniu od przedstawionych wcześniej selektorów grupowych w selektorach kontekstowych poszczególne członki oddzielone są spacjami, a nie przecinkami.

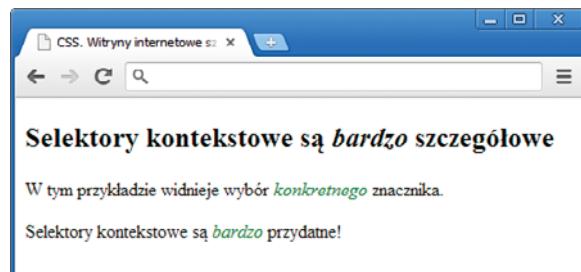
Zmieńmy ten selektor kontekstowy tak, by wskazywał znaczniki bardziej szczegółowo.

`p em {color:green;}`

Teraz `p` określa kontekst, a `em`, jako że jest podany ostatni, jest wybieranym znacznikiem. Tą regułę można wyrazić następująco: „wybierz wszystkie znaczniki `em`, których przodkiem jest znacznik `p`”. Spójrz na diagram hierarchii. Które znaczniki zostaną wybrane przez tę regułę?

Otoż ta reguła wybiera znaczniki `em` zawarte w dwóch akapitach, ale pomija znacznik `em` znajdujący się w nagłówku, który nie ma akapitu jako przodka (**rysunek 2.7**).

RYSUNEK 2.7. Wobec tego, że podano akapit jako kontekst, reguła nie oddziałuje na znacznik `em` zawarty w nagłówku

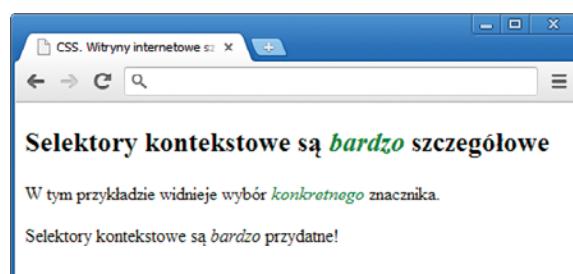


Na **rysunku 2.8** widnieje inny wariant tego selektora z jednym elementem kontekstowym. Zmieńmy regułę CSS następująco:

```
article em {color:green;}
```

Efekt widać poniżej.

RYSUNEK 2.8. Kiedy elementem kontekstowym jest znacznik `article`, znacznik `em` zawarty w elemencie `aside` zostaje pominięty



Teraz elementem kontekstowym jest znacznik `article`, wobec czego tekst zawarty w znacznikach `em` znajdujących się w nagłówku `h1` i akapicie znacznika `article` jest zielony. Tekst w znaczniku `em` zawartym w elemencie `aside` zostaje pominięty, ponieważ `em` w `aside` nie ma przodka `article`. Zwrót uwagę na to, że obecność znacznika `p` między `em` a `article` w hierarchii nie ma znaczenia. Znacznik `em` musi po prostu mieć jakiegoś przodka `article` nad sobą w hierarchii, żeby ten selektor zadziałał.

Jeżeli chcesz wybrać jedynie znacznik `em` znajdujący się w nagłówku, musisz doprecyzować kontekst.

```
article h1 em {color:green;}
```

Teraz wybieram jedynie znacznik `em` będący potomkiem znacznika `h1`, który jest potomkiem znacznika `article`.

RYSUNEK 2.9. Wykorzystanie dwóch selektorów kontekstowych pozwala na wybranie znacznika bardziej szczegółowo



Jak widać na **rysunku 2.9**, czasami konieczne jest stworzenie ciągu selektorów kontekstowych, żeby wskazać konkretne ten znacznik, który się chce.

Wyspecjalizowane selektory kontekstowe

Jak dotąd zetknąłeś się z selektorami korzystającymi ze znaczników przodków do określenia kontekstu. Dopóki dany znacznik ma znacznik kontekstowy nad sobą w hierarchii, to zostaje on wybrany. Nie jest istotne, jak wiele innych znaczników znajduje się w hierarchii między podanymi selektorami. Czasami jednak może Ci zależeć na tym, by kontekst był bardziej precyzyjny niż „jakiś przodek”. Co zrobić, jeśli chcesz wybrać element w odniesieniu do znacznika będącego jego bezpośrednim rodzicem lub poprzedzającym bratem?

Skorzystajmy z nowego kodu HTML do zademonstrowania kilku wyspecjalizowanych selektorów kontekstowych.

```
<section>
  <h2>Nagłówek H2</h2>
  <p>Oto akapit pierwszy</p>
  <p>W akapicie drugim znajduje się <a href="#">odnośnik</a>.</p>
  <a href="#">Odnośnik</a>
</section>
```

Selektor dziecka >

`znacznik1 > znacznik2`

znacznik2 musi być dzieckiem elementu *znacznik1*. Innymi słowy, *znacznik1* musi być rodzicem elementu *znacznik2*. W odróżnieniu od standardowego selektora kontekstowego ta reguła nie jest wykorzystywana, jeśli *znacznik1* znajduje się w jakimś wyższym miejscu w hierarchii.

```
section > h2 {font-style:italic;}
```

Na **rysunku 2.10** widać, jak tę regułę wykorzystuje przeglądarka.

RYSUNEK 2.10. Znacznik **h2** został wybrany, ponieważ jest dzieckiem znacznika **section**



Selektor sąsiadującego brata +

```
znacznik1 + znacznik2
```

znacznik2 musi się znajdować bezpośrednio za swoim bratem *znacznik1*.

```
h2 + p {font-variant:small-caps;}
```

Na **rysunku 2.11** widać, jak tę regułę wykorzystuje przeglądarka.

RYSUNEK 2.11. Pierwsze **p** zostało wybrane, ponieważ jest pierwszym bratem **h2**



Ogólny selektor braci ~



Znak ~ (tylde) uzyskasz, przytrzymując Shift i naciskając klawisz znajdujący się po lewej stronie klawisza 1.

```
znacznik1 ~ znacznik2
```

znacznik2 musi znajdować się za bratem *znacznik1*, choć nieznacznie bezpośrednio po nim.

```
h2 ~ a {color:red;}
```

Na rysunku 2.12 widać, jak tę regułę wykorzystuje przeglądarka.

RYSUNEK 2.12. Wybrany został tylko ten znacznik, który jest bratem



Selektor uniwersalny *

* (Naciśnij *Shift+8*)

 Właściwość `color` określa kolor pierwszego planu, co wpływa zarówno na kolor tekstu, jak i obramowania. Używa się jej jednak głównie do nadawania koloru samemu tekstowi.

Selektor uniwersalny * odnosi się do każdego elementu, więc jeśli podasz kod

```
* {color:green;}
```

to tekst i obramowania wszystkich elementów staną się zielone.

Z zasady używa się go z innymi selektorami:

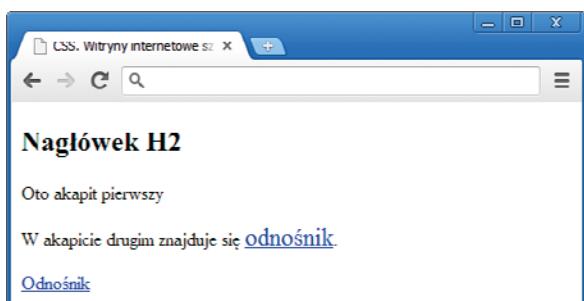
```
p * {color:red;}
```

Powyższy kod sprawia, że tekst zawarty we wszystkich znacznikach `p` zmienia kolor na czerwony.

Ciekawym sposobem na wykorzystanie tego selektora jest użycie go jako odwrotności selektora dziecka, niejako selektora „nie-dziecka”.

```
section * a {font-size:1.3em;}
```

RYSUNEK 2.13. Wybrany został odnośnik będący wnukiem; ten, który jest dzieckiem, został natomiast pominięty



Na rysunku 2.13 widać, że wybrany jest każdy znacznik `a`, który jest przynajmniej wnukiem, ale nie dzieckiem znacznika `section`. Nie jest istotne, jaki znacznik jest rodzicem elementu `a`.

W skrócie, reguła CSS z pojedynczym selektorem wybiera wszystkie znaczniki danego typu na stronie, a selektorami kontekstowymi można szczegółowo wskazać przodków i braci, jakich dany element musi mieć.

Identyfikatory i klasy



Identyfikatorom i klasom możesz nadawać dowolne nazwy, z tym że nie mogą się zaczynać od cyfr i symboli.

Posługiwianie się identyfikatorami i klasami jest odmiennym sposobem stylizowania dokumentu, przy którym nie trzeba się przejmować jego hierarchią. Klasy i identyfikatory możesz wykorzystać do bezpośredniego wybierania określonych obszarów dokumentu. W tym celu należy najpierw dodać atrybuty `id` i `class` do znaczników w kodzie HTML, a następnie odnieść się do nich selektorami CSS.

Atrybut `class`

Atrybut `class` można dodać do dowolnego elementu HTML znajdującego się w obrębie znacznika `body`. Oto fragment kodu ilustrujący wykorzystanie atrybutu HTML `class`.

```
<h1 class="specialtext">Oto nagłówek z
<span>tej samej klasy</span> co akapit drugi.</h1>
<p>Ten znacznik nie należy do żadnej klasy.</p>
<p class="specialtext">Kiedy znacznik oznaczony jest
atrybutem class, możesz go wybrać <span>niezależnie</span>
od jego położenia w hierarchii.</p>
```

Na rysunku 2.14 widać, jak to wygląda w przeglądarce.

RYSUNEK 2.14. Domyślna stylizacja kodu

Oto nagłówek z tej samej klasy co akapit drugi.

Ten znacznik nie należy do żadnej klasy.

Kiedy znacznik oznaczony jest atrybutem class, możesz go wybrać niezależnie od jego położenia w hierarchii.

Zauważ, że dwóm znacznikom nadałem atrybut `class` o wartości `specialtext`.



Zauważ, że w selektorze klasy nazwa klasy poprzedzona jest znakiem `.` (kropką). Nie stawiaj spacji między kropką

SELEKTOR KLASY

`.klasa`

Nazwę klasy elementu HTML podaje się w CSS, bezpośrednio poprzedzając jej nazwę znakiem `.` (kropką).

Nadajmy elementom następujące dwa style CSS:

```
p {font-family:helvetica, sans-serif; font-size:1.2em;}  
.specialtext {font-style:italic;}
```

Na rysunku 2.15 widać, jak wpływają na kod.

RYSUNEK 2.15. Tekst akapitu jest teraz większy i zapisany Helvetiką. Nagłówek i drugi akapit oznaczone są klasą `specialtext`, wobec czego zapisane są kursywą



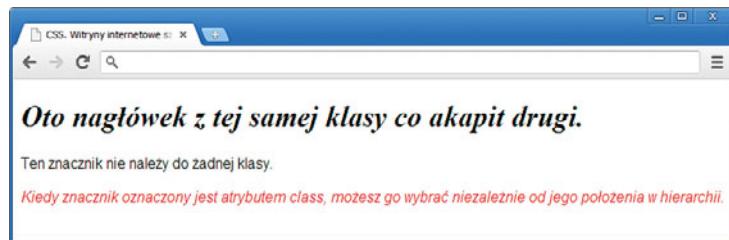
Wykorzystanie tych reguł sprawia, że obydwa akapity są zapisane krojem Helvetica (lub standardowym krojem bezszeryfowym przeglądarki, jeśli Helvetica nie jest dostępna), a akapit należący do klasy `specialtext` jest dodatkowo zapisany kursywą. Tekst w znaczniku `h1` zapisany jest domyślnym fontem przeglądarki (zwykle jest to Times), ponieważ styl z Helvetiką został nadany wyłącznie akapitom. Tym niemniej zapisany jest kursywą, ponieważ należy do klasy `specialtext`. Zauważ, że `span`, znacznik pozbawiony domyślnego stylu wizualnego, zwyczajnie odziedziczył styl swojego rodzica, ponieważ sam mu konkretnego stylu nie nadałem.

SELEKTOR ZNACZNIKA NALEŻĄCEGO DO WYBRANEJ KLASY

Jeśli chcesz wybrać jedynie akapit należący do wybranej klasy, możesz utworzyć selektor składający się z nazwy znacznika i klasy

```
p {font-family:helvetica, sans-serif; font-size:1.2em;}  
.specialtext {font-style:italic;}  
p.specialtext {color:red;}
```

RYSUNEK 2.16. Łącząc nazwę znacznika z nazwą klasy, zwiększasz szczegółowość wyboru selektora



Jak widać na **rysunku 2.16**, wyróżniony kod CSS wybiera znacznik `p` należący do klasy `specialtext`. Zestawienie w ten sposób znacznika z nazwą klasy pozwoli Ci jeszcze dokładniej wskazać znacznik, który chcesz wybrać.

Dodajmy jeszcze jedną regułę, żeby rozwinąć tę koncepcję.

```
p {font-family:helvetica, sans-serif; font-size:1.2em;}
.specialtext {font-style:italic;}
.p.specialtext {color:red;}
.p.specialtext span {font-weight:bold;}
```

Jak widzisz na **rysunku 2.17**, słowo „niezależnie” jest napisane kursywą i pogrubione, ponieważ zawarte jest w znaczniku `span`, który znajduje się w akapicie należącym do klasy `specialtext` — tak jak wskazuje reguła. Wszystkie cztery reguły wpływają na stylizację tego znacznika `span`, gdyż dziedziczy po swoim rodzicu style trzech pierwszych reguł. Dziedziczenie omówię szczegółowo w dalszej części rozdziału.

RYSUNEK 2.17. Dodając drugi selektor, możesz bardzo dokładnie określić, który znacznik ma zostać obstylowany



SELEKTOR WIELOKLASOWY

Element możesz przypisać do więcej niż jednej klasy:

```
<p class="specialtext featured">Ten znacznik span <span>może
być obstylowany</span>ale nie musi.</p>
```

Jak widzisz, nazwy klas `specialtext` i `featured` zamieszczone są w jednym cudzysłowie i oddzielone spacją. Mówiąc ściślej, atrybutowi HTML `class` można nadać więcej niż jedną wartość, oddziela-

jąc poszczególne wartości spacjami. Aby wybrać element należący do obydwu klas, należy napisać kod w rodzaju:

```
.specialtext.featured {font-size:120%;}
```

Zauważ, że pomiędzy nazwami tych dwóch klas w CSS nie ma spacji, ponieważ deklaruję, że jedynie element należący *do nich obydwu* ma być wybrany. Spacja pomiędzy nazwami wskazywałaby relację między przodkiem a potokiem w ramach selektora kontekstowego.

Często popełnianym błędem jest nadawanie znacznikom HTML kolejnych klas osobnymi atrybutami `class` zamiast nadawania kilku wartości jednemu atrybutowi, tak jak zrobiłem to w powyższym przykładzie. Praktyczne zastosowania wielu klas przedstawię w kolejnych rozdziałach.

Atrybut id

Identyfikatory są w zapisie podobne do klasy. Wskazuje się je znakiem `#` (kratką) tak samo, jak klasy znakiem `.` (kropką).

Identyfikatory a nawigacja w obrębie strony

Identyfikatorów używa się także do tworzenia odnośników nawigacyjnych działających w obrębie strony. Oto kod HTML odnośnika, który odsyła do lokalizacji na stronie, na której odnośnik sam się znajduje.

```
<a href="#bio">Biografia</a>
```

Zauważ, że znak `#` poprzedzający wartość atrybutu `href` wskazuje, iż odnośnik odsyła do lokalizacji na tej samej stronie. Przeglądarka nie wczytuje strony o takiej nazwie, co zrobiłaby, gdyby znak `#` nie był obecny.

Atrybut `href` wykorzystuje tę samą składnię `#identyfikatorów`, co w kodzie CSS, do wskazywania na stronie elementów z docelowymi atrybutami `id`. Taki element docelowy znajduje się gdzieś poniżej w kodzie strony.

```
<h3 id="bio">Biografia</h3>
```

```
<p>Urodziłem się bardzo młodym będąc...</p>
```

Zauważ, że wartość atrybutu `id` elementu docelowego nie jest poprzedzona kratką.

Po kliknięciu odnośnika strona natychmiast zostaje przewinięta tak, by u góry okna znajdował się element `h3` o identyfikatorze `bio`. Odnośnik z samym znakiem `#` podanym jako wartość `href` przewija stronę z powrotem do góry.

```
<a href="#">Do góry</a>
```

Nie trzeba nawet tworzyć elementu docelowego z identyfikatorem `#` — to po prostu działa.

Zauważ, że odnośnik nie działa, jeśli wartość atrybutu `href` jest pusta, wobec czego uznanyem zwyczajem jest wstawianie znaku `#` jako symbolu zastępczego, kiedy nie wie się jeszcze, jaki poda się adres. Programiści z zespołu często w dalszym toku pracy wstawiają w takie miejsca zmienne, np. z PHP, aby odnośniki przyjmowały adresy URL z bazy danych.

Jeśli akapit jest oznaczony identyfikatorem, tak jak tutaj:

```
<p id="specialtext">To jest specjalny tekst.</p>
```

to odpowiada mu selektor

```
#specialtext {tutaj jakieś reguły CSS}
```

lub

```
p#specialtext {tutaj jakieś reguły CSS}
```

Poza tym identyfikatory działają tak samo jak klasy i można zastosować w odniesieniu do nich (niemal) wszystko, co powiedziałem o klasach. W czym więc tkwi różnica?

Kiedy używać identyfikatorów, a kiedy klas?

Na pierwszy rzut oka może się wydawać, że klas i identyfikatorów można używać wymiennie — i jedne, i drugie są atrybutami HTML służącymi do wskazywania konkretnych znaczników w kodzie.

W rzeczywistości służą one różnym celom.

KIEDY UŻYWAĆ IDENTYFIKATORÓW



Identyfikatora możesz też użyć do powiązania kodu JavaScript ze znacznikiem (np. po to, żeby przywołać skrypt włączający animację, kiedy użytkownik najeżdża myszą na odnośnik). Identyfikatory związane z JavaScript w szczególności muszą mieć unikalne nazwy w obrębie strony, gdyż inaczej skrypt może działać nieprzewidywalnie.

Identyfikator służy do wskazywania unikalnych elementów na stronie. Z tego względu każdy zastosowany na stronie atrybut `id` musi mieć unikalną wartość (czyli nazwę). Innymi słowy, każda nazwa identyfikatora może być użyta na stronie tylko raz.

```
<nav id="mainmenu">
```

W tym przypadku na stronie nie może być innego elementu z identyfikatorem `mainmenu`. Do wskazania jakieś części kodu strony, np. menu nawigacyjnego, możesz nadać identyfikator elementowi `nav`, w którym zawarte są elementy tego menu.

```
<nav id="mainmenu">
  <ul>
    <li><a href="#">Yin</a></li>
    <li><a href="#">Yang</a></li>
  </ul>
</nav>
```

Ten unikalny identyfikator wskazuje znajdujące się na stronie menu, dzięki czemu można wybierać różne rodzaje zawartych w nim znaczników przy użyciu prostego selektora kontekstowego. Możesz na przykład nadać odnośnikom `a` w menu kolor pomarańczowy,

pomijając jednocześnie wszystkie inne odnośniki na stronie, selektorem

```
#mainmenu a {color:orange;}
```

Przy użyciu tego unikalnego identyfikatora możesz z łatwością wybrać w kodzie CSS wyłącznie ten element i jego dzieci. Zauważysz jeszcze zauważysz, że sam zwykle nadajesz identyfikator najwyższemu w hierarchii elementowi z każdej sekcji strony, aby uzyskać jednoznaczny kontekst, pozwalający mi na pisanie reguł CSS wybierających jedynie te znaczniki, które znajdują się w tych fragmentach kodu, które mnie interesują.

Skoro już wiesz, że identyfikator wskazuje unikalny element HTML na stronie, przyjrzyjmy się sposobowi, w jaki korzysta się z klas.

KIEDY UŻYWAĆ KLAS

Klasy służą do wskazywania zbioru elementów, które dzielą szereg cech, tak jak było to widać wcześniej w przykładzie z klasą `specialtext`.

W poniższej liście imion dla dzieci chcę oznaczyć imiona chłopców na niebiesko, a dziewczynek na różowo. Zaczynam od wskazania płci dzieci w znacznikach przy użyciu klas.

```
<nav>
  <ul>
    <li class="boy"><a href="#">Adam</a></li>
    <li class="girl"><a href="#">Agnieszka</a></li>
    <li class="boy"><a href="#">Andrzej</a></li>
    <li class="girl"><a href="#">Aneta</a></li>
    <li class="girl"><a href="#">Anna</a></li>
    <li class="boy"><a href="#">Antoni</a></li>
  </ul>
</nav>
```

Następnie nadajesz odnośnikom kolory w CSS

niebieski —————|.boy a {color:#6CF;}
różowy —————|.girl a {color:#F9C;}

Pierwsza reguła wybiera wszystkie elementy `a` z przodkiem należącym do klasy `boy`, a druga wszystkie elementy `a` z przodkiem należącym do klasy `girl`. W obydwu przypadkach przodkami są rodzice odnośników — elementy `li`.

Nie szalej z klasami

Unikaj tego, co guru internetowy Jeffrey Zeldman nazywa „wysypką klasową”, polegającą na nadawaniu unikalnych klas praktycznie wszystkim znacznikom w kodzie i pisaniu osobnych reguł dla każdej z nich. Jeżeli już nałogowo nadajesz każdemu znacznikowi klasę — co robi większość ludzi, którzy z radością zabierają się za pracę z CSS bez znajomości zasad, jakimi rządzi się dziedziczenie i działanie selektorów kontekstowych — to prawdopodobnie tworzysz powtarzające się style dla różnych znaczników (np. przypisując ten sam font wszystkim znacznikom na stronie). Dziedziczenie i selektory kontekstowe pozwalają na objęcie stylami wielu znaczników, a tym samym zminimalizowanie ilości kodu CSS, który trzeba napisać i obsługiwać.

Identyfikatory i klasy — podsumowanie

Identyfikator służy do wskazywania konkretnego, unikalnego elementu w kodzie strony. Zapewnia on kontekst, który pozwala na pisanie reguł CSS ignorujących resztę kodu, a wybierających wyłącznie znaczniki objęte tym kontekstem.

Klasy są z kolei zbiorczymi identyfikatorami, które można przypisać dowolnej liczbie elementów HTML na wielu stronach, by nadać im te same reguły CSS. Klasa umożliwia również nadawanie tych samych reguł stylistycznych różnego rodzaju znacznikom.

Selektory atrybutów

Pokazałem Ci już, jak używać selektorów kontekstowych, identyfikatorów oraz klas do wybierania znaczników HTML w kodzie CSS. Trzecia metoda wybierania opiera się na selektorach atrybutów. Selektory atrybutów wybierają elementy na podstawie atrybutów znaczników HTML. Poniżej znajdziesz dwa użyteczne przykłady.

Selektor nazwy atrybutu

nazwaZnacznika[nazwaAtrybutu]

Wybiera wszystkie znaczniki typu *nazwaZnacznika*, które są oznaczone atrybutem *nazwaAtrybutu*.

Poniższy kod CSS

```
img[title] {border:2px solid blue;}  
sprawia, że każdy znacznik img z atrybutem title, taki jak ten  

```

jest wyświetlany w kolorze niebieskim i otoczony dwupikselowym obramowaniem. Wartość atrybutu title nie ma tutaj znaczenia — chodzi tylko o to, żeby ten atrybut był obecny. Takiego stylu możesz użyć, by wskazać użytkownikowi, że może najechać kursorem na obraz, żeby wyświetlić chmurkę podpowiedzi (czyli tekst generowany przez atrybut title). Powszechnie przyjęło się nadawanie jednakowych wartości tekstowych atrybutom alt i title. Tekst atrybutu alt wyświetla się, kiedy nie można wczytać obrazu, a także może być odczytany przez czytnik ekranowy, podczas gdy atrybut title wyświetla chmurkę podpowiedzi po najechaniu kursorem na obraz.

Selektor wartości atrybutu



W HTML5 wartości atrybutów nie trzeba ujmować w cudzysłach. Sam dodaje je w celu zachowania przejrzystości.

nazwaZnacznika[nazwaAtrybutu="wartośćAtrybutu"]

Taki selektor wybiera wszystkie znaczniki opatrzone atrybutem nazwaAtrybutu o wartości wartośćAtrybutu.

Selektor ten pozwala Ci dokładnie wskazać wartość atrybutu. Przykładowa reguła

```
img[title="red flower"] {border:4px solid green;}
```

nadaje obrazowi obramowanie, jeśli atrybut title obrazu ma wartość red flower, czyli jeżeli znacznik wygląda następująco:

```

```

Pełną listę selektorów atrybutów znajdziesz na stronie <http://www.kurshtml.edu.pl/css/selektory.html>.

Selektory atrybutów — podsumowanie

Wybieranie znaczników według nazw atrybutów i innych właściwości atrybutów elementów daje Ci możliwość wskazywania konkretnych znaczników jednego typu. Planując z wyprzedzeniem, możesz napisać kod, z którego możliwe będzie wybieranie znaczników przy użyciu selektorów atrybutów.

Wszystkie selektory, z którymi się dotąd zetknąłeś, łączy to, że odnoszą się do jakichś elementów kodu — nazw znaczników, klas, identyfikatorów, atrybutów bądź wartości atrybutów. CSS możesz także wykorzystać do obstylowywania elementów w odpowiedzi na różne zdarzenia, np. najechanie kursorem na odnośnik (co nazywamy efektem rollover). Robi się to przy użyciu pseudoklas.

Pseudoklasy

Pseudoklasy, nazywane tak, ponieważ działają jak selektory klas, po-mimo że owe klasy w rzeczywistości wcale nie występują w kodzie HTML, dzielą się na dwie grupy:

- **Pseudoklasy interfejsu** sprawiają, że reguły są nadawane elementom HTML, kiedy te znajdują się w określonym stanie, np. kiedy kurSOR znajduje się nad odnośnikiem.
- **Pseudoklasy strukturalne** sprawiają, że reguły są nadawane elementom HTML, kiedy w kodzie występują pewne relacje strukturalne, np. kiedy jakiś element jest pierwszym lub ostatnim w zbiorze powiązanych elementów.

Pseudoklasy interfejsu

Pseudoklasy interfejsu nadawane są w odniesieniu do stanu, w jakim znajduje się dany element HTML. Najczęściej używa się ich z odnośnikami (czyli znacznikami), co pozwala na zmianę ich obstylowania — np. zmianę koloru lub usunięcie podkreślenia — gdy najeżdża się na nie kursorem. Można ich jednak używać także w celu osiągnięcia różnych innych reakcji, jak np. wyświetlenia panelu informacyjnego po najechaniu kursem na element. Pokażę to jeszcze w rozdziale o komponentach interfejsu.

PSEUDOKLASY ODNOŚNIKÓW

Istnieją cztery pseudoklasy interfejsu odnośników, ponieważ odnośniki zawsze znajdują się w jednym z czterech stanów:

- **Link.** Odnośnik widnieje na stronie i czeka, aż ktoś go kliknie.
- **Visited.** Użytkownik już wcześniej kliknął odnośnik.
- **Hover.** Nad odnośnikiem znajduje się kurSOR.
- **Active.** Odnośnik jest właśnie klikany (tj. przycisk myszy jest naciśnięty, ale nie został jeszcze puszczyony).



Ponieważ te cztery pseudoklasy odznaczają się jednakową precyzją (które to pojęcie omówię w dalszej części rozdziału), przeglądarki mogą wyświetlać inne rezultaty od zamierzonych, jeśli te cztery selektory nie zostaną wymienione w podanej tutaj kolejności.



Pojedynczy dwukropka (:) wskazuje pseudoklasy, ale nowe pseudoelementy wprowadzone w CSS3 wymagają użycia podwójnego dwukropka (::). Choć przeglądarki obecnie obsługują pojedyncze dwukropki używane przy pseudoelementach z CSS1 i CSS2, powinieneś przyzwyczaić się do korzystania z dwukropków podwójnych, jako że pojedyncze mogą zostać wycofane z czasem. Więcej na ten temat przeczytasz na stronie <http://www.w3.org/TR/2005/WD-css3-selectors-20051215/#pseudo-elements>.

Oto selektory pseudoklas tych stanów, dołączone do selektora a i kilku przykładowych deklaracji:

```
a:link {color:black;}  
a:visited {color:gray;}  
a:hover {text-decoration:none;}  
a:active {color:red;}
```

Charakterystyczny znak : (dwukropka) wskazuje, że mamy do czynienia z selektorem pseudoklasy.

Według powyższych deklaracji odnośniki są wyjściowo czarne i domyślnie podkreślone. Po najechaniu na odnośnik kursorem podkreślenie znika, ale odnośnik wciąż jest czarny, ponieważ nie zdefiniowano innego koloru dla stanu `hover`. Kiedy użytkownik naciska klawisz myszy, odnośnik staje się aktywny i zmienia kolor na czerwony. Po kliknięciu odnośnika, czyli naciśnięciu iпусzczeniu klawisza myszy nad jednym elementem, co jednocześnie przywołuje jego adres URL, odnośnik już zawsze (a właściwie do wygaśnięcia lub usunięcia historii przeglądania) będzie szary.

Nie musisz definiować wszystkich tych stanów. Jeżeli chcesz jedynie zdefiniować styl odnośnika i jego stan `hover`, to nie ma problemu — zresztą często jest to bardzo sensowne rozwiązanie. Jeżeli masz na przykład długi spis odnośników, to bardzo przydatne jest wskazanie, np. jaśniejszym kolorem, które z nich zostały odwiedzone czy też kliknięte. Zmiana koloru klikniętego odnośnika nie ma jednak sensu, gdy jest to odnośnik w pasku nawigacyjnym.

Sam zazwyczaj definiuję stan a i `:hover` — ten ostatni po to, żeby użytkownik po najechaniu na element widział, że można go kliknąć. Możliwość stylizowania stanów odnośnika jest bardzo przyjemna, ale prawdziwe możliwości tych pseudoklas odnośników uwidaczniają się, kiedy używa się ich w ramach selektorów kontekstowych. Możesz dzięki nim przypisać różne style graficzne i zachowania różnym grupom odnośników. Łatwo na przykład zróżnicować wygląd i zachowanie odnośników znajdujących się w elementach `nav`, `footer`, `sidebar` i `article`, o czym jeszcze wspomnę.

Zauważ, że tych pseudoklas możesz używać z dowolnymi elementami, a nie tylko a, do tworzenia różnorakich efektów rollover.

```
p:hover {background-color:gray;}
```



W tym przykładzie i kolejnych e reprezentuje dowolny element, np. p, h1 albo section.

PSEUDOKLASA :FOCUS

e: focus

Elementy takie jak pola tekstowe formularza stają się aktywne po kliknięciu i to w nich pojawiają się wtedy wpisywane przez użytkownika dane. Kod

```
input:focus {border:1px solid blue;}
```

wobec tego zamieszcza niebieskie obramowanie wokół takiego pola, kiedy kurSOR jest w nim aktywny. Taki efekt informuje użytkownika, gdzie znajdą się dane, które wpisze.

PSEUDOKLASA :TARGET

e: target

Kiedy użytkownik kliką odnośnik prowadzący do jakiegoś znajdującego się na stronie elementu, ów element staje się celem, który można wybrać przy użyciu pseudoklasy :target.

W przypadku poniższego odnośnika

```
<a href="#more_info">Więcej informacji</a>
```

celem jest znajdujący się gdzieś na stronie element z identyfikatorem more_info. Może on wyglądać tak:

```
<h2 id="more_info">To tych informacji szukasz.</h2>
```

Poniższa reguła CSS

```
#more_info:target {background:#eee;}
```

nadaje elementowi z identyfikatorem more_info szare tło po kliknięciu przez użytkownika odnośnika, który do niego prowadzi.

W Wikipedii pseudoselektory :target są standardowo używane z przypisami. Odnośniki do przypisów w Wikipedii są niepozornymi liczbami przedstawionymi w tekście jako linki. Same przypisy wchodzą natomiast w skład długiej listy pod artykułem. Bez podświetlenia zapewnionego pseudoselektorem :target czytelnik nie wiedziałby, który konkretnie przypis odnosi się do klikniętego odnośnika.

Dodatkowe pseudoklasy interfejsu znajdziesz na stronie <http://www.kurshtml.edu.pl/css/selektory.html>.

Pseudoklasy strukturalne

Przy użyciu pseudoklas strukturalnych można nadawać style na podstawie struktury kodu, np. wskazując rodzica danego elementu lub poprzedzającego go brata.

:FIRST-CHILD I :LAST-CHILD

`e:first-child`

`e:last-child`

`:first-child` jest pierwszym elementem ze zbioru braci, a `:last-child` ostatnim. Gdyby regułę

```
ol.results li:first-child {color:blue;}
```

zastosowano do tego kodu

```
<ol class="results">
```

```
  <li>Mój Pręzny Kucyk</li>
```

```
  <li>Wierny Rumak</li>
```

```
  <li>Stara Chabeta</li>
```

```
</ol>
```

to tekst „Mój Pręzny Kucyk” zostałby wyświetlony na niebiesko. Z kolei selektor

```
ol.results li:last-child {color:red;}
```

zaznaczyłby tekst „Stara Chabeta” na czerwono.

:NTH-CHILD

`e` oznacza tu nazwę elementu, a `n` liczbę (można też użyć słów kluczowych `odd` i `even`)

—| `e:nth-child(n)`

Przykładowo,

```
li:nth-child(3)
```

wybiera co trzeci element ze zbioru elementów listy.

`nth-child` bardzo często wykorzystuje się do zwiększenia czytelności tabel poprzez nadawanie ich rzędom przemiennych kolorów tła, co omówię w rozdziale 6.

Istnieją jeszcze inne pseudoklasy strukturalne, których pełną listę znajdziesz na stronie <http://www.kurshtml.edu.pl/css/selektory.html>.

Pseudoelementy



W celu stworzenia efektu inicjału bez pseudoelementu

musiałbyś zatrzymać pierwszą literę akapitu w elemencie `span`, a następnie go obstyloować.

Przedstawiony tutaj pseudoelement pozwala zamiast tego na sprawne wstawienie takiego „niewidocznego” kodu.

RYSUNEK 2.18. Pseudoelement `::first-letter` umożliwia tworzenie inicjałów

Pseudoelementy, jak sama nazwa wskazuje, wywołują taki efekt, jakby dodatkowe elementy cudownie pojawiały się w kodzie dokumentu. Poniżej znajdują się przykłady ich zastosowania.

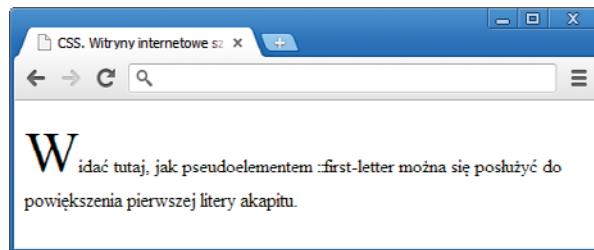
PSEUDOELEMENT :FIRST-LETTER

`e::first-letter`

Poniższy kod CSS

```
p::first-letter {font-size:300%;}
```

pozwala na powiększenie otwierającej akapit litery, tak jak widać to na rysunku 2.18.



PSEUDOELEMENT ::FIRST-LINE

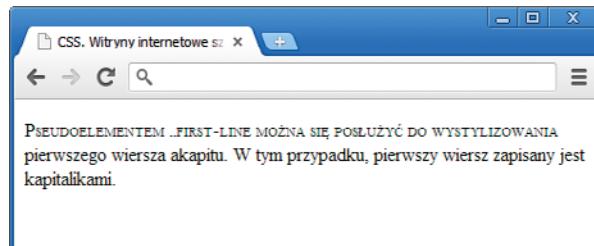
`e::first-line`

pozwala na obstylowanie pierwszego wiersza ciągu tekstu, przeważnie akapitu. Zastosowanie kodu

```
p::first-line {font-variant:small-caps;}
```

sprawia, że tekst w pierwszym wierszu wyświetlany jest kapitalikami (rysunek 2.19).

RYSUNEK 2.19. Pseudoelement `::first-line` posłużył do zapisania pierwszego wiersza kapitalikami. Zauważ, że zakres tekstu objętego selektorem `::first-line` zmienia się wraz ze zmianą wielkości okna przeglądarki, a co za tym idzie długości wiersza



PSEUDOELEMENTY ::BEFORE I ::AFTER

Te dwa pseudoelementy

`e::before` i `e::after`

dodają podaną treść przed elementem lub po nim. Kod

```
<p class="age">25</p>
```

wraz ze stylami

```
p.age::before {content: "Wiek: " ;}
```

```
p.age::after {content: " lat." ;}
```

tworzy tekst o treści:

Wiek: 25 lat.

Ta technika przydaje się najbardziej, kiedy zawartość znacznika generowana jest na podstawie bazy danych. Jeżeli rezultatem jest sama liczba, to podane tu selektory pozwalają na opatrzenie danych opisami, gdy pokazuje się je użytkownikowi.

Powyżej widać podstawowe, choć użyteczne zastosowanie pseudoelementów `::before` i `::after`. Dalej jednak przedstawię, jak można je wykorzystać przy dołączaniu nowych elementów do wybranych znaczników w celu uzyskania ciekawych możliwości stylizacji.

Przegląd różnych rodzajów selektorów CSS dobiegł tym samym końca. Czas teraz przyjrzeć się bliżej temu, jak działa CSS.

W dużym arkuszu stylów dowolna liczba reguł może wybierać i definiować jedną właściwość danego elementu. Przykładowo, akapitowi z atrybutem `class` reguła używająca nazwy znacznika może przypisywać jakiś font, podczas gdy inna reguła, wykorzystująca nazwę jego klasy, przypisuje mu inny font. Rzecz jasna, właściwość określająca font może mieć w danej chwili tylko jedną wartość. CSS posługuje się trzema powiązanymi mechanizmami, służącymi do rozwiązywania takich konfliktów i określania, które reguły „wygrywają” w walce o określenie właściwości. Te mechanizmy to dziedziczenie, kaskadowość i precyzja, którym się teraz kolejno przyjrzymy.



Zwróc uwagę, że w wartościach deklaracji podano spację, żeby w tekście końcowym znalazły się odpowiednie odstępy.



Wyszukiwarki nie odczytują treści pseudoelementów, ponieważ nie znajdują się one w kodzie HTML. Nie używaj ich zatem do dodawania ważnych treści, które chcesz, żeby były indeksowane.

Dziedziczenie

Podobnie jak dziedziczenie majątku, dziedziczenie w CSS wiąże się z przekazywaniem czegoś potomkom przez przodków — wartości właściwości CSS. Z mojego omówienia hierarchii dokumentu w rozdziale 1. pamiętasz, że znacznik `body` jest praprzodkiem wszystkich znaczników w kodzie. Jeśli znacznikowi `body` nadasz taki styl:

```
body {font-family:helvetica, arial, sans-serif;}
```

to, za sprawą dziedziczenia CSS, każdy element tekstowy w dokumencie odziedziczy go i będzie wyświetlany krojem Helvetica (lub któryms z innych wymienionych, jeżeli Helvetica nie będzie dostępna). Użyteczność dziedziczenia jest jasna — zamiast podawać tę samą wartość `font-family` dla każdego znacznika w kodzie, wystarczy określić wybrany font na szczytcie hierarchii jako główny dla całego dokumentu. Następnie wystarczy dodawać kolejne właściwości `font-family` do elementów, które trzeba oznaczyć innymi krojami.

Wiele właściwości CSS, w tym te, które odnoszą się do tekstu — takie jak kolor, font i wielkość — dziedziczonych jest właśnie w ten sposób. Tymczasem wiele innych właściwości nie dziedziczy się, ponieważ nie miałoby to sensu w ich wypadku. Takie niedziedziczone właściwości przede wszystkim odnoszą się do położenia i sposobu wyświetlania elementów pól, takich jak obramowania, marginesy i dopełnienia, o których poczytasz więcej w kolejnym rozdziale.

Wyobraź sobie, że chcesz utworzyć pasek boczny z listą odnośników. Możesz to zrobić, tworząc element `nav` z odnośnikami jako dziećmi i nadając mu styl obejmujący wielkość pisma oraz obramowania, np. czerwonej linii grubej na dwa piksele. Choć to, że każdy odnośnik odziedziczy wielkość pisma elementu `nav`, jest bardzo wygodne, to nie ma sensu, żeby wszystkie odziedziczyły również jego czerwone obramowanie. I tak się wcale nie stanie, ponieważ właściwości obramowania nie są dziedziczone.

Jako że właściwości kroju i stylizacji tekstu są dziedziczone, musisz być bardzo uważny w pracy ze względnymi wielkościami tekstu w rodzaju procentów i em. Jeśli nadasz wartość 80% wielkości tekstu znacznika, którego wielkość tekstu również wynosi 80%, to wielkość

tekstu w pierwszym znaczniku będzie wynosić 64% (czyli $80\% \times 80\%$), co może nie być wartością, na której Ci konkretnie zależy. W rozdziale 4. omówię wady i zalety określania wielkości tekstu wartościami absolutnymi i względnymi.

Dziedziczenie w działaniu przedstawię jeszcze w dalszych przykładach i pokaż Ci, jak wykorzystywać dziedziczenie stylów do zmniejszenia ilości kodu CSS, który trzeba napisać w celu uzyskania pożądanego rezultatu.

Kaskadowość

Wiesz już wystarczająco dużo, bym mógł Ci powiedzieć coś sensowego o kaskadowości. Kaskadowość polega na tym, że style „spływają” z wyższych poziomów hierarchii dokumentu do niższych, a służy ona temu, by przeglądarka mogła zdecydować, z którego ze źródeł wartości właściwości danego znacznika należy skorzystać.

Kaskadowość to potężny mechanizm. Zrozumienie jej pomoże Ci w pisaniu kodu CSS w możliwie zwięzły i łatwy do modyfikowania sposób, umożliwiając Ci tworzenie dokumentów wyglądających tak, jak powinny, a jednocześnie pozostawiających użytkownikowi pewną ilość kontroli nad elementami graficznymi, takimi jak wielkość tekstu.

Źródła stylów

Style pochodzą z różnych źródeł. Przede wszystkim, przeglądarka ma własny, domyślny arkusz stylów, ponieważ każdy znacznik jest jakoś wyświetlany, nawet jeśli nie nada mu się stylu. Znaczniki **h1** wyświetlane są wielkim, pogrubionym tekstem, znaczniki **em** kursywą, a listy są wcięte i mają przypisane punktory bądź liczby porządkowe.

Jest też arkusz stylów użytkownika. Użytkownik również może stworzyć swój arkusz, choć niewielu to robi. Przydaje się to osobom niedowidzącym, gdyż można w ten sposób zwiększyć ogólną wielkość tekstu na wszystkich wczytywanych stronach lub zmienić jego kolor tak, by zwiększyć jego czytelność. Niedowidzący użytkownik może utworzyć styl w rodzaju

```
body {font-size:200%}
```

który podwaja wielkość każdego tekstu. Tu znowu widzimy dziedziczenie w działaniu.



Ramkę „Trzy sposoby
dołączania stylów
do dokumentu”
znajdziesz we wcześniejszej części
tego rozdziału.

Wreszcie są też autorskie arkusze stylów, które piszesz Ty sam, autor strony. Omówiłem już źródła, z jakich pobiera się style autora: zewnętrzne arkusze stylów, style osadzone w stronie oraz dołączane do znaczników style lokalne.

Oto kolejność, w jakiej przeglądarka przegląda źródła w poszukiwaniu stylów:

- domyślny arkusz przeglądarki,
- arkuszów użytkownika,
- autorskie arkusze stylów (w kolejności, w jakiej są załączone na stronie),
- osadzone style autorskie,
- lokalne style autorskie.

Przeglądarka aktualizuje wartości właściwości kolejnych znaczników (o ile takie zostały zdefiniowane), które napotyka, gdy przegląda kolejno reguły z każdej lokalizacji. Wartości właściwości elementów definiowane wraz z końcem procesu określają to, jak będą one wyglądały na ekranie.

Jeśli na przykład autorski arkusz stylów nadaje właściwości `font-family` elementu `p` wartość `Helvetica`, ale osadzony na stronie styl używa tego samego selektora do wybrania wartości `Verdana`, akapit będzie wyświetlany krojem Verdana. Osadzone style odczytywane są po autorskim arkuszu stylów. Tymczasem, jeśli akapitem nie nadano żadnego kroju w arkuszu użytkownika lub autorskim, to są one wyświetlane krojem Times New Roman, ponieważ to on jest podany we wszystkich domyślnych arkuszach stylów przeglądarki.

Wszystko to powinno stać się bardziej zrozumiałe, gdy poznasz zasady kaskadowości, decydujące o tym, które style mają być nadawane elementom strony.



Więcej informacji o kaskadowości znajdziesz na stronie W3C (www.w3.org/TR/CSS2/cascade.html).

Zasady kaskadowości

Oto zasady określające działanie kaskadowości:

Zasada 1: Znajdź wszystkie deklaracje, odnoszące się do wszystkich elementów i właściwości. Wczytując stronę, przeglądarka sprawdza wszystkie reguły CSS i określa, na które elementy HTML mają one wpływ.

Zasada 2: Uporządkuj według kolejności i wagi. Przeglądarka kolejno sprawdza wszystkie pięć źródeł, ustawiając przy tym wszelkie podane i skojarzone właściwości, na jakie trafia. Jeśli jakaś znaleziona właściwość jest ponownie zdefiniowana w dalszej części sekwencji, przeglądarka aktualizuje jej wartość. W razie konieczności powtarza tę czynność, aż do sprawdzenia wszystkich możliwych lokalizacji właściwości każdego znacznika. Wartość danej właściwości u końca tego procesu określa, jak element jest wyświetlany.

Pod uwagę bierze się także wagę deklaracji. Zwiększenie wagi deklaracji jest możliwe poprzez wskazanie, że jest ważna:

```
p {color:green !important; font-size:12pt;}
```

Właściwość `!important` oddzielona jest spacją od stylu, któremu chcesz przydać wagi, ale poprzedza separator ; (średnik).

Ten styl wskazuje, że określony jako zielony kolor tekstu jest ważny, wobec czego tekst zawsze będzie wyświetlany na zielono, nawet jeśli dalej w ramach kaskady określony jest inaczej. To tak, jakbyś powiedział: „Ten styl musi być wykorzystywany bez względu na okoliczności”. Do wymuszania zastosowania danego stylu regułą `!important` musisz podchodzić ostrożnie, ponieważ może mieć to wpływ na czyjsz osobisty arkusz stylów, który ustawiony jest w dany sposób z bardzo ważnego powodu. Jeśli uważasz, że musisz z niego skorzystać, najpierw bardzo skrupulatnie przejrzyj swój kod CSS – prawdopodobnie dojdzieś do wniosku, że możesz znaleźć lepsze rozwiązanie. Sam bardzo rzadko używam deklaracji `!important`, prawie w ogóle.

Zasada 3: Uporządkuj według precyzji. Precyzja decyduje o szczególowości reguły. Gdyby czynnik precyzji nie istniał, byłbyś zmuszony ciągle zmieniać kolejność stylów w arkuszu, żeby dopilnować, by stosowany był ten właściwy.

Jak wiesz, jeśli w arkuszu stylów znajduje się reguła

```
p {font-size:12px;}  
i  
p.targettext {font-size:16px;}
```

to tekst

```
<p class="largetext">Trochę tekstu</p>
```

będzie wyświetlany w rozmiarze 16 pikseli, ponieważ selektor drugiej reguły podaje zarówno nazwę znacznika, jak i klasy. Tym samym jest bardziej precyzyjny i przesłania prostszą regułę. Powyższy przykład może się wydawać oczywisty, ale zastanówmy się, co się stanie z tym kodem, jeśli użyje się następujących stylów:

```
p {font-size:12px;}  
.largetext {font-size:16px;}
```

Otocz chociaż obydwie reguły odnoszą się do znacznika, pierwszeństwo ma klasa, więc tekst wyświetlany jest w rozmiarze 16 pikseli. Jest tak, ponieważ selektory klas są bardziej precyzyjne od prostych selektorów znaczników. Precyzyjność zależy od liczby znaczników, klas i identyfikatorów w selektorze.

Obliczanie precyzji

Poniżej znajduje się sposób na obliczenie precyzji dowolnego selektora. Istnieje prosty system obliczania, który oparty jest na wzorze „ICE” o trzech wartościach:



ICE w rzeczywistości nie podaje trzycyfrowej liczby. Po prostu w większości przypadków odczytywanie wyniku w ten sposób sprawdza się. Najwyższa liczba zwycięża. Musisz tylko wiedzieć, że nawet jeśli otrzymasz wartość w stylu 0-1-12, to 0-2-0 i tak będzie bardziej precyzyjna.

I – C – E

Kreski w tym wzorze nie są znakami odejmowania, tylko separatormi. Działa on następująco:

1. Dodaj jeden do wartości I za każdy identyfikator w selektorze.
2. Dodaj jeden do wartości C za każdą klasę w selektorze.
3. Dodaj jeden do wartości E za każdy element (znacznik) w selektorze.
4. Odczytaj wynik jako trzycyfrową liczbę.

Przyjrzymy się zatem precyzji poniższych przykładów

0 - 0 - 1 precyza=1 | p

0 - 1 - 1 precyza=11 | p.largetext

1 - 0 - 1 precyza=101 | p#largetext

1 - 0 - 2 precyza=102 | body p#largetext

1 - 1 - 3 precyza=113 | body p#largetext ul.mylist

1 - 1 - 4 precyza=114 | body p#largetext ul.mylist li

Precyza ka dego kolejnego przyk adu jest wi ksza od poprzedniego.

Zasada 4: Upor dakuj wedle kolejno ci. Kiedy dwie regu y wp ywaj  na t  sam  w a ciwo c  elementu oraz odznaczaj  si  tak  sam  precyzj , pierwsze stwo ma ta, kt ra znajduje si  w dalszej cz esci kaskady.

Oto jak, drogi Czytelniku, przedstawia si  kaskadowo c . Owszem, nie艣ta艣o j  zrozumie , zw aszcza je si nie ma si  wi kszego do wiadczenia z CSS, ale moja uproszczona wersja zasad kaskadowo ci (opisana w „Podsumowaniu uproszczonych zasad kaskadowo ci”) sprawdza si  niemal zawsze, a jest o wiele łatwiejsza.

Podsumowanie uproszczonych zasad kaskadowo ci

S t r y uproszczone zasady kaskadowo ci i znajduj  one zastosowanie niemal w ka dej sytuacji.

Zasada 1: Selektory z identyfikatorami przes aniaj  selektory z klasami, kt re z kolei przes aniaj  selektory z nazwami znacznik w.

Zasada 2: Je si ta sama w a ciwo c  dla tego samego znacznika zdefiniowana jest w ponad jednej lokalizacji, style lokalne przes aniaj  style osadzone, a te przes aniaj  arkusze styl w. W arkuszach styl w zdefiniowane p o ej style przes aniaj  wcze sniejsze style o takiej samej precyzji.

Zasada 2. ust puje jednak zasadzie 1. — bardziej precyzyjny selektor przes ania wszystko inne.

Zasada 3: Zdefiniowane style przes aniaj  style dziedziczone, niezale nje od precyzji. Zasada ta wymaga pewnego obja nienia. Poni szy kod

```
<div id="cascade_demo">  
  <p id="inheritance_fact">Dziedziczenie w obr bie kaskady jest <em>s abe</em> </p>  
</div>
```

w po czeniu z regu  

```
div#cascade_demo p#inheritance_fact {color:blue;}
```

2 - 0 - 2 (du a precyzja)

s prawia,  e ca y tekst, w tym wyraz „s abe”, wyświetlany jest na niebiesko, poniewa  znacznik **em** dziedziczy kolor swojego rodzica — znacznika **p**.

Wystarczy jednak nada  znacznikowi **em** regu  

```
em {color:red;}
```

0 - 0 - 1 (ma a precyzja)

zeby tekst **em** wyświetlany by  na czerwono, pomimo  e precyzja tej regu y (0 - 0 - 1) nie mog aby by  mniejsza. Odziedziczony styl **em** zostaje przes aniety przez zdefiniowany styl **em**, pomimo  e odziedziczona regu a jest o wiele precyzyjniesza.

Deklaracje reguł

Do tej pory koncentrowałem się na wykorzystaniu selektorów reguł CSS do wybierania znaczników, ale nie omówiłem jeszcze drugiej części reguły CSS — deklaracji. Podałem wiele różnych deklaracji przy omawianiu przykładów selektorów, ale objaśniłem ich znanie w ograniczonym zakresie. Czas przyjrzeć się deklaracjom szczegółowo.

Podany wcześniej diagram opisujący strukturę reguły CSS ([rysунek 2.2](#)) pokazuje, że deklaracja składa się z dwóch części: właściwości i wartości. Właściwość określa, na jaki aspekt elementu reguły ma wpływać (kolor, wysokość itp.), a wartość z kolei ustawia właściwość (jako zieloną, 12 px itp.).

Każdemu elementowi przypisany jest szereg właściwości, które można ustawić przy użyciu kodu CSS; różnią się one z elementu na element. Właściwość `font-size` można ustawić dla tekstu, ale nie dla obrazu. Poszczególnym właściwościom różnych elementów HTML przyjrzymy się w kolejnych rozdziałach, ale wartości CSS omówię już teraz, ponieważ istnieje zaledwie kilka ich rodzajów.

Wartości należą do trzech głównych typów:

Wartości słowne. W deklaracji `font-weight:bold, bold` jest wartością słowną. Wartości słowne nazywamy również słowami kluczowymi.

Wartości liczbowe. Do wartości liczbowych dołączane są jednostki, takie jak cale i punkty. W deklaracji `font-size:12px, 12` jest wartością liczbową, a `px` jednostką — pikselem. Zauważ, że nie trzeba podawać jednostki, jeśli wartość wynosi 0.

Wartości kolorów. Takie wartości można zapisywać w różnych formatach: RGB (czerwony, zielony, niebieski), HSL (barwa, nasycenie, jasność) oraz szesnastkowym (np. `color:#336699`).

Przyjrzyjmy się bliżej tym trzem rodzajom wartości właściwości.

Wartości słowne

Słowami definiuje się różnorakie wartości właściwości CSS.

Właściwość `visibility` na przykład posługuje się wartościami `visible` i `hidden`; `border-style` posługuje się m.in. wartościami `solid`, `dashed` i `inset`.

Poza paroma takimi przykładami trudno podać cokolwiek więcej o wartościach słownych, co by miało sens przed omówieniem wartości w dalszych rozdziałach, jako że używane słowa odnoszą się do konkretnych właściwości. Wartości liczbowe i kolorów można jednak wyrazić na pewne określone sposoby.

Wartości liczbowe

Wartościami liczbowymi określa się wielkość, przy czym w CSS ów termin odnosi się ogólnie do wysokości, szerokości, grubości różnych elementów. Te wartości można podzielić na bezwzględne (absolutne) i względne.

Wartości bezwzględne (**tabela 2.1**) odnoszą się do rzeczywistych wielkości (np. 6 centymetrów), a wartości względne do relacji między dwiema rzeczami, które można zmierzyć (np. „dwa razy dłuższe niż...”).

TABELA 2.1. Przykłady wartości bezwzględnych

WARTOŚĆ BEZWZGLĘDNA	JEDNOSTKA	PRZYKŁAD*
Cale	in	height:6in
Centymetry	cm	height:40cm
Milimetry	mm	height:500mm
Punkty	pt	height:60pt
Pica	pc	height:90pc
Piksele	px	height:72px

*Nie podano tu równych wielkości.

W tej książce i w mojej pracy piksele są jedyną jednostką absolutną, z jakiej korzystam, chyba że tworzę arkusze stylów do druku — ponieważ wielkość papieru mierzy się w calach, sensownym rozwiązaniami jest projektowanie układów graficznych przy użyciu tychże jednostek.

Sensu jednostek bezwzględnych nie trzeba tłumaczyć, natomiast jednostki względne (**tabela 2.2**) wymagają pewnego omówienia.

TABELA 2.2. Przykłady wartości względnych

WARTOŚĆ WZGLĘDNA	JEDNOSTKA	PRZYKŁAD*
Em	em	height:1.2em
Ex	ex	height:6ex
Procent	%	height:120%

*Nie podano tu równych wielkości.

Em i ex są jednostkami miary wielkości tekstu, choć w CSS można je odnieść do dowolnej właściwości elementu, takiej jak wielkość. Em wywodzi się z szerokości litery *M* w kroju, toteż jego wielkość jest inna w zależności od zastosowanego fonta. Ex jest odpowiednikiem wysokości małej litery *x* w foncie, czyli środkowej części liter, bez wydłużen górnych i dolnych, które posiadają np. *d* i *p*.

Wartości procentowe przydają się do określania szerokości elementów zagnieżdżonych, takich jak znaczniki `div`, względem szerokości kontenera. Wykorzystanie strukturalnych elementów HTML, których szerokości określone są wartościami procentowymi względem elementu `body`, jest podstawą „ płynnych ” layoutów, które zmieniają się proporcjonalnie wraz ze zmianą wielkości okna przeglądarki, o czym przekonasz się w rozdziale 5.

Wartości kolorów

Kolory można określić kilkoma różnymi rodzajami wartości. Jeśli chcesz, możesz zamieszcać różnego rodzaju wartości w obrębie jednego arkusza.

NAZWA KOLORU (NP. RED)

Jak widziałeś we wszystkich wcześniejszych przykładach kolorów w omówieniach selektorów, kolor możesz określić nazwą, czy też — używając oficjalnego terminu — słowem kluczowym.

W3C podaje szesnaście słów kluczowych na określenie kolorów: `aqua`, `black`, `blue`, `fuchsia`, `gray`, `green`, `lime`, `maroon`, `navy`, `olive`, `purple`, `red`, `silver`, `teal`, `white` i `yellow`. Na stronie <http://www.w3.org/TR/css3-color/#html4> znajdziesz listę nazw i odpowiadających im wartości kolorów sRGB.

Większość współczesnych przeglądarek obsługuje dużo więcej barw (140 nazw kolorów X11), ale jeśli chcesz definiować kolory nazwami, to możesz polegać jedynie na 16, które są podane wyżej. Na stronie http://en.wikipedia.org/wiki/X11_color_names znajdziesz listę nazw i odpowiadających im wartości kolorystyczne RGB.

Ogólnie rzecz biorąc, słowa kluczowe nazw barw przydają się do określania czerni i bieli, ale do poważnej pracy z kolorami musisz korzystać z formatów, które opisuję poniżej.

Szesnastkowy (#RRGGBB i #RGB)

 Niewiele wartości szesnastkowych da się odgadnąć na pierwszy rzut oka. Np. `#7ca9be` to ciemny odcień koloru niebieskozielonego, ale po czym to poznać? Przyjrzymy się pierwszym cyfrom każdej pary `rgb`, czyli w tym wypadku 7, a i b. Niebieski i zielony są prawie równe, a czerwony nie jest zbyt mocny. Wiedząc to, można z pewnym przekonaniem wnioskować, że ma się do czynienia z niebieskozielonym.

Jeśli znasz już języki w rodzaju C++, PHP czy JavaScript, to znana jest Ci szesnastkowa (heksadecymalna) notacja kolorów. Format ten wygląda następująco:

`#rrggbb – np. #ff8800 – pomarańczowy`

Nie zapomnij o `#` (kratce) poprzedzającej wartość!

W tej sześciocyfrowej wartości pierwsze dwa znaki określają czerwony, kolejne dwa zielony, a ostatnie dwa niebieski. Komputery liczą potęgami dwójką, zamiast używać systemu dziesiętnego jak my, śmiertelnicy. Z tego względu system szesnastkowy ma podstawę 16 (2 do potęgi 4) i używa się w nim szesnastu znaków, w tym cyfr 0 – 9 oraz liter a – f. Litery a – f działają w praktyce jak liczby 10 – 15. Ponieważ każdy z kolorów składowych `rgb` reprezentowany jest przez parę takich znaków szesnastkowych, każdy może mieć jedną z 256 (16×16) możliwych wartości, co daje razem 16 777 216 ($256 \times 256 \times 256$) kombinacji barw.

Przykładowo, czysty czerwony to `#ff0000`, czysty zielony to `#00ff00`, a czysty niebieski to `#0000ff`.

Możesz też użyć skróconej notacji heksadecymalnej

`#rgb`

 jeżeli chcesz uzyskać kolor, którego każda para składa się z tych samych cyfr.

Nawiązując do powyższych przykładów, czysty czerwony możesz zapisywać jako `#f00`, czysty zielony jako `#0f0` i czysty niebieski jako `#00f`. `#ff3322` (mocną czerwień) można przedstawić skrótnie jako `#f32`. Jest to szczególnie przydatne do szybkiego podawania odcieni szarości. Możesz na przykład określić czerń wartością `#000`, 75-procentową szarość `#444`, 50-procentową `#888`, 25-procentową `#bbb`, a wartością `#fff` biel.

Numeryczny RGB (R, G, B)

Każdy element zdefiniowany jest wartością od 0 do 255. Wygląda on następująco:

`rgb(r, g, b)`

Deklaracja `rgb(0, 255, 0)` oznacza czystą zielon.

W istocie rzeczy jest to ten sam format, co heksadecymalny RGB, tylko że o innym sposobie zapisu wartości; każdemu z trzech kolorów można przypisać jedną z 256 wartości, wobec czego można zdefiniować tą notacją tyle samo wartości szesnastkowych. Różnica jest taka, że posługujemy się tu znajomym systemem dziesiętnym, którego uczymy się w przedszkolu, a nie systemem szesnastkowym, którego uczymy się w szkole na informatyce (a uczyłeś się informatyki, prawda?).

PROCENTY RGB (R%, G%, B%)

Jest to notacja, w której każdy kolor określany jest procentowo `r%`, `g%`, `b%`

Dopuszczalne są w niej wartości od 0% do 100%. Marny milion kombinacji kolorów ($100 \times 100 \times 100$), które można z tego uzyskać, większości z nas zupełnie wystarczy. Ponadto dużo łatwiej domyślić się koloru w procentach niż w notacji szesnastkowej.

Pełna czerwień to zatem `100%, 0%, 0%`, pełna zieleń to `0%, 100%, 0%`, a `46%, 76%, 80%` jest bliższe ciemnawemu kolorowi zielononiebieskiemu, który przedstawiłem wcześniej w notacji heksadecymalnej.

NOTACJA HSL (BARWA, NASYCENIE, JASNOŚĆ)

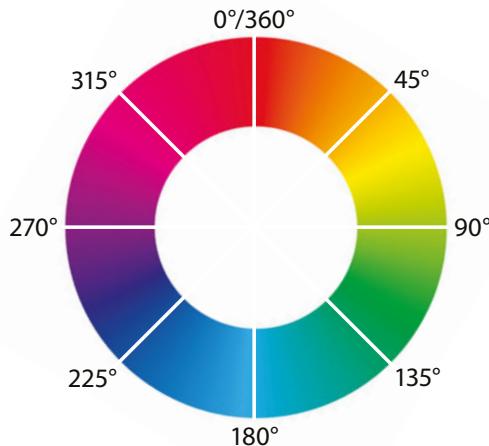
Format ten wygląda następująco:

`HSL (0, 0%, 0%)`

Notację HSL wskazuje się kolory w sposób bardziej intuicyjny niż przy użyciu przedstawionych dotąd wariantów rgb, ponieważ łatwiej tworzy się w nim i odczytuje kolory.

Pierwsza wartość w definicji HSL określa barwę, czyli właściwy kolor, np. czerwony lub zielony. Kolory rozłożone są na kole barw, a wartość barwy określa stopień na tym kole (**rysunek 2.20**).

RYSUNEK 2.20. W modelu kolorów HSL wartość barwy określona jest liczbą stopni na kole barw



Czerwony znajduje się na 0. i 360. stopniu. Cyjan znajduje się na przeciwnie, na 180. stopniu. Poniżej znajdziesz przybliżone wartości barwy kolorów tęczy.

Czerwony: 0

Pomarańczowy: 35

Żółty: 60

Zielony: 125

Niebieski: 230

Indygo: 280

Fioletowy: 305

Nasyście i jasność nietrudno zrozumieć. Nasyście określa intensywność barwy. Szarawe barwy mają mniej nasyście, a żywe kolory większe. Jasność określa, jak ciemny bądź jasny jest kolor; wartość 0% zawsze przekłada się na czarny, a 100% zawsze na biały, więc trzeba podać coś pośredniego, aby w ogóle zobaczyć barwę.

Mając powyższą listę wartości kolorów tęczy pod ręką w pracy, odkryjesz szybko, że możesz z łatwością stworzyć dowolny kolor. Podawanie kolorów RGB w zapisie szesnastkowym wymaga mieszanego poziomu kolorów w głowie, aby „odczytać” barwę, ale przy HSL wystarczy znać pojedynczą wartość barwy, aby następnie użyć odpowiedni odcień, regulując nasycenie i jasność, zaczynając od poziomu 50%.

KANAŁY ALFA

Zauważ, że zarówno RGB, jak i HSL pozwalają na podanie wartości kanału alfa, który określa krycie koloru (tj. jak bardzo tło przez niego prześwituje), przy użyciu formatów RGBA i HSLA. W obydwu formatach czynnikowi A (alfa) można nadać wartość ujętą pomiędzy 1 (pełnym kryciem) a 0 (przezroczystością). O wykorzystaniu kanałów alfa przeczytasz więcej w dalszych rozdziałach.

Dodatkowe materiały o kolorach

<http://colrd.com> — Colrd jest stroną w stylu Pinterest, na której znajdują się inspirujące grafiki i obrazy wraz z opartymi na nich paletami kolorystycznymi.

<http://kuler.adobe.com> — Na stronie Adobe Kuler znajdują się tysiące próbek kolorów, narzędzia do tworzenia palet i listy popularnych kolorów, z których korzystają inni ludzie.

Podsumowanie

W tym rozdziale poznaleś reguły CSS oraz dowiedziałeś się, jak określają właściwości stylistyczne elementów HTML. Zapoznałeś się z licznymi selektorami reguł, które można ze sobą łączyć w celu wybierania zbiorów elementów kodu, oraz dowiedziałeś się, jak deklaracje reguł definiują nowe ustawienia właściwości wybranych elementów. Zobaczyłeś, jak kaskadowość, dziedziczenie i precyzja wspólnie określają, który styl przypisywany jest elementowi, gdy stylów jest więcej niż jeden. Poznałeś też liczbowe i słowne sposoby zapisu wartości właściwości, a także różne modele kolorów, służące do podawania barw.

W następnym rozdziale nauczysz się, jak stylizować i nadawać strukturę hierarchii wizualnej tekstu na stronie, aby Twoja typografia wyglądała bardziej interesująco i profesjonalnie.

ROZDZIAŁ 3

Pozycjonowanie elementów

W TYM ROZDZIALE ZAPOZNASZ SIĘ z modelem polowym, właściwościami `position` i `display` oraz dowiesz się, jak tworzyć obiekty pływające i oczyszczające. Techniki pozycjonowania, które poznasz w tym rozdziale, są nieodzowne do zrozumienia CSS i pozwolą Ci z powodzeniem tworzyć layouty w tym języku.

Model polowy odnosi się do prostokątnych pól generowanych dla każdego znacznika HTML w kodzie. Pola te rozmieszczone są na stronie zgodnie z modelem formatowania graficznego. Formatowanie graficzne obsługiwane jest przede wszystkim trzema właściwościami: `position`, która określa relację przestrenną między elementami na stronie; `display`, która określa, czy elementy rozmieszczone są pionowo, obok siebie, czy może w ogóle nie są wyświetlane; `float`, która obsługuje dodatkowe funkcje pozycjonujące, a której można używać do rozmieszczenia elementów w(layoutach wielokolumnowych.

Model polowy

Jak widziałeś w rozdziale 1., każdy element kodu tworzy na stronie pole. Strona HTML jest zatem zbiorem pól.

Obramowania pól są domyślnie niewidoczne, a tła pól przezroczyste, wobec czego struktura polowa strony nie jest widoczna na pierwszy rzut oka. Jak wiesz, przy użyciu CSS i narzędzi w rodzaju Web Developer Toolbar łatwo włączyć widoczność obramowań oraz nadać kolor tłom pól. Dzięki temu możesz ujrzeć strukturę strony w zupełnie nowym świetle.

Przyjrzyjmy się najpierw właściwościom wszystkich pól elementów. Należą one do trzech grup:



Więcej na temat modelu polowego przeczytasz na stronie <http://www.w3.org/TR/RECSS2/box.html>.

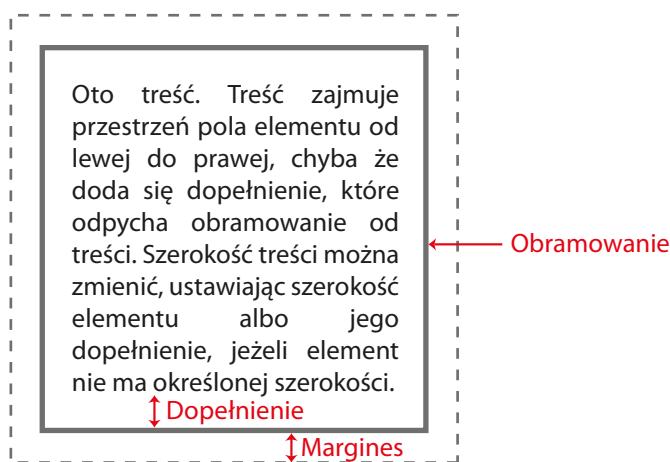
- **Obramowanie.** Możesz ustalić grubość, styl i kolor obramowania.
- **Dopełnienie.** Możesz ustawić odległość treści pola od jego obramowania.
- **Margins.** Możesz określić odległość między danym polem a sąsiednimi elementami.

Prostym sposobem na potraktowanie tych właściwości jest uznanie, że marginesy rozchodzą się poza obramowanie, a dopełnienie rozchodzi się do wewnątrz pola, co widać na **rysunku 3.1**. Ponieważ pole ma cztery boki, właściwości związane z obramowaniem, dopełnieniem i marginesem obsługują cztery ustawienia: **top**, **right**, **bottom** i **left**.

RYSUNEK 3.1. W tym diagramie modelu polowego widać relację między obramowaniem, dopełnieniem i marginesem elementu HTML



Pole elementu ma również warstwę tła, która można nadać kolor lub przypisać obraz. Właściwości związane z tłem elementu omówię pod koniec tego rozdziału.



O ile poszczególne wartości obramowania, marginesu i szerokości czterech krawędzi ramki określasz osobno, CSS oferuje kilka wygodnych zbiorczych metod stylizacji, dzięki którym nie musisz podawać dwunastu deklaracji dla każdego wyświetlanego pola. Więcej na ten temat przeczytasz w ramce „Właściwości zbiorcze”.



Czwarte ustawienie właściwości **border**, **border-radius**, nie wpływa na działanie modelu polowego; omówię je w rozdziale 7.

Obramowanie

Właściwość **border** obsługuje trzy ustawienia:

- **width** — można mu nadać wartość **thin**, **medium**, **thick** lub jakąkolwiek wartość liczbową o dowolnej jednostce miary, z wyjątkiem wartości procentowych i ujemnych.
- **style** — można mu nadać wartość **none**, **hidden**, **dotted**, **dashed**, **solid**, **double**, **groove**, **ridge**, **inset** lub **outset**.
- **color** — można mu nadać jakąkolwiek wartość kolorów (np. w zapisie RGB, HSL, szesnastkowym lub słowo kluczowe).

Właściwości zbiorcze

CSS obsługuje właściwości zbiorcze, służące do definiowania obramowań, dopełnień i marginesów w pojedynczej deklaracji. W deklaracjach zbiorczych krawędzie pola zawsze występują w kolejności: góra, prawa, dolna, lewa, czyli zgodnie z ruchem wskazówek zegara, począwszy od godziny 12. Zamiast pisać

```
{margin-top:5px; margin-right:10px; margin-bottom:12px; margin-left:8px;}
```

marginesy elementu możesz określić prostym

```
{margin: 5px 10px 12px 8px;}
```

Zwróć uwagę na spację pomiędzy kolejnymi wartościami; nie trzeba wstawiać separatora w rodzaju przecinka. Nie musisz nawet podawać wszystkich czterech wartości. Kiedy jakaś wartość zostaje pominięta, margines danej krawędzi otrzymuje wartość marginesu przeciwnielego krawędzi.

```
{margin: 12px 10px 6px;}
```

W poprzednim przykładzie, w którym brakuje ostatniej wartości, określającej margines lewej krawędzi, lewemu marginesowi nadana zostaje wartość marginesu prawej krawędzi — **10px**. W przykładzie

```
{margin: 12px 10px;}
```

podane są jedynie dwie pierwsze wartości, czyli górnej i prawej krawędzi, więc brakującym ustawieniom dolnej i lewej krawędzi nadane są odpowiednio wartości **12px** i **10px**. Wreszcie, jeśli podana jest tylko jedna wartość

```
{margin: 12px;}
```

to zostaje ona nadana wszystkim czterem krawędziom. W deklaracji zbiorczej nie możesz zdefiniować wartości dolnej i lewej krawędzi bez podania wartości krawędzi górnej i prawej, nawet jeśli mają one mieć wartość zerową. W takim przypadku możesz jednak podać o, tak jak widać poniżej

```
{margin: 0 0 2px 4px;}
```

Istnieją ponadto trzy poziomy szczegółowości definiowania właściwości pól, których używa się w zależności od tego, z jaką dokładnością chcesz się odnieść do danej krawędzi i właściwości pola. Poziomy te, od najmniej do najbardziej szczegółowego, są następujące:

1. Wszystkie trzy ustawienia odnoszące się do wszystkich czterech krawędzi

```
{border: 2px dashed red;}
```

2. Jedno ustawienie odnoszące się do wszystkich czterech krawędzi

```
{border-style: dashed;}
```

3. Jedna właściwość odnosząca się do jednej krawędzi

```
{border-left-style: dashed;}
```

Zestawianie tych trzech poziomów stylizacji zbiorczej w celu uzyskania pożądanego rezultatu jest powszechnie. Powiedzmy, że chcę, żeby pole miało czerwoną krawędź, szeroką na cztery piksele u góry i u dołu, jednopikselową, czerwoną krawędź po lewej, a po prawej w ogóle nie miało obramowania.

```
{border:4px solid red;} /* nadaje wszystkim krawędziom jednakową wartość */
```

```
{border-left-width:1px;} /* zmienia szerokość lewej krawędzi */
```

```
{border-right:none;} /* usuwa prawą krawędź */
```

Te same trzy poziomy wariacji można wykorzystywać z innymi właściwościami, m.in. `padding` i `border-radius`.

W ramce „Właściwości zbiorcze” przedstawiłem kilka prostych stylów obramowania. Poniżej znajdziesz kilka bardziej złożonych przykładów, które pozwolą Ci zebrać ogół wiedzy w całość.

```
p.warning {border:solid #F33;}
```

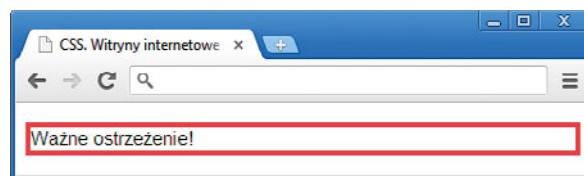
Powyższa reguła nadaje każdemu akapitowi z klasy `warning` przyciągające uwagę, czerwone obramowanie, szerokie na cztery piksele i będące jednolitą linią (rysunek 3.2).

RYSUNEK 3.2. Ponieważ chcę, by wszystkie krawędzie były jednakowe, obstylouję obramowanie zbiorczą właściwością border



Szerokości `thin`,
`medium` i `thick` nie są konkretnie zdefiniowane w rekomendacjach CSS, toteż grubość linii podanych słowami kluczowymi może się różnić w zależności od przeglądarki. Style linii, z wyjątkiem `solid`, który oznacza prostą, ciągłą linię, nie są konkretnie opisane w specyfikacjach CSS, toteż obramowanie o stylu `dashed` może się różnić pod względem wyglądu kresek i odstępów między nimi w zależności od przeglądarki, w jakiej jest wyświetlane.

RYSUNEK 3.3. Definiując osobną regułę `border-width`, nadaje odmienne wartości szerokościom poszczególnych krawędzi pola

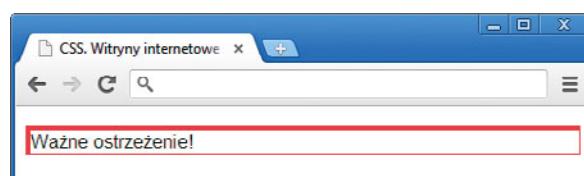


Powiedzmy, że chcę, by obramowanie było jednolitą, czerwoną linią ze wszystkich stron, ale — dla urozmaicenia — cieńszą z prawej strony i u dołu. Do uzyskania pożądanego rezultatu użyję dwóch reguł. Pierwsza ma wykorzystać właściwość zbiorczą `border`, odnoszącą się do wszystkich czterech krawędzi naraz, a druga określić osobno grubość obramowania różnych krawędzi właściwością `border-width`.

```
p.warning {border:solid #f33;}
```

```
p.warning {border-width: 4px 1px 1px 4px;}
```

W rezultacie otrzymujemy to, co widać na rysunku 3.3.



Tymczasowe wyświetlanie obramowania pola podczas pracy nad stroną bywa bardzo przydatne, gdyż można dzięki temu uzyskać lepszygląd rezultatów zastosowania stylów w rodzaju `margin` i `padding`. Style pola elementów domyślnie wyglądają tak, że `border-width` ma wartość `medium`, `border-style` wartość `none`, a `border-color` wartość `black`. Ponieważ `border-style` ma wartość `none`, to

obramowanie nie jest widoczne. Aby wyświetlić obramowanie akapitu, możesz podać następującą regułę

```
p {border: solid 1px;}
```

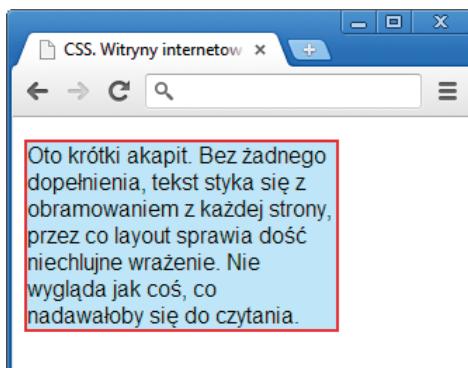
Nadaje to właściwości `border-style` charakter jednolitej linii, dzięki czemu ramka staje się widoczna. Zauważ, że nadałem obramowaniu szerokość `1px` — mniejszą od domyślnej szerokości `3px` — aby zminimalizować jego wpływ na szerokość i wysokość układu graficznego.

Dopełnienie

Dopełnienie tworzy przestrzeń między zawartością pola a jego obramowaniem. Na [rysunku 3.4](#) widać treść tekową elementu i obramowanie pola.

```
p {font:16px helvetica, arial, sans-serif; width:220px;  
border:2px solid red; background-color:#caebff;}
```

RYSUNEK 3.4. Kiedy brakuje dopełnienia, treść pola styka się z obramowaniem

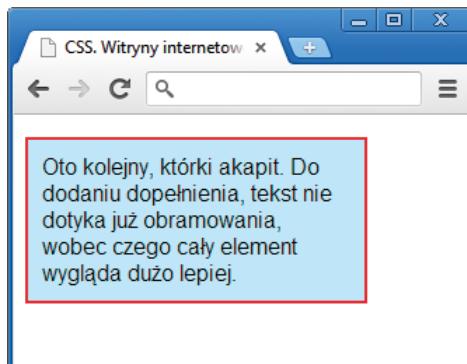


Tekst elementu domyślnie styka się z obramowaniem elementu, co nie wygląda zbyt dobrze. Odrobina dopełnienia może znaczco zmienić rezultat, co widać na [rysunku 3.5](#).

```
p {font:16px helvetica, arial, sans-serif; width:220px;  
border:2px solid red; background-color:#caebff;  
padding:10px;}
```

Jako część wewnętrznego obszaru pola, dopełnienie wyświetlane jest w kolorze tła elementu. Porównując [rysunek 3.4](#) i [3.5](#), można zauważyc, że dopełnienie powiększa szerokość pola, zamiast — jak można by się spodziewać — zmniejszyć obszar treści. Wróć do tego jeszcze w dalszej części rozdziału.

RYSUNEK 3.5. Jeśli wyświetlasz obramowanie elementu, niemal zawsze warto dodać dopełnienie, aby zawartość elementu się z nim nie stykała



Margines

Problem marginesu pola jest nieco bardziej skomplikowany niż obramowania i dopełnienia. Na rysunku 3.6 widnieją trzy zestawienia składające się z jednego nagłówka i dwóch akapitów. W pierwszym widać domyślny wygląd nagłówka i akapitów. W drugim widnieje to samo zestawienie, ale z widocznymi obramowaniami, co pozwala ujrzeć, że domyślne marginesy tworzą odstęp między elementami. W trzecim zestawieniu widać, co się dzieje po nadaniu marginesom wartości zerowej — elementy stykają się.

Przykład marginesu

Te dwa akapity nie są w żaden sposób obstylowane.

Wyraźnie jednak widać odstęp między nimi.

Przykład z włączonymi obramowaniami

Nagłówek wraz z obydwoema akapitami zestawiono tutaj jednakowo, ale tym razem obramowania są widoczne. Widać teraz, jak domyślne marginesy tworzą odstęp między elementami.

Marginsy odsuwają element od elementów, z którymi by się w innym wypadku stykały.

Przykład z włączonymi obramowaniami, ale bez marginesów

Marginsy mają tutaj wartość 0 (zero), wobec czego obramowania elementów stykają się ze sobą. Badź świadom, że niemal każdy element ma domyślnie przypisane marginesy. Często jednak dla uzyskania pożądanego efektu trzeba je usunąć lub zmienić ich wielkość.

RYSUNEK 3.6. Umiejętność kontrolowania marginesów elementów jest kluczowa dla tworzenia layoutów

Wyzerowanie marginesów i dopełnień

Warto mieć zwyczaj zamieszczania następującej deklaracji jako pierwszej reguły w arkuszu stylów:

```
* {margin: 0; padding: 0;}
```

Sprawia to, że domyślne marginesy i dopełnienia wszystkich elementów zostają wyzerowane. Po zamieszczeniu tego w arkuszu stylów wszystkie domyślne marginesy i dopełnienia znikają. Następnie, przy dalszym stylizowaniu strony, możesz przywracać marginesy tym elementom, które powinny je mieć. Jak się jeszcze przekonasz, domyślne wartości dopełnienia i marginesów różnią się w zależności od przeglądarki, zwłaszcza jeśli chodzi o złożone elementy w postaci formularzy i list. „Neutralizując” domyślne ustawienia i dodając swoje własne, pozwalasz na uzyskanie większej spójności wyglądu strony na przestrzeni różnych przeglądarek.

Arkusz stylów reset.css Erica Meyera, z którego korzystam we własnych projektach, nie tylko wyzerowuje marginesy i dopełnienia, ale pomaga również ustandaryzować wygląd wielu elementów w różnych przeglądarkach. Przemyślenia Erica na temat rozległego wyzerowywania domyślnych ustawień przeglądarki znajdziesz na stronie <http://meyerweb.com/eric/thoughts/2007/04/18/reset-reasoning/>. Jeśli chcesz, możesz też pobrać plik reset.css ze strony <http://meyerweb.com/eric/tools/css/reset>.

Scalanie marginesów

Marginesy w pionie ulegają scaleniu, o czym koniecznie musisz pamiętać. Pozwól, że wytłumaczę, co to oznacza i dlaczego to jest ważne. Wyobraź sobie, że masz trzy akapity, jeden po drugim, a każdy jest obstylowany regułą

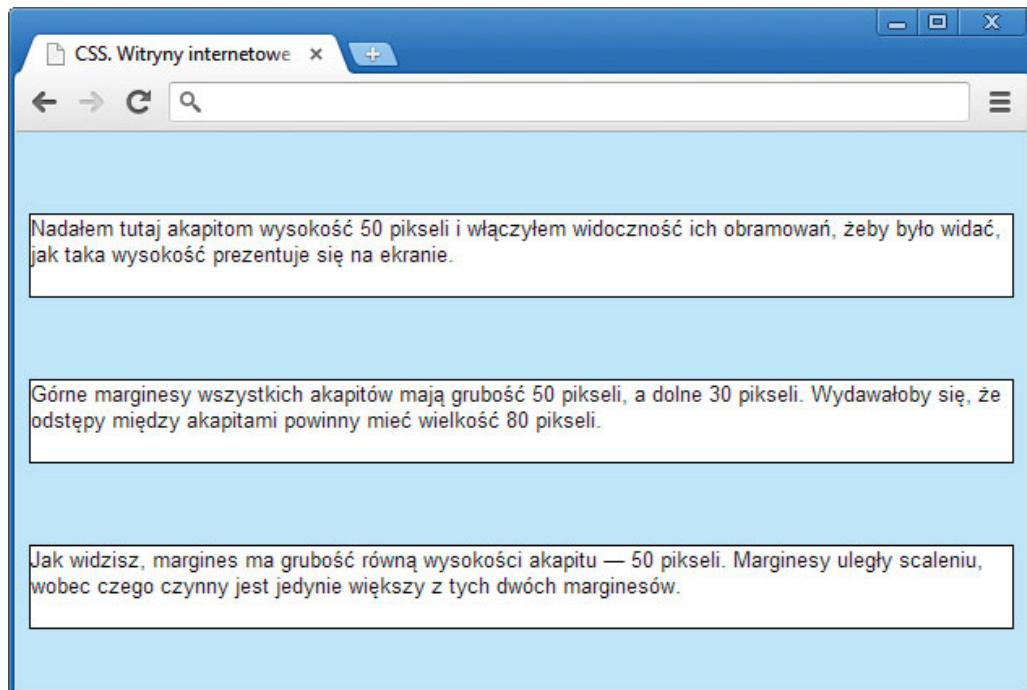
dla zwięzości pomijam deklaracje fontów

```
p {height:50px; border:1px solid #000; backgroundcolor:#fff; margin-top:50px; margin-bottom:30px;}
```

Ponieważ dolny margines pierwszego akapitu sąsiaduje z górnym marginesem drugiego, mógłbyś przypuścić, że pomiędzy akapitami znajduje się odstęp 80 pikseli ($50 + 30$). Tak jednak nie jest. W rzeczywistości dzieli je odstęp 50 pikseli. Kiedy górny i dolny margines się stykają, nachodzą one na siebie, aż któryś z nich dotrze do krawędzi sąsiadującego elementu. W tym przypadku pierwszy do krawędzi dociera większy, górny margines dolnego akapitu i określa tym samym oddalenie elementów — tutaj 50 pikseli (rysunek 3.7). Efekt ten nazywamy scalaniem marginesów.

 Choć pionowe marginesy ulegają scaleniu, to z marginesami w poziomie tak nie jest. Marginesy poziome zachowują się zgodnie z oczekiwaniami: ich wartości zostają dodane, tworząc odstęp między rozmieszczenymi poziomo elementami.

Pozwól, że wytłumaczę celowość efektu scalania. Wyobraź sobie sekwencję obstylowanych jednakowo akapitów. Kiedy któryś z nich jest pierwszym lub ostatnim z serii, jego górny bądź dolny margines określa jego odległość od górnej lub dolnej krawędzi jego kontenera. Akapity znajdujące się pomiędzy nie potrzebują obydwu marginesów. Jak widać na rysunku 3.7, marginesy te zwyczajnie ulegają scaleniu, a odległość definiowana jest wielkością większego marginesu.



RYSUNEK 3.7. Pionowe marginesy ulegają scaleniu (o ile nachodzą na siebie), aż margines jednego elementu styka się z obramowaniem drugiego

Wybieranie jednostek miary marginesów

Często możesz zechcieć przemieszać jednostki miary przy definiowaniu marginesów elementów tekstowych. Lewy i prawy margines akapitu możesz określić w pikselach, aby tekst znajdował się w określonym oddaleniu od krawędzi kontenera niezależnie od wielkości tekstu, ale górne i dolne marginesy określić jednostkami em, aby odstępy pionowe między akapitami odnosiły się do wielkości tekstu:

Zwróć uwagę na zapis zbiorczy — górnemu i dolnemu marginesowi nadaję wielkość 0,75 em, a lewemu i prawemu 30 pikseli.

`p {font-size:1em; margin:.75em 30px;}`

W tym przykładzie odstęp pionowy między akapitami zawsze jest wielkością trzech czwartych wysokości tekstu (górny i dolny margines, obydwa o wielkości 0,75 em, scalają się w jeden margines o takiej właśnie wielkości). Jeżeli powiększysz tekst bądź jeśli użytkownik to zrobi, sam tekst akapitu stanie się większy, ale odstępy między akapitami również ulegną powiększeniu w proporcjonalnym stopniu, zachowując ogólny wygląd layoutu. W tym

przypadku lewy i prawy margines, które są zdefiniowane pikselami, pozostały niezmienione, gdyż zapewne nie chcesz, by zmiana wielkości tekstu wpływała na szerokość layoutu.

Wielkość pola



Wraz z przechodzeniem na standardy HTML5 i zanikaniem starszych, niestandardowych przeglądarek definicje typu dokumentu XHTML, wskazujące, czy kod strony ma być interpretowany zgodnie ze standardem (i z przedstawieniem zachowania współczesnego modelu polowego W3C) czy w trybie osobliwości (z innym zachowaniem modelu polowego w przeglądarce IE6 i wcześniejszych wersjach), stają się zbędne. Jeśli chcesz się dowiedzieć więcej o trybie osobliwości, zajrzyj na stronę <http://www.quirksmode.org/css/quirksmode.html>.

Sposób działania modelu polowego W3C jest odpowiedzialny za niektóre z najbardziej frustrujących — zarówno dla początkujących, jak i dla ekspertów — aspektów CSS. Zauważ, że w dalszej części tekstu będę się odnosił do elementów blokowych, takich jak nagłówki, akapity i listy; elementy liniowe zachowują się inaczej.

Omówmy teraz model polowy krok po kroku. Zaczynając od omówienia definiowania szerokości pola, jako że jest to kluczowe przy tworzeniu wielokolumnowych layoutów. Na początek zobaczymy efekty dodania obramowań, dopeleń i marginesów do elementu o nieokreślonej szerokości. Następnie przyjrzymy się zmianie zachowania następującej po nadaniu elementowi szerokości w CSS.

POLE O NIEOKREŚLONEJ SZEROKOŚCI

Od tej pory będę używał pojęć „element” i „pole” zamiennie, stosownie do tego, które w danej chwili będzie bardziej trafne.

Kiedy nie ustalasz szerokości elementu blokowego, to jego domyślna szerokość ma wartość `auto` — przejmuje on wtedy szerokość swojego rodzica. Przyjrzymy się elementowi w domyślnym stanie `auto`.

Wykorzystamy do tego prosty kod

```
<body>
  <p>Szerokość tego elementu nie jest określona...</p>
</body>
```

i użyjemy następującego kodu CSS:

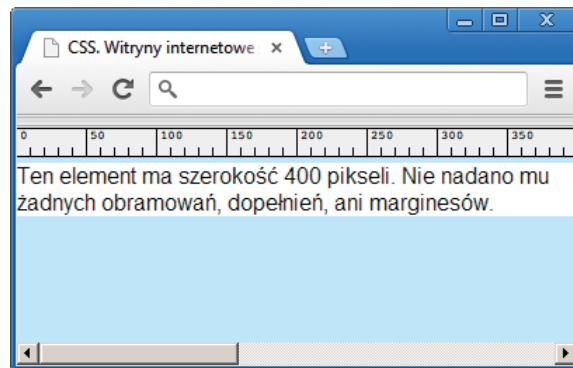
```
body {font-family:helvetica, arial, sans-serif;
      font-size:1em; margin:0px; background-color:#caebff;}
p {margin:0; background-color:#fff;}
```

Elementowi okalającemu — `body` — określiłem font, tło i usunąłem domyślne marginesy, co pozostanie niezmienione w kolejnych przykładach pracy z szerokością pól. Pozbawiłem akapit domyślnych marginesów. Bez nich element `body` wypełnia całkowicie okno przeglądarki, a akapit wypełnia element `body`, co widać na rysunku 3.8.

RYSUNEK 3.8. Widoczny tu akapit nie ma obramowania, dopełnienia i marginesów

 U góry okna zamieściłem grafikę z miarką (której pomijam w kodzie), abyś mógł dokładnie zobaczyć, co dzieje się na każdym etapie. Dzięki temu mogłem również dokładnie przeciągnąć róg okna, by uzyskać szerokość 400 pikseli na potrzeby przykładu.

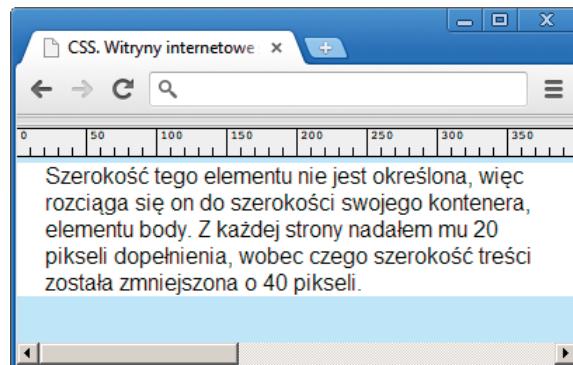
pomijam deklaracje fontów w celu zachowania zwięzości



Kiedy brakuje obramowania, dopełnienia i marginesów, tekst akapitu również ma szerokość elementu `body`. Dodajmy teraz odrobinę dopełnienia, aby stworzyć nieco przestrzeni wokół tekstu (rysunek 3.9).

`p {margin:0; background-color:#fff; padding:0 20px;}`

RYSUNEK 3.9. Dopełnienie sprawia, że blok tekstu staje się węższy

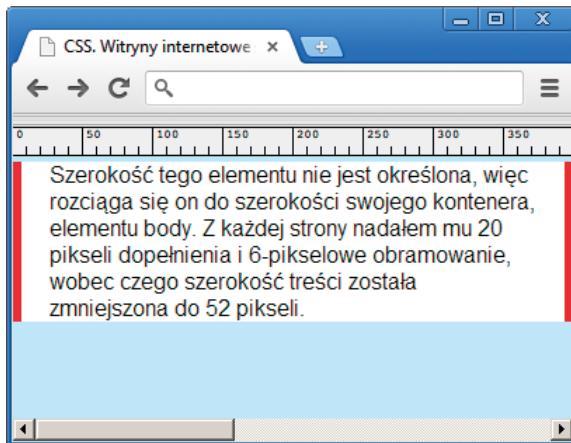


Po dodaniu dopełnienia blok tekstowy jest teraz szeroki na 360 pikseli, gdyż po obydwu jego stronach znajduje się 20 pikseli pustej przestrzeni.

Następnie dodaję lewe i prawe obramowanie, szerokie na 6 pikseli (**rysunek 3.10**).

```
p {margin:0; background-color:#fff; padding:0 20px;  
border:solid red; border-width:0 6px 0 6px;}
```

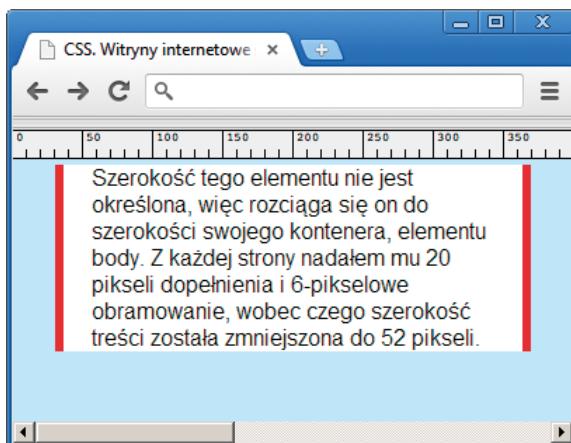
RYSUNEK 3.10. Dodanie obramowania dodatkowo zmniejsza szerokość pola treści



Po dodaniu z obydwu stron 6-pikselowego obramowania i 20 pikseli dopełnienia treść jest teraz szeroka na 348 pikseli ($400 - 52$). Wreszcie, dodajmy po obydwu stronach elementu margines (**rysunek 3.11**).

```
p {margin:0 30px; background-color:#fff; padding:0 20px;  
border:solid red; border-width:0 6px 0 6px;}
```

RYSUNEK 3.11. Pole treści jest dużo węższe po dodaniu obramowania, dopełnienia i marginesów



Marginesy tworzą przestrzeń między polem elementu a oknem, wobec czego treść jest teraz szeroka na 288 pikseli ($400 - ((20 + 6 + 30) \times 2)$). Całkowita przestrzeń elementu pozostaje jednak taka sama — jest to szerokość rodzica, czyli 400 pikseli.

Spostrzeżenie #1. Elementy bez ustawionej szerokości zawsze rozszerzają się do szerokości swojego kontenera. Dodanie poziomych obramowań, dopeleń i marginesów sprawia, że obszar treści zmniejsza swoją szerokość ołączną szerokość tychże elementów stylistycznych.

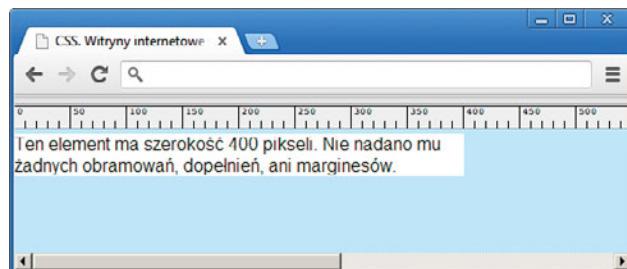
POLE O OKREŚLONEJ SZEROKOŚCI

Wykonajmy to samo ćwiczenie z użyciem elementu o szerokości zdefiniowanej w CSS (rysunek 3.12).

pomijam deklaracje fontów
w celu zachowania związkowości

`p {width:400px; background-color:#fff; margin:0;}`

RYSUNEK 3.12. Po określeniu właściwości width ten element blokowy nie odznacza się już domyślnym zachowaniem polegającym na rozszerzaniu się do szerokości kontenera (w tym wypadku elementu body)

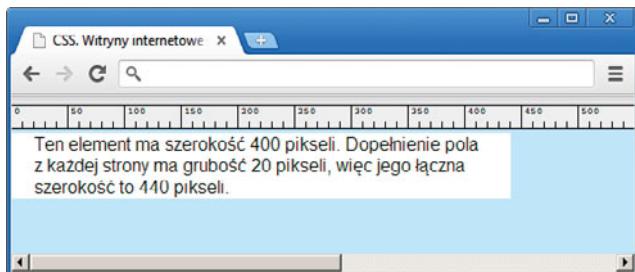


Akapit ma teraz ustaloną szerokość 400 pikseli. Bez dopełnienia treści elementu również jest tej szerokości i styka się z krawędziami pola. Nadajmy teraz elementowi 20-pikselowe dopełnienie:

`p {width:400px; background-color:#fff; margin:0; padding:20px;}`

Po poprzednim przykładzie możesz oczekiwac, że treść elementu zostanie skrócona do szerokości 360 pikseli. Tak jednak nie jest. Zamiast tego, kiedy szerokość pola jest ustalona, dodanie dopełnienia sprawia, że staje się on szerszy o 40 pikseli (rysunek 3.13).

RYSUNEK 3.13. Dodanie dopełnienia sprawia, że pole staje się szersze

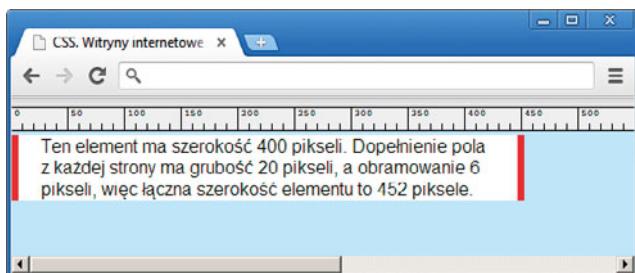


Jeśli następnie dodam 6-pikselowe obramowanie do prawej i lewej krawędzi pola (**rysunek 3.14**)

```
p {width:400px; background-color:#fff; margin:0; padding:0 20px; border:solid red; border-width:0 6px 0 6px;}
```

to pole staje się szersze o kolejne 12 pikseli. Teraz pole, które było pierwotnie szerokie na 400 pikseli, ma szerokość 452 pikseli ($6 + 20 + 400 + 20 + 6 = 452$).

RYSUNEK 3.14. Dodanie obramowania sprawia, że pole staje się jeszcze szersze

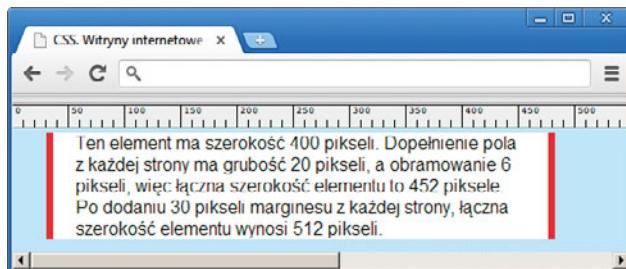


Dodajmy teraz jeszcze prawy i lewy margines, żeby stworzyć przestrzeń po obydwu stronach elementu (**rysunek 3.15**).

```
p {width:400px; background-color:#fff; margin:0 30px; padding:0 20px; border:solid red; border-width:0 6px 0 6px;}
```

Dodając 30-pikselowe marginesy, dalej zwiększamy ogólny obszar elementu, który ma teraz szerokość 512 pikseli ($30 + 6 + 20 + 400 + 20 + 6 + 30 = 512$).

RYSUNEK 3.15. Marginesy tworzą dodatkową przestrzeń wokół elementu. Zmieniem rozmiar okna przeglądarki, żeby ukazać bieżącą szerokość elementu



Nowa właściwość CSS3 **box-sizing** pozwala sprawić, by pole o określonej szerokości zachowywało się tak, jakby nadano mu domyślną wartość **auto**. Jest ona jednak obsługiwana tylko przez najnowsze przeglądarki, więc w chwili, gdy to piszę (lato 2012 roku), nie polecam korzystania z niej.

Spostrzeżenie #2. Pole o określonej szerokości poszerza się i zwiększa swój obszar wraz z dodawaniem obramowań, dopełnień i marginesów. Właściwość width w praktyce jedynie określa szerokość treści pola, a nie obszar pola w poziomie.

Ten bardzo szczegółowy opis różnic w zachowaniu pól o ustalonej i nieustalonej szerokości przedstawiłem tu celowo. Owe różnice mają bardzo duże znaczenie przy tworzeniu layoutów wielokolumnowych, gdzie kolumny muszą zachowywać swoją szerokość, by układ graficzny funkcjonował poprawnie. „Pływające layouty”, o których dowiesz się w rozdziale 5., mogą być wyświetlane niepoprawnie, jeśli szerokość kolumny ulega nieumyślnemu zwiększeniu w wyniku zmian obramowań, dopełnień i marginesów.

Musisz więc zapamiętać, że kiedy właściwości **width** elementu nadana jest wartość, to reaguje on inaczej, niż gdy działa zgodnie z domyślnym ustawieniem **auto**, gdy dodawane są do niego obramowania, dopełnienia i marginesy.

Przyjrzyjmy się teraz innej kluczowej technice, którą musisz zrozumieć, by tworzyć layouty oparte na CSS — tworzeniu elementów pływających i oczyszczających.

Elementy pływające i oczyszczające

Kolejna wartościowa technika służąca organizowaniu układu strony wiąże się z tworzeniem elementów pływających i oczyszczających przy użyciu właściwości **float** i **clear**. Nadanie elementowi właściwości **float** pozwala na wyciągnięcie go ze standardowej struktury wizualnej dokumentu. Służy to nie tylko temu (jak to pierwotnie było), żeby oblewać tekst wokół obrazów, ale także do tworzenia kolumn i rozmieszczania obok siebie w poziomie elementów blokowych. Elementy, które następują po pływającym elemencie, umieszczone są obok niego, jeżeli starczy na to miejsca.



Zachowaniem pływających elementów rzodzi wiele innych zasad.

Przeczytasz o nich w książce Erica Meyer Cascading Style Sheets 2.0 Programmer's Reference (2006, McGraw-Hill Osborne Media).

W skrócie, jak to ujmuje Eric:
„Zasada o pływających elementach mówi, żeby „umieścić je możliwie najwyżej i najdalej w którąś stronę””. Pomimo że ta książka została wydana dłuższy czas temu, jest ona fundamentalnym źródłem dla każdego szanującego się programisty CSS, gdyż nigdzie indziej tak szczegółowo nie opisano zasad rzządzących działaniem CSS.



Moduł Columns CSS3 określa, jak tworzyć kolumny przy użyciu CSS, ale kiedy to piszę, jedynie Opera i IE10 obsługują tę możliwość.

Na razie można zatem przyjąć, że przez dłuższy czas zastosowanie elementów pływających będzie najlepszym sposobem na tworzenie kolumn.

w celu zachowania zwięzości pomijam deklaracje fontów

margines ma sprawić, by tekst nie stykał się z obrazem

Właściwość **clear** pozwala na powstrzymanie takiego elementu przed przemieszczeniem się na miejsce obok elementu pływającego. Jeżeli masz na przykład dwa akapity i chcesz, żeby jedynie pierwszy z nich znalazł się obok pływającego elementu, pomimo że obydwa by się zmieściły, możesz „oczyścić” drugi akapit, zatrzymując go tym samym pod elementem pływającym.

Zanim omówię szczegółowo obydwie te właściwości, muszę wspomnieć, że choć zastosowanie elementów pływających jest kluczową i bardzo przydatną techniką CSS, to ich obecność może być dość dezorientująca dla początkujących programistów CSS. Jest tak, ponieważ pływające elementy wykraczają poza rama struktury graficznej dokumentu, wpływając tym samym na rozmieszczenie elementów, które je zawierają lub następują po nich. Dalsze przykłady zastosowania elementów pływających i oczyszczających rozplanowałem uważnie, by dostarczyć Ci wiedzy, która pozwoli Ci na udaną pracę z elementami pływającymi.

Właściwość float

Właściwość **float** służy przede wszystkim do oblewania obrazów tekstem, ale zastosowanie jej jest również najprostszym sposobem tworzenia wielokolumnowych layoutów.

Zacznijmy od przykładu oblewania obrazu tekstem.

OBLEWANIE OBRAZU TEKSTEM

Aby efekt oblewania mógł zadziałać, w kodzie trzeba najpierw podać obraz, a dopiero po nim oblewający tekst.

```
<img ..... />
<p>...tekst akapitu...</p>
```

Oto kod CSS.

```
p {margin:0; border:1px solid red;}
img {float:left; margin:0 4px 4px 0;}
```

Ten kod CSS sprawia, że obraz spływa w lewo, wobec czego tekst oblewa go z prawej (**rysunek 3.16**).

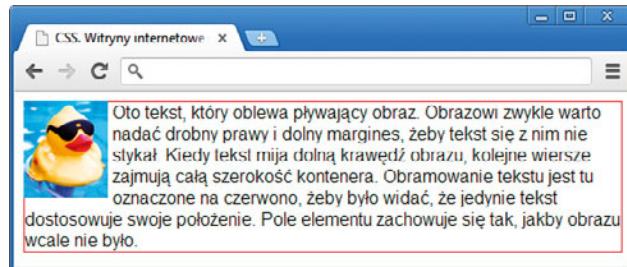
Mówiąc krótko, kiedy sprawiasz, że obraz lub dowolny inny element staje się pływający, nakazujesz przeniesienie go możliwie jak najdalej w górę i do jednej krawędzi bocznych jego rodzica; w tym przykładzie rodzicem jest **body**. Widzimy też, że akapit (z czerwonym obramowaniem) nie traktuje pływającego elementu jako elementu poprzedzającego go w strukturze graficznej dokumentu, więc także on zajmuje położenie w lewym górnym rogu rodzica. Tym niemniej jego treść, czyli tekst, oblewa pływający obraz.

RYSUNEK 3.16. Pływający obraz zostaje wyciągnięty ze struktury graficznej dokumentu. Jeżeli w kodzie znajduje się po nim element tekstowy, to jego tekst oblewa obraz



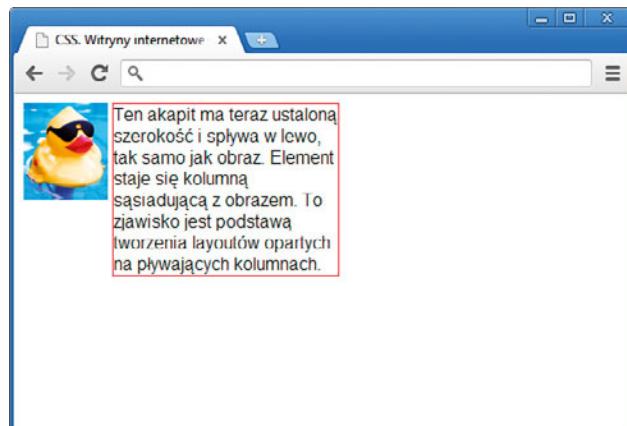
Kiedy tworzysz element pływający, musisz także określić jego szerokość, żeby nie narazić się na nieprzewidywalne rezultaty. Obrazy jednak z założenia mają szerokość, więc nie musisz im w takiej sytuacji przypisywać szerokości samodzielnie.

RYSUNEK 3.17. Kiedy akapit o określonej szerokości sąsiadujący z pływającym obrazem sam staje się elementem pływającym, zostaje on kolumną i nie oblewa już obrazu



Odtąd nietrudno będzie używać właściwości `float` do tworzenia kolumn (**rysunek 3.17**). Wystarczy nadać akapitowi szerokość i zrobić z niego element pływający.

```
p {float:left; margin:0; width:200px; border:1px solid red;}
img {float:left; margin:0 4px 4px 0;}
```



Kiedy zarówno obraz, jak i akapit o ustalonej szerokości są elementami pływającymi, efekt oblewania tekstem przestaje działać, a akapit również stara się przesunąć ku górze i możliwe najdalej w lewo, tym samym stając się kolumną sąsiadującą z obrazem. Oto jak wygląda tworzenie wielokolumnowych layoutów przy użyciu właściwości `float`. Wystarczy nadać kilku elementom braciom określone szerokości, nadać właściwość `float` i — jeśli wystarczy miejsca — zostaną one rozmieszczone obok siebie.

Jeśli utworzysz trzy pływające elementy o określonej szerokości, to zostaną w ten sposób rozmieszczone obok siebie, tworząc layout z trzech kolumn, działających jako kontenery, w których można zawrzeć inne elementy. Pływające layouty omówię dogłębnie w rozdziale 5.

Przyjrzyjmy się teraz innemu ważnemu aspektowi pływających elementów: takie elementy znajdują się poza ciągiem strukturalnym, wobec czego nie są zawarte w swoich elementach rodzicach. Może to zniekształcać wygląd layoutu.

Trzy sposoby włączania pływających elementów do kontenerów

Ponieważ pływający element wyłączony jest z ciągu strukturalnego dokumentu, jego rodzic nie widzi go, a więc nie zawiera go w sobie. Jako że takie zachowanie nie jest zawsze pożądane, pokażę Ci trzy sposoby, na jakie możesz zmusić elementy, by włączały w siebie swoje dzieci, którym nadano właściwość `float`. Musisz zapoznać się ze wszystkimi trzema, aby móc wybrać spośród nich najlepszy w danej sytuacji.

W celu zilustrowania takiego zachowania, jego wpływu na postać layoutu i trzech sposobów, by temu zaradzić, zaczniemy od omówienia przykładu obrazu i jego podpisu zawartego w znaczniku `section`. Po znaczniku `section` znajduje się element `footer`, który — w tym przykładzie — oznacza szeroką na całą stronę stopkę, na jaką często trafiajemy u dołu stron internetowych.

```
<section>
  
  <p>Fajnie sobie pływać.</p>
</section>
<footer> Oto stopka, która znajduje się u dołu strony.</footer>
```

Aby móc dokładnie zobaczyć, co się dzieje, pola `section` i `footer` wyświetlam tak, jak widać to na [rysunku 3.18](#).

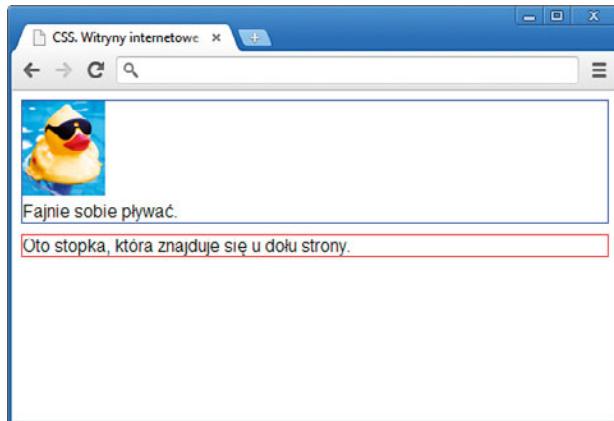
```
section {border:1px solid blue; margin:0 0 10px 0;}
```

usuwa duże górne i dolne marginesy  `p {margin: 0;}`

pomijam deklaracje fontów  `font {border:1px solid red;}`
w celu zachowania zwięzości

Mamy tu do czynienia z normalną strukturą dokumentu: elementy blokowe obejmują wszelkie swoje dzieci i rozmieszczone są w pionie. Powiedzmy, że chcemy, by podpis znajdował się po prawej stronie obrazu, a nie pod nim. Jak widziałeś w poprzednim ćwiczeniu, najprościej tego dokonać, zmieniając obraz w element pływający. Spróbujmy.

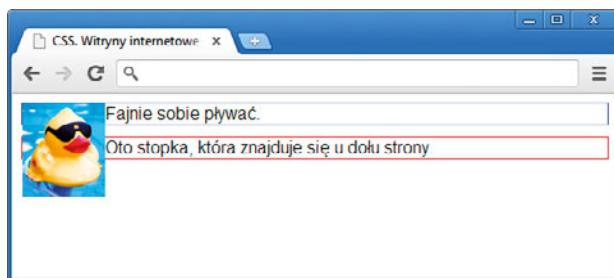
RYSUNEK 3.18. Dwa elementy blokowe: `section`, który zawiera obraz i podpis, oraz `footer` w ramach standardowej struktury dokumentu rozmieszczone są jeden pod drugim



```
section {border:1px solid blue; margin:0 0 10px 0;}
img {float:left;}
footer {border:1px solid red;}
```

Rezultat widać na rysunku 3.19.

RYSUNEK 3.19. Kiedy obraz przekształca się w element pływający, aby można było umieścić obok niego podpis, rodzić obydwo elementów `section` skraca swoją wysokość do wysokości niepływającego elementu tekstopowego



Ojej! Podpis znajduje się obok obrazu, tak jak chcieliśmy, ale element `section` nie zawiera już pływającego elementu, a jedynie niepływający element tekstowy. Znacznik `footer` przesuwa się w górę i znajduje się bezpośrednio pod poprzedzającym go elementem blokowym, `section`, tak jak powinien. Rezultat nie jest jednak zgodny z oczekiwaniami.

METODA 1. — NADAJ RODZICOWI WŁAŚCIWOŚĆ OVERFLOW:HIDDEN

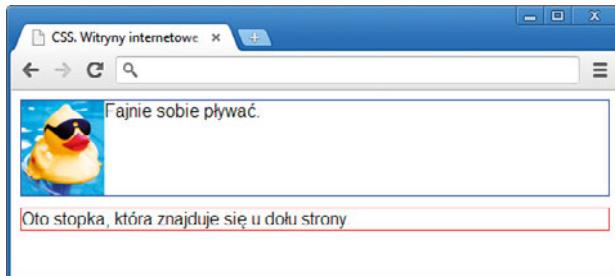
Na to, żeby znacznik `section` zawarł w sobie pływający element, jest prosty, choć mało intuicyjny sposób: wystarczy nadać rodzicowi właściwość `overflow:hidden`.

```
section {border:1px solid blue; margin:0 0 10px 0;
overflow:hidden;}
```

```
img {float:left;}  
p {border:1px solid red;}
```

Po nadaniu kontenerowi deklaracji `overflow:hidden` stopka powraca na właściwe miejsce (**rysunek 3.20**).

RYSUNEK 3.20. Kontener, po nadaniu mu deklaracji `overflow:hidden`, obejmuje teraz pływające elementy, które są jego zawartością



Właściwość `overflow: hidden` właściwie służy temu, by treści o zbyt dużej wielkości — takie jak duże obrazy — nie zmuszały swoich kontenerów do zwiększenia swojej wielkości na tyle, by móc je w pełni wyświetlić. Gdy nadana jest właściwość `overflow:hidden`, kontener zachowuje swoją zdefiniowaną wielkość, a „przepięcie” zawartość jest zwyczajnie obcinana. Tymczasem `overflow:hidden` bardzo dobrze sprawdza się w swojej drugiej roli, zmuszając elementy do objęcia swojej pływającej zawartości.

METODA 2. — ZMIEN RODZICA W PŁYWĄCY ELEMENT

Drugim sposobem jest przekształcenie rodzica w pływający element.

```
section {border:1px solid blue; float:left; width:100%;}  
img {float:left;}  
footer {border:1px solid red; clear:left;}
```

Element `section`, jako element pływający, obejmuje swoje dzieci, ograniczając swój rozmiar do nich, bez względu na to, czy same są pływające. Trzeba zatem dodać właściwość `width:100%`, aby znacznik `section` odzyskał swoją pełną szerokość. Ponieważ `section` teraz sam jest elementem pływającym, znacznik `footer` będzie się starał znaleźć obok niego. Trzeba zatem zmusić `footer`, by pozostał pod elementem `section`, dodając mu właściwość `clear:left`. Oczyszczony element nie może przesunąć się do góry, by znaleźć się obok pływającego elementu. Zastosowanie tego kodu przynosi taki sam efekt, jak przedstawiono na **rysunku 3.20**.

METODA 3. — DODAJ NIEPŁYWĄCY ELEMENT OCZYSZCZAJĄCY

Trzeci sposób na zmuszenie rodzica do objęcia swoich pływających dzieci polega na dodaniu niepływającego elementu, który byłby ostatnim dzieckiem i oczyszczał go. Ponieważ kontener zawsze obejmuje niepływające elementy, to poza ostatnim dzieckiem obejmie również poprzedzające je pływające elementy. Element oczyszczający jako ostatnie dziecko kontenera można utworzyć na dwa sposoby.

Pierwszym, choć nie idealnym sposobem jest umieszczenie bezpośrednio w kodzie elementu HTML jako ostatniego dziecka i nadanie mu właściwości CSS `clear`; najlepszy jest do tego znacznik `div`, ponieważ nie ma żadnego domyślnego stylu, a zatem nie tworzy w obrębie layoutu dodatkowej przestrzeni.

```
<section>
  
  <p>Fajnie sobie pływać.</p>
  <div class="clear_me"></div>
</section>
<footer> Oto stopka, która...</footer>
```

Elementowi `div` nadałem tutaj klasę, aby móc go oczyścić w kodzie CSS:

```
section {border:1px solid blue;}
img {float:left;}
.clear_me {clear:left;}
footer {border:1px solid red;}
```

Pływające elementy są teraz zawarte w kontenerze, tak jak widać na **rysunku 3.20**. Jeśli wolisz unikać czysto prezentacyjnych elementów jak ten, to możesz stworzyć element oczyszczający przy użyciu samego kodu CSS. Najpierw należy przypisać znacznikowi `section` klasę:

```
<section class="clearfix">
  
  <p>Fajnie sobie pływać.</p>
</section>
<footer> Oto stopka, która...</footer>
```

oraz użyć magicznego kodu CSS `clearfix!`

```
.clearfix:after {
  content: ".";
  display:block;
  height:0;
```



Podany tutaj kod `clearfix`, opracowany przez programistę

Tony'ego Asletta, dodaje oczyszczony, niepływający element, który zawiera jedynie kropkę (jakaś treść musi być obecna, a kropka to najmniej, jak się da). Kilka dodatkowych deklaracji sprawia, że ten pseudoelement nie dodaje wysokości do layoutu i nie jest widoczny na stronie.



Wartość `both` właściwości `clear` oznacza, że element `section` oczyszcza (czyli znajduje się poniżej) elementów płynących zarówno w lewą, jak i w prawą stronę. W tym przypadku mogę użyć wartości `left`, ale dziękianiu `both` — jeśli później zmienię wartość `float` obrazów na `right` — `clear` wciąż będzie działać.

```
visibility:hidden;  
clear:both;  
}
```

Elementy płynające także w tym przypadku zostają zawarte w kontenerze, tak jak na **rysunku 3.20**, ale tym razem bez zamieszczenia dodatkowego elementu w kodzie HTML. Możesz tymczasowo usunąć deklaracje wysokości i widoczności w przedstawionym powyżej kodzie `clearfix`, żeby zobaczyć kropkę, która zostaje dodana do kodu.

Kodu CSS `clearfix` używam do rozwiązywania tego typu problemów z płynącymi elementami praktycznie na wszystkich moich stronach, jako że tworzenie takich elementów (dopóki więcej przeglądarek nie zacznie obsługiwać specyfikacji modułu CSS3 Columns) jest jedynym pewnym sposobem tworzenia kolumn.

Podsumowując, istnieją trzy sposoby na zmuszenie rodziców do zawarcia w sobie swoich „pływających” dzieci:

- Nadanie rodzicowi właściwość `overflow:hidden`.
- Przekształcenie rodzica w element płynający.
- Dodanie niepływającego elementu jako ostatniego dziecka rodzica, albo poprzez umieszczenie go w kodzie, albo poprzez nadanie rodzicowi klasy `clearfix` (choć oczywiście musisz wtedy zamieścić odpowiedni kod CSS `clearfix` w arkuszu stylów).

Wybór metody zależy od sytuacji. Metody z `overflow:hidden` nie można użyć na najwyższym poziomie rozwijanego menu, bo dalsze rozwijane podmenu nie będą się wyświetlać. Wynika to z tego, że rozwijane menu wyświetlają się poza obszarem rodzica, czemu `overflow:hidden` ma konkretnie zapobiegać. Metody przekształcania rodzica w płynący element nie można użyć w przypadku elementu wyśrodkowanego marginesami o wartości `auto`, ponieważ ów element zostaje wtedy przesunięty w prawo lub w lewo, w zależności od podanej wartości. Musisz zatem umieć się posłużyć wszystkimi trzema z tych technik, aby móc się odnieść do wszystkich sytuacji, w których konieczne jest zawarcie płynących elementów w kontenerze.

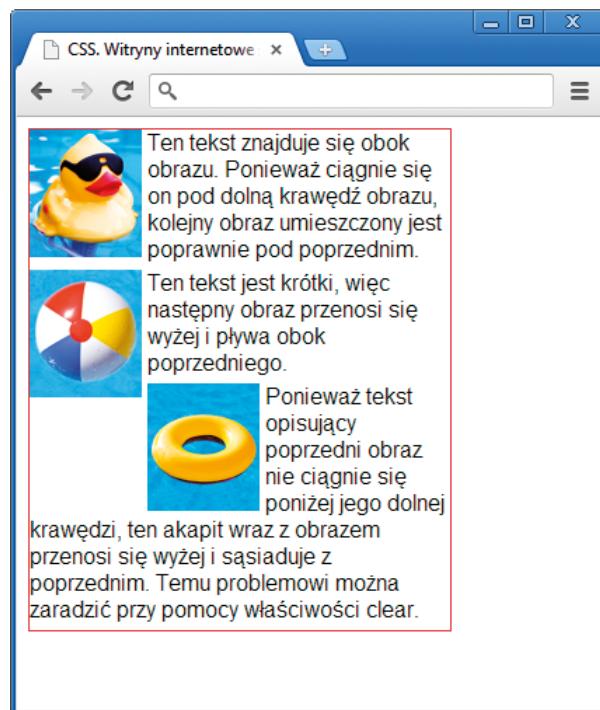
ZASTOSOWANIE WŁAŚCIWOŚCI CLEAR BEZ KONTENERA

Czasami trzeba oczyścić płynające elementy, które nie mają jakiegokolwiek wygodnego kontenera. Najprostszym sposobem jest nadanie kodu CSS `clear:both` elementowi, który przenosi się do góry, aby go zmusić, by pozostał pod płynającym elementem. Kiedy jednak wy-

starczy miejsca, by więcej niż jeden element przeniósł się do góry, to proste podejście może nie zadziałać, zmuszając Cię do zastosowania bardziej twórczych środków.

W ramach przykładu, na **rysunku 3.21** widnieje layout składający się z sześciu elementów: trzech obrazów z sąsiadującymi opisami. Taki layout można uzyskać, przekształcając obrazy w elementy pływające, aby znajdujące się w kodzie pod kolejnymi obrazami teksty przenosiły się w górę, by sąsiadować z pływającymi obrazami.

RYSUNEK 3.21. Ponieważ staczy tu miejsca, trzeci obraz wraz z tekstem może przemieścić się i znaleźć się w sąsiedztwie drugiego obrazu, co nie jest pożdanym efektem



Oto kod HTML z poprzedniego rysunku (ze skróconym tekstem, żeby oszczędzić miejsca):

```
<section>
  
  <p>Ten tekst znajduje się obok obrazu...</p>
  
  <p>Ten tekst jest krótki, więc następny obraz...</p>
  
  <p>Ponieważ tekst opisujący poprzedni obraz nie...</p>
</section>
```

któremu nadaję następujący kod CSS:

```
section {width:300px; border:1px solid red;}  
img {float:left; margin:0 4px 4px 0;}
```

pomijam deklaracje fontów —————| p {margin:0 0 5px 0;}

dla zwięzłości

Celem było sprawienie, żeby każdy blok tekstowy znajął się obok obrazu, do którego przynależy. Ponieważ jednak tekst drugiego akapitu nie jest wystarczająco długi, by wyjść poza dolną krawędź drugiego płynnego obrazu, obecność wolnej przestrzeni pozwala kolejnej parze obrazu i tekstu na przejście w górę.

W poprzednim przykładzie layout wydaje się całkowicie poprawnie sformułowany: trzecia para obrazu i tekstu ma miejsce, żeby znaleźć się obok płynnego elementu. I tam też zostaną umieszczone, ponieważ płynność elementów z założenia ma sprawiać, by elementy znajdowały się możliwie wysoko i na skraju (z lewej lub z prawej, w zależności od wartości właściwości `float`). Pod względem graficznym rezultat nie jest jednak taki, jaki być powinien.

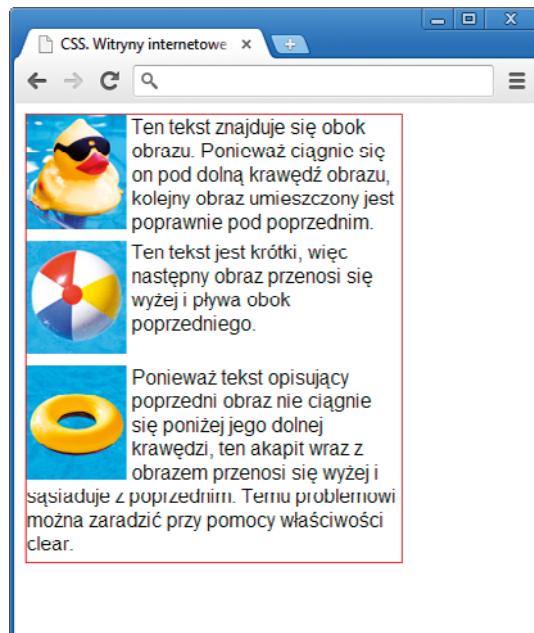
Ponieważ wokół poszczególnych par obrazów i akapitów nie ma kontenerów, nie mogę skorzystać z technik zmuszania rodzica do okalania swoich dzieci z poprzednich przykładów. Mogę jednak użyć kodu CSS `clearfix`

```
.clearfix:after {  
    content:". .";  
    display:block;  
    height:0;  
    visibility:hidden;  
    clear:both;  
}
```

w ten sposób:

```
<section>  
      
    <p class="clearfix">Ten tekst znajduje się obok obrazu...</p>  
      
    <p class="clearfix">Ten tekst jest krótki, więc następny  
    obraz...</p>  
      
    <p class="clearfix">Ponieważ tekst opisujący poprzedni  
    obraz nie...</p>  
</section>
```

RYSUNEK 3.22. Po dodaniu elementu „oczyszczającego” klasą `clearfix` layout wyświetlany jest poprawnie



Jak widać na rysunku 3.22, „oczyszczające” elementy zostają dodane do kodu po każdym akapicie. Ponieważ trzecia para obrazu i akapitu znajdują się po jednym z tych oczyszczonych elementów, nie może ona już przejść do góry, dzięki czemu uzyskujemy pożądaną postać layoutu. Klasę `clearfix` nadałem *wszystkim* akapitom, nie tylko drugiemu, który potrzebował tego w tym przykładzie. Służy to zilustrowaniu tego, co zrobiłbym, gdybym tworzył prawdziwą stronę — gdyby w przyszłości tekst któregoś z tych akapitów był krótszy od wysokości obrazu, layout dzięki temu by się nie rozłożył.

Skoro już wiesz, jak działają właściwości `float` i `clear`, zakończmy ten rozdział, przyglądając się dwóm pozostałym właściwościom, które są kluczowe dla tworzenia layoutów w CSS: `position` i `display`.

Właściwość position

Właściwość **position** leży u podstaw wszystkich layoutów opartych na CSS; definiuje położenie pola elementu względem tego, gdzie normalnie znajdowałoby się w ramach ciągu strukturalnego dokumentu. Właściwość **position** obsługuje cztery wartości: **static**, **relative**, **absolute** i **fixed**, przy czym pierwsza z nich jest domyślna. Zastosowanie każdej z nich zaprezentuję Ci na podstawie poniższego kodu z czterema akapitami.

```
<p>Pierwszy akapit</p>
<p>Drugi akapit</p>
<p id="specialpara">Trzeci akapit (z identyfikatorem)</p>
<p>Czwarty akapit</p>
```

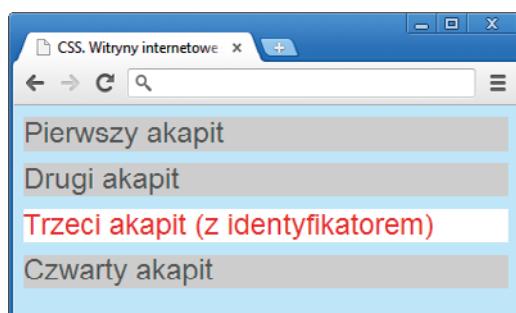
W każdym przykładzie akapit pierwszy, drugi i czwarty będą znajdująć się w domyślnym położeniu **static**, a wartość właściwości **position** akapitu trzeciego będzie się zmieniać.

Trzeci akapit oznaczyłem identyfikatorem **specialpara**, aby móc zmienić jego właściwość **position**, nie wpływając na pozostałe akapity.

Pozycjonowanie statyczne

Przyjrzyjmy się najpierw czterem akapitom w domyślnym położeniu **static** (rysunek 3.23).

RYSUNEK 3.23. Położenie **static** sprawia, że elementy blokowe ustawiają się zgodnie z domyślnym ciągiem strukturalnym dokumentu



W położeniu statycznym każdy element zamieszczony jest w standardowym ciągu dokumentu — mamy tu do czynienia z elementami blokowymi, które rozmiędzzone są na stronie jeden pod drugim. Także nieobstyłowane layouty HTML, które pokazałem w rozdziale 1., wyświetlane są w położeniu określonym wartością **static**.

Aby oderwać się od tej standardowej kolejności rozmiędzenia elementów, musisz zmienić wartość właściwości **position** pola na któryś z pozostałych.

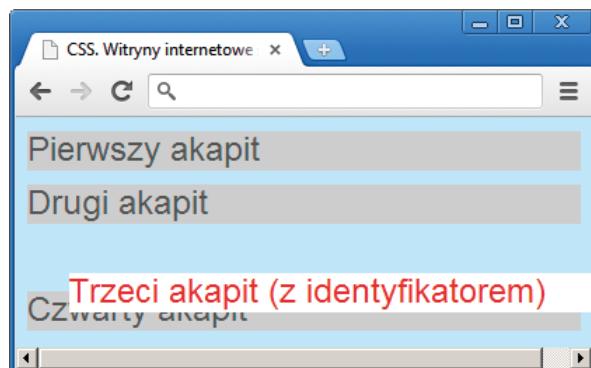
Pozycjonowanie względne

Nadajmy teraz wartości `position` trzeciego akapitu wartość `relative`. Samo to nie wywołuje widocznej zmiany, ale po nadaniu elementowi położenia względnego można go przesuwać względem jego domyślnego położenia przy użyciu właściwości `top`, `right`, `bottom` i `left`. W większości przypadków wystarczy podać wartości `top` i `left`, by uzyskać pożądany rezultat. Przykładowy kod

```
#specialpara {position: relative; top:25px; left:30px;}
```

pozwala na uzyskanie rezultatu widocznego na **rysunku 3.24**.

RYSUNEK 3.24. Pozycjonowanie względne pozwala na zastosowanie właściwości `top` i `left` do przesunięcia elementu względem jego domyślnego położenia w ciągu strukturalnym dokumentu



Akapit zostaje teraz przesunięty w dół o 25 pikseli i w prawo o 30 pikseli względem swojego domyślnego położenia w ciągu strukturalnym dokumentu, co wysuwa go poza obszar kontenera `body`, tym samym zamieszczając jego fragment poza ekranem. Choć jednak element przesuwa się względem swojego domyślnego położenia, nic poza tym nie ulega zmianie. Miejsce pierwotnie zajęte przez element pozostaje zachowane, tak samo jak pozycjonowanie pozostałych elementów.

 Właściwościom `top` i `left` możesz również nadawać ujemne wartości, by przesunąć element w górę i w lewo.

Wniosek z tego taki, że kiedy przenosisz element w ten sposób, musisz zapewnić mu jakąś przestrzeń. W przykładzie z **rysunku 3.24** możesz wykonać kolejny krok i dodać właściwość `margin-top` o wartości `30px` lub wyższej do czwartego akapitu, by przesunąć go w dół, a tym samym sprawić, by nie nachodził na niego przesunięty trzeci akapit.

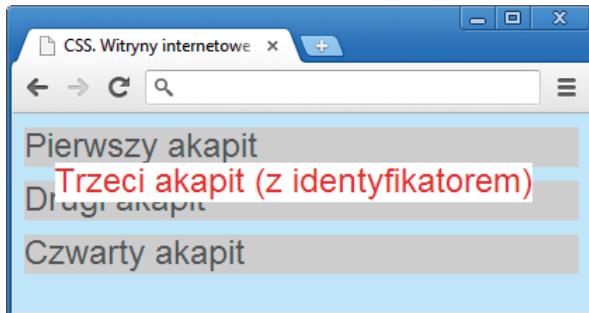
Pozycjonowanie bezwzględne

Pozycjonowanie bezwzględne zupełnie różni się od statycznego i względnego, jako że całkowicie wyciąga ono element z ciągu strukturalnego dokumentu. Zmieńmy kod zastosowany w przykładzie pozycjonowania względnego, zmieniając wartość `relative` na `absolute`:

```
p#specialpara {position: absolute; top:25px; left:30px;}
```

Rezultat widać na **rysunku 3.25**.

RYSUNEK 3.25. Pozycjonowanie bezwzględne pozwala na usunięcie elementu z ciągu strukturalnego dokumentu i określenie jego położenia względem obiektu — w tym przypadku w odniesieniu do domyślnego kontekstu pozycjonowania, czyli elementu `body`



Na **rysunku 3.25** widać, że miejsce zajmowane wcześniej przez element zniknęło. Bezwzględnie pozycjonowany element został całkowicie usunięty z ciągu strukturalnego dokumentu, a jego położenie określone jest teraz względem elementu najwyższego poziomu, czyli `body`. W ten sposób dochodzimy do istotnej kwestii kontekstu pozycjonowania.

Zacznijmy od tego, że domyślnym kontekstem pozycjonowania elementu pozycjonowanego bezwzględnie jest element `body`. Jak widać na **rysunku 3.25**, odległość podana wartościami `top` i `left` przesuwa pozycjonowany bezwzględnie element w odniesieniu do elementu `body` — najdalszego przodka w hierarchii kodu — a nie w odniesieniu do domyślnego położenia elementu w ciągu strukturalnym dokumentu, tak jak dzieje się to w przypadku pozycjonowania względnego.

Ponieważ kontekstem pozycjonowania elementu pozycjonowanego bezwzględnie jest element `body`, zmienia się on, kiedy strona jest przewijana. Dzieje się tak, aby zachować relację przestrzenną elementu względem `body`, który również przesuwa się podczas przewijania.

Zanim pokażę Ci, jak użyć elementu innego niż `body` w charakterze kontekstu pozycjonowania elementu pozycjonowanego bezwzględnie, omówię jeszcze ostatni z czterech rodzajów pozycjonowania — stały.

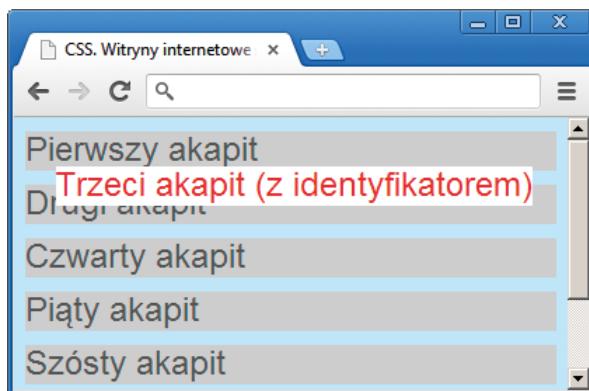
Pozycjonowanie stałe

Pozycjonowanie stałe przypomina bezwzględne w tym, że element zostaje całkowicie usunięty z ciągu strukturalnego dokumentu.

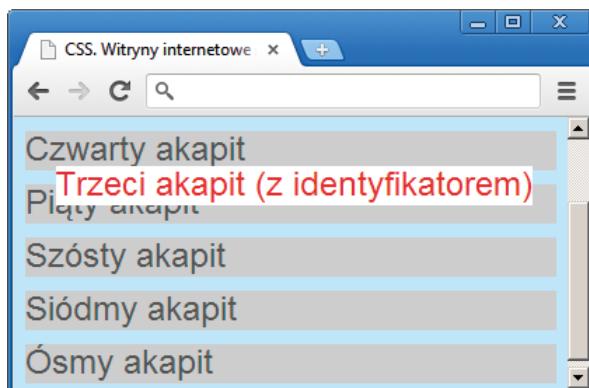
```
p#specialpara {position: absolute; top:30px; left:20px;}
```

Różnica jest taka, że kontekstem pozycjonowania elementu pozycjonowanego w ten sposób jest obszar widoku (np. okno przeglądarki lub ekran urządzenia przenośnego), wobec czego element nie przesuwa się wraz z przewijaniem strony. Na **rysunkach 3.26 i 3.27** widać rezultat zastosowania pozycjonowania stałego.

RYSUNEK 3.26. Pozycjonowanie stałe przypomina pozycjonowanie bezwzględne...



RYSUNEK 3.27. ...dopóki nie przewiniesz strony — pozycjonowany element nie przesuwa się



Z pozycjonowania stałego nie korzysta się zbyt często. Na ogół służy do tworzenia elementów nawigacyjnych, które mają pozostać w miejscu podczas przewijania strony.

Skoro już znasz różnice między czterema wartościami właściwości `position`, powiedzmy sobie nieco więcej o kontekście pozycjonowania.

Kontekst pozycjonowania

Kiedy zmieniasz wartość właściwości `position` elementu na `relative`, `absolute` lub `fixed`, a następnie przesuwasz go właściwościami `top`, `right`, `bottom` lub `left`, zmieniasz jego położenie względem innego elementu. Ten inny element nazywamy właśnie kontekstem pozycjonowania.

Jak dowiedziałeś się z punktu „Pozycjonowanie bezwzględne”, domyślnym kontekstem pozycjonowania elementu pozycjonowanego bezwzględnie jest `body`. Jest tak, ponieważ `body` to jedyny element, który jest przodkiem wszystkich znaczników w kodzie. Możesz jednak użyć dowolnego przodka pozycjonowanego bezwzględnie elementu w charakterze kontekstu pozycjonowania, nadając właściwości `position` przodka wartość `relative`.

Przyjrzyjmy się kodowi HTML

Tekst musi być poprawnie oznaczony elementem w rodzaju paragrafu, aby zdefiniować go semantycznie. W tym przykładzie w celu zachowania przejrzystości zamieściłem go jednak bezpośrednio w wewnętrznym znaczniku `div`.

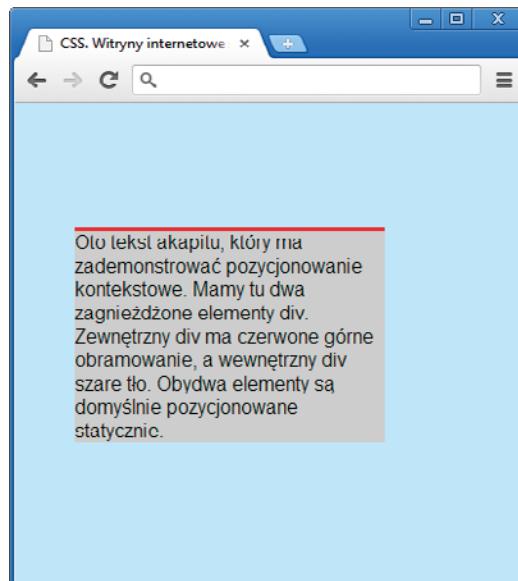
```
<body>
  <div id="outer">
    <div id="inner">Oto tekst...</div>
  </div>
</body>
```

i następującym regułom CSS:

```
div#outer {width:250px; margin:50px 40px; border-top:3px solid red;}
div#inner {top:10px; left:20px; background:#ccc;}
```

Ponieważ w kodzie podano przesunięcie wewnętrznego znacznika `div` względem góry i lewej strony, możesz się zastanawiać, dlaczego na **rysunku 3.28** wewnętrzny `div` nie jest przesunięty w dół względem górnej krawędzi zewnętrznego znacznika o 10 pikseli oraz o 20 pikseli w lewo, tak jak zdawałoby się z tego wynikać. Zamiast tego obydwa elementy mają wspólny punkt początkowy (czyli lewy górnny róg). Otóż chodzi o to, że wewnętrzny (a także zewnętrzny, ale to nieistotne) element `div` jest pozycjonowany domyślnie, czyli zgodnie z wartością `static`. Oznacza to, że jest częścią standardowego ciągu strukturalnego dokumentu, a ponieważ zewnętrzny `div` nie zawiera żadnej treści, `div` wewnętrzny zaczyna się w tym samym miejscu, co on. Dopiero kiedy nadajesz elementowi jedną z pozostałych trzech wartości pozycjonowania — `relative`, `absolute` lub `fixed` — właściwości `top`, `right`, `bottom` i `left` rzeczywiście zaczynają działać. Zobaczmy to zachowanie w działaniu, nadając właściwości `position` wewnętrznego elementu `div` wartość `absolute`.

RYSUNEK 3.28. Oto dwa zagnieżdżone elementy div. Górną krawędzi zewnętrznego elementu nadalem czerwone obramowanie, a wewnętrzny element oznaczyłem kolorem szarym. Ponieważ wewnętrzny div pozycjonowany jest statycznie (czyli domyślnie), właściwości top i left są ignorowane

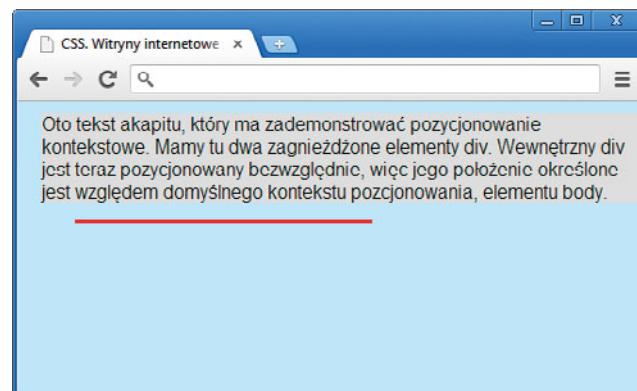


 Jeśli uważasz, że możesz zignorować właściwości position, to na ogół do tworzenia layoutów stron wystarczy korzystać z pozycjonowania statycznego. Wielu początkujących programistów CSS niesuszenie zmienia właściwość position niemal wszystkich elementów, a później odkrywa, że trudno odzyskać kontrolę nad wszystkimi wyzwolonymi w ten sposób elementami. Nie zmieniaj wartości position elementu z domyślnej static, chyba że naprawdę musisz.

RYSUNEK 3.29. Choć wewnętrzny div (oznaczony kolorem szarym) znajduje się w kodzie wewnętrznej zewnętrznej elementu div (który można poznać po czerwonym obramowaniu górnym), jego bezwzględne pozycjonowanie wraz z tym, że brakuje innego pozycjonowanego względnie elementu, którego mógłby użyć jako kontekstu, powoduje, że pozycjonowany jest względem elementu body

```
div#outer {width:250px; margin:50px 40px; border-top:3px solid red;}
div#inner {position: absolute; top:10px; left:20px; background:#ccc;}
```

Względem czego jednak zachodzi tutaj pozycjonowanie bezwzględne? Ponieważ nie ma żadnego pozycjonowanego względnie elementu, do którego pozycjonowany div mógłby się odnosić, to przesuwa się on domyślnie względem elementu body. Jest tak, ponieważ body jest domyślnym kontekstem pozycjonowania. Wewnętrzny div całkowicie ignoruje swojego rodzica (czyli zewnętrzny div), a jego właściwości top i left odsuwają go od elementu body, co widać na rysunku 3.29.

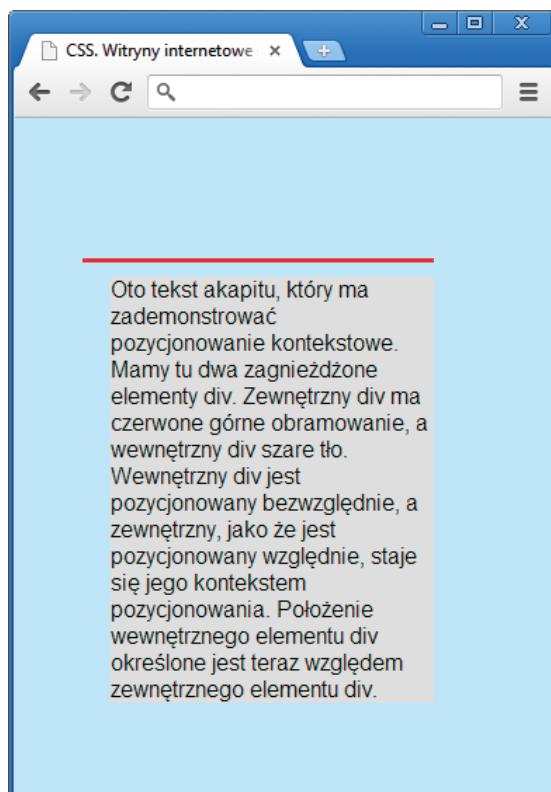


Jeżeli zmienimy teraz wartość `position` zewnętrznego znacznika `div` na `relative`

```
div#outer {position:relative; width:250px; margin:50px 40px; border-top:3px solid red;}  
div#inner {position:absolute; top:10px; left:20px; background:#ccc;}
```

to kontekstem pozycjonowania pozycjonowanego bezwzględnie wewnętrznego znacznika `div` stanie się teraz zewnętrzny element `div`, tak jak widać na **rysunku 3.30**.

RYSUNEK 3.30. Kiedy zewnętrzny `div` pozycjonowany jest względnie, jego pozycjonowanie bezwzględnie potomkowie zostają rozmieszczeni względem niego, zgodnie z wartościami ich właściwości `top` i `left`



Właściwości `top` i `left` wewnętrznego znacznika `div` przesuwają go teraz względem zewnętrznego elementu `div`. Gdybyś teraz przesunął zewnętrzny `div`, nadając jego właściwościom `left` i `top` wartość inną niż zerową, wewnętrzny `div` przesunąłby się o tę samą odległość, by zachować relację przestrzenną z elementem zewnętrznym — jego kontekstem pozycjonowania.

Właściwość `display`

Tak samo jak z właściwością `position`, każdy element ma właściwość `display`. Choć istnieją różne wartości `display`, większość elementów ma domyślną wartość `block` lub `inline`. Jeśli zapomniałeś to, czego nauczyłeś się z rozdziału 1., przypominam różnice między elementami blokowymi i liniowymi:

- Elementy blokowe, takie jak akapity, nagłówki i listy, wyświetlane są w przeglądarce jeden nad drugim.
- Elementy liniowe, w rodzaju `a`, `span` i `img`, wyświetlane są w przeglądarce obok siebie w poziomie i przechodzą do kolejnego wiersza dopiero, kiedy zaczyna brakować miejsca w poprzednim.

Możliwość przekształcania elementów blokowych w liniowe i na odwrót

domyślnie element blokowy  `p {display:inline;}`

domyślnie element liniowy  `a {display:block;}`

jest cenna, pozwalając m.in. na zmuszenie elementu liniowego do wypełnienia swojego kontenera. Zrobię to później z odnośnikami przy tworzeniu rozwijanych menu w CSS.

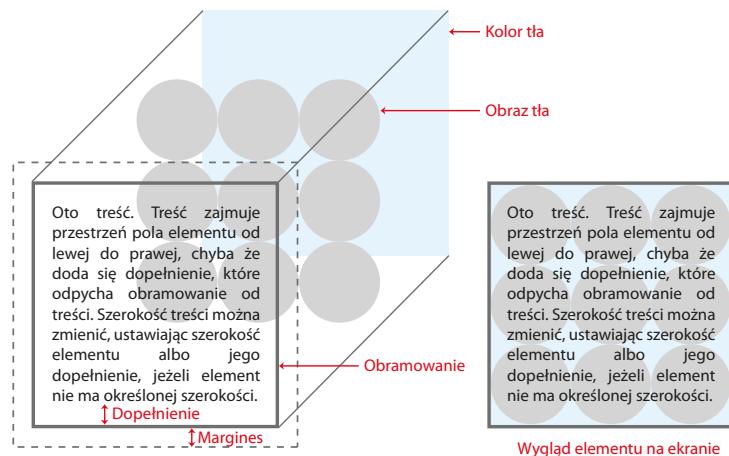
Warto wspomnieć tu jeszcze o jednej wartości `display` — `none`. Kiedy właściwość `display` elementu ma wartość `none`, to element ten oraz wszelkie elementy w nim osadzone nie są wyświetlane na stronie. Miejsce zajmowane przez element znika. Działa to tak, jakby kod elementu w ogóle nie istniał. Działa to inaczej niż właściwość `visibility`, której najprzydatniejsze wartości to `visible` (czyli wartość domyślna) i `hidden`. Kiedy `visibility` elementu ma wartość `hidden`, element zostaje ukryty, ale zajmowany przez niego obszar nie zostaje usunięty.

Tła

Ostatnim aspektem związanym z pozycjonowaniem elementów są tła, które umożliwiają dodawanie kolorów bądź obrazów w tleach elementów. Jeżeli pracowałeś z programami graficznymi w rodzaju Adobe Photoshop czy Adobe Fireworks, to znane jest Ci pojęcie warstw. Pole każdego elementu można podzielić na dwie warstwy. Warstwa pierwszego planu składa się z treści elementu (np. tekstu lub obrazu) oraz obramowania pola. Warstwę tła elementu można wypełnić jednolitym kolorem przy użyciu właściwości `background-color` albo zawrzeć w niej dowolną liczbę obrazów przy użyciu właściwości `background-image`, która zamieszcza obrazy na kolorze tła.

Zanim pojawił się CSS3, warstwie tła można było jedynie nadać kolor i zamieścić na niej jeden obraz. Teraz możesz umieszczać na niej wiele obrazów (oraz nowe gradienty CSS3). Wróćmy teraz do diagramu modelu polowego z poprzedniej części rozdziału i ukażmy go w trzech wymiarach, żeby zobrazować także warstwę tła elementu (**rysunek 3.31**).

RYSUNEK 3.31. Ten diagram modelu polowego ukazuje części składowe pierwszego planu i warstwy tła elementu



Właściwości tła CSS

Oto właściwości tła CSS:

- `background-color`,
- `background-image`,
- `background-repeat`,
- `background-position`,
- `background-size`,
- `background-attachment`,
- `background` (zbiorczy) ,
- `background-clip`, `background-origin`, `background-break` (obsługiwane obecnie niezbyt dobrze).

Te właściwości dają Ci rozległą kontrolę nad elementami tła. Omówmy je po kolejni.

Kolor tła

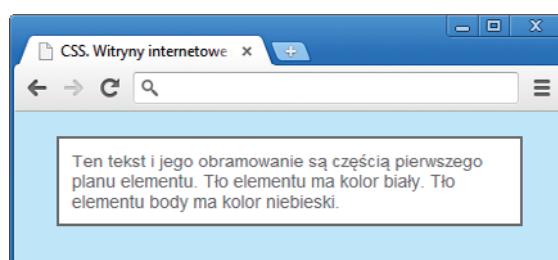
Właściwość `background-color` jest najprostszą z właściwości tła.

Przy jej użyciu określasz kolor, którym zostaje wypełniona warstwa tła, tak jak widać na [rysunku 3.32](#).

```
style pola elementu ——————→ body {background-color:#caebff;}

ukazane w tym przykładzie ——————→ p {font-family:helvetica, arial, sans-serif; font-size:18px;
style tła i pierwszego planu                width:350px; margin:20px auto; padding:10px;
                                                    background-color:#fff; color:#666; border:4px solid;}
```

RYSUNEK 3.32. Właściwości `background-color` elementu `body` nadalem kolor niebieski. Właściwości `background-color` akapitu nadalem kolor biały, a jego właściwości `color`, która określa kolor pierwszego planu, wpływający zarówno na obramowanie, jak i na tekst, kolor szary



W tym przykładzie pokazałem nie tylko, jak nadać elementowi kolor tła, ale również wskazałem, że warstwa pierwszego planu składa się zarówno z treści właściwej, jak i z obramowania. Kiedy używasz właściwości `border-color` do określenia stylu i szerokości obramowania elementu, ale nie jego koloru, to jego barwa określona jest właściwością `color`, która również definiuje kolor tekstu. Domyślny kolor to czarny. Jeśli chcesz, by obramowanie i tekst miały inne barwy, musisz im je nadać osobno.

Obraz tła

Oto obraz z małym kółkiem, którego użyję do zilustrowania zastosowania właściwości `background-image` i `background-repeat` ([rysunek 3.33](#)).

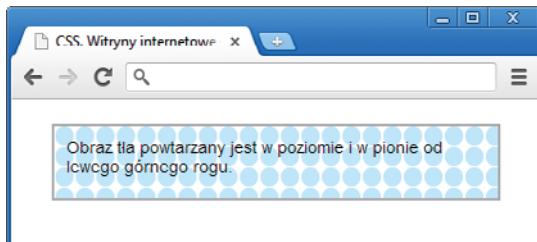
RYSUNEK 3.33. Obraz małego kółka otoczyłem tutaj ramką, żeby było widać pustą przestrzeń wokół figury



Zacznę od zastosowania właściwości `background-image`, by umieścić obraz kółka w tle elementu, tak jak widać na [rysunku 3.34](#).

```
p {font-size:28px; font-family:helvetica, arial, sans-serif;  
width:345px; height:110px; margin:20px auto; padding:10px;  
color:#000; border:4px solid #aaa; background-color:#fff;  
background-image:url(images/blue_circle.png);}
```

RYSUNEK 3.34. Obraz tła, który jest mniejszy od elementu, powtarzany jest w poziomie i pionie, aby go wypełnić



Jak widać na poprzednim rysunku, obraz jest domyślnie powtarzany w poziomie i w pionie od lewego górnego rogu elementu, aż do wypełnienia go. Ze względu na to, że punkt początkowy znajduje się w lewym górnym rogu, kółka u dołu i z prawej zostają obcięte w miejscu określonym przez wysokość i szerokość pola.

Zauważ, że obraz tła podaje się inaczej niż obraz w znaczniku `img`, w formacie

```
background-image:url(lokalizacjaObrazu/nazwaObrazu)
```

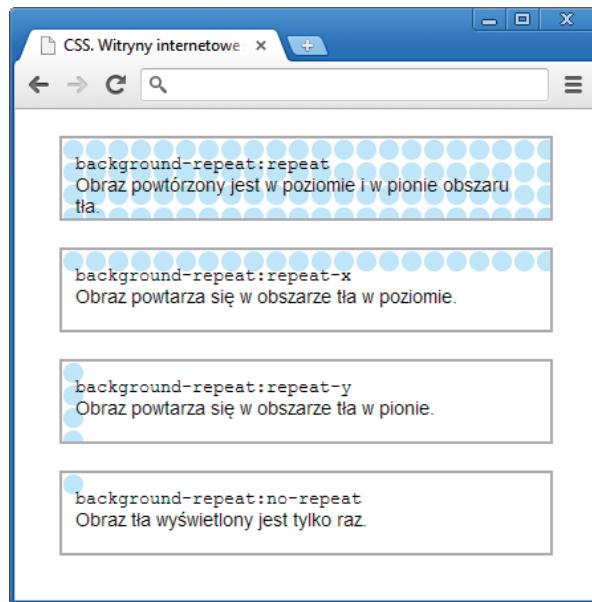
Nie musisz zamieszczać adresu obrazu w cudzysłowie, choć możesz. Domyślne ustawienia powtórzeń w poziomie i w pionie oraz punkt początkowy możesz zmienić, odpowiednio, właściwościami `background-repeat` i `background-position`, którym się teraz przyjrzymy.

Powtórzenia obrazu tła

Właściwości `background-repeat` można nadać jedną z czterech wartości. Domyślna to `repeat`, która, jak widziałeś na poprzednim rysunku, powtarza obraz w poziomie i w pionie tyle razy, by zapełnić element. Pozostałe wartości to `repeat-x`, która powtarza obraz w poziomie, `repeat-y`, która powtarza obraz w pionie, oraz `no-repeat`, która sprawia, że obraz wyświetlony jest tylko raz. Zachowania tych wartości zilustrowane są na [rysunku 3.35](#).

Z tych opcji powtórzeń można korzystać w różnych celach. Dzięki wartościom `repeat-x` i `repeat-y` możesz w bardzo łatwy sposób dodawać powtarzające się ozdobne obrazy składające się na ramki elementów, a `no-repeat` możesz użyć do stworzenia tła

RYSUNEK 3.35. Cztery wartości background-repeat



składającego się z jednego obrazu. Dodatkową kontrolę nad właściwościami tła, które dotąd przedstawiłem, zapewnia właściwość `background-position`, której się zaraz przyjrzymy.

Na koniec wspomnę jeszcze o tym, że CSS3 oferuje na razie nieobsługiwane funkcje, które pozwalają na określanie konkretnej liczby powtórzeń:

- `background-repeat:round` skaluje obraz tak, by został powtórzony określoną liczbę razy.
- `background-repeat:space` dodaje odstępy między obrazami, aż równomiernie wypełnią element.

Położenie tła

Właściwość `background-position` to prawdopodobnie jedna z najbardziej skomplikowanych właściwości tła. Pięć podstawowych ustawień `background-position` to słowa kluczowe `top`, `left`, `bottom`, `right` i `center`, a jako wartość możesz podać dowolne dwa z nich. Jeżeli podasz `top right`, to prawy górnego róg obrazu zostanie umieszczony w prawym górnym rogu elementu. Jeżeli podasz `center center`, to środek obrazu zostanie umieszczony pośrodku elementu. Pozwolę to sobie opisać dokładniej.

Właściwość `background-position` jednocześnie określa punkt początkowy elementu i obrazu. Punkt początkowy określa współrzędne pionowe i pionowe punktu w obrębie elementu i obrazu, wobec

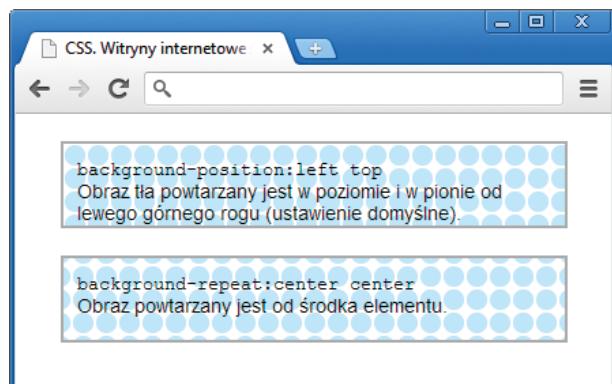
którego zostają one rozmieszczone. Domyślne wartości punktu początkowego właściwości `background-position` to `top` i `left`. Jeśli więc nie zdefiniujesz właściwości `background-position` obrazu tła, lewy górny róg obrazu zostanie umieszczony w lewym górnym rogu elementu, od którego zacznie się także powtarzanie obrazu. Domyślne położenie zostało przedstawione w czterech przykładach na poprzednim rysunku.

Wiedząc wszystko powyższe, przyjrzyjmy się właściwości `background-position` w działaniu i przyjrzyjmy się znowu pierwszemu z przykładów na poprzednim obrazie, w którym powtórzenia poziome i pionowe określone są domyślnymi ustawieniami właściwości `background-position`. Porównajmy to z działaniem właściwości `background-position` o wartościach `center center` ([rysunek 3.36](#)).

zbiorcza deklaracja `center center` → `p#center {background-position:center;}`

RYSUNEK 3.36. Kiedy położenie tła określone jest właściwościami `center center`, obraz jest wyśrodkowany w elemencie i powtarzany jest w każdym kierunku

 Jeśli podane jest tylko jedno słowo kluczowe `background-position`, tak jak w tym przykładzie, to domyślną drugą wartością jest `center`.



Jak widać z porównania obydwu przykładów na powyższym rysunku, w drugim przykładzie obraz powtarzany jest od środka we wszystkich kierunkach.

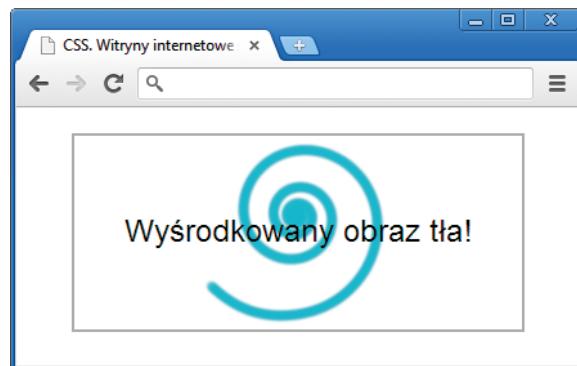
Pośrodku elementu umieszę teraz większy obraz. Tym razem do określenia położenia obrazu użyję wartości procentowych ([rysunek 3.37](#)).

```
div {height:150px; width:250px; border:2px solid #aaa;  
margin:20px auto; background-image:url(images/turq_spiral_150.png);  
background-repeat:no-repeat; background-position:50% 50%;}
```

Nadając właściwości `background-position` wartości `50% 50%`, a `background-repeat` wartość `no-repeat`, umieszczam obraz pośrodku warstwy tła.

RYSUNEK 3.37. Obraz tła ułożony pośrodku elementu przy użyciu właściwości `background-position`

 Tekst tego elementu wyśrodkowalem w pionie, nadając właściwości `line-height` tekstu wysokość elementu. Interlinia jest rozkładana równomiernie powyżej tekstu i pod nim. Nadałem też właściwości `text-align` tekstu wartość `center`, aby wyśrodkować go także w poziomie — wyśrodkowując go tym samym w obydwu wymiarach, tak samo jak obraz.



Wielkość tła

Teraz, choć niezbyt dobrze obsługiwana właściwość CSS3 `background-size` daje Ci kontrolę nad wielkością obrazu tła. Poniżej znajdziesz przykłady wartości tej właściwości.

Wartości położenia tła

Istnieją trzy rodzaje wartości, którymi możesz określić położenie obrazów tła: słowa kluczowe, wartości procentowe oraz bezwzględne i względne wartości liczbowe w rodzaju pikseli lub procentów. Położeniu poziomemu i pionowemu można nadać dwie osobne wartości.

Słowa kluczowe można podawać w dowolnej kolejności; `left bottom` i `bottom left` dają jednakowy rezultat. W celu uzyskania możliwie największej kompatybilności z przeglądarkami lepiej nie mieszać słów kluczowych z wartościami liczbowymi.

Kiedy podajesz wartości liczbowe takie jak `40% 30%`, pierwsza z nich określa położenie w poziomie, a druga w pionie. Kiedy podajesz tylko jedną wartość, zostaje ona wykorzystana do określenia położenia w poziomie, a na określenie położenia w pionie przyjmuje się wartość `center`.

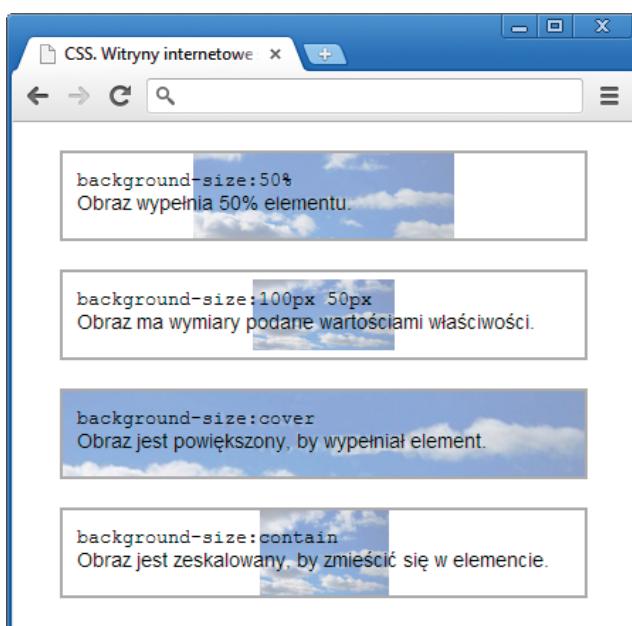
Kiedy określasz położenie słowami kluczowymi lub wartościami procentowymi, są one nadawane zarówno elementowi, jak i obrazowi. Innymi słowy, jeśli podasz wartości `33% 33%`, to punkt znajdujący się w 33% szerokości obrazu umieszczany jest w punkcie znajdującym się w 33% szerokości elementu. To samo odnosi się do rozmieszczania w pionie. Widziałeś już zresztą, że wartości `center center` umieszczały środek obrazu pośrodku elementu.

Wartości bezwzględne w rodzaju pikseli działają inaczej. Określając położenie pikselami, sprawiasz, że lewa i góra krawędź obrazu umieszczana jest w podanej odległości od lewego górnego rogu elementu.

Möżesz także podawać odległości ujemne, aby umieścić lewą bądź górną krawędź obrazu poza elementem, by ukazać w jego obrębie tylko jego fragment. Taki sam rezultat można osiągnąć, podając wystarczająco wysokie wartości dodatnie, aby wypchnąć obraz poza prawą lub dolną krawędź elementu. Obszar obrazu znajdujący się poza obszarem elementu nie jest wyświetlany.

- **50%** — obraz zostaje zeskalowany tak, by zajmował 50% wymiaru elementu o wyższej wartości.
- **100px 50px** — obraz otrzymuje szerokość 100 pikseli i wysokość 50 pikseli.
- **cover** — obraz zostaje powiększony tak, by całkowicie wypełnić obszar elementu.
- **contain** — obraz zmienia rozmiar tak, by zmieścić się w elemencie. Opierając się na przykładzie określania położenia tła z poprzedniego rysunku, w którym pojedynczy obraz wyśrodkowałem w elemencie, dodałem wymienione powyżej wartości (**rysunek 3.38**).

RYSUNEK 3.38. Różne wartości `background-size` nadane niepowtarzającemu się, wyśrodkowanemu obrazowi tła



Ta nowa właściwość zapewnia wiele dodatkowych możliwości aranżowania obrazów tła. Uważaj tylko, by nie powiększać małych obrazów, gdyż wpływa to niekorzystnie na ich jakość.

Zaczepienie tła

Właściwość `background-attachment` określa, czy obraz tła przewijanego elementu porusza się wraz z przewijaniem owego elementu. Domyślna wartość to `scroll`, przy której obraz tła porusza się wraz z elementem. Wartość `fixed` sprawia, że obraz tła nie porusza się podczas przewijania elementu.

Właściwość `background-attachment:fixed` często wykorzystuje się, by dodać wytłumiony znak wodny, wyśrodkowany względem elementu `body`, tak by zawartość strony przewijała się nad nieruchomym obrazem.

Reguła CSS, która pozwala na osiągnięcie takiego efektu, mogłaby wyglądać następująco:

```
body {  
    background-image:url(images/watermark.png);  
    background-position:center;  
    background-color:#fff;  
    background-repeat:no-repeat;  
    background-size:contain;  
    background-attachment:fixed;  
}
```

Jak widzisz, reguły definiujące obrazy tła mogą być dość długie, ale wszystkie z nich możesz podać w postaci jednej reguły, używając właściwości `background`.

Właściwość zbiorcza tła

Właściwość `background` to właściwość zbiorcza, która pozwala na zamieszczenie wszystkich właściwości tła w jednej regule. Przykład kodu z powyższego punktu można by zapisać następująco:

```
body {background:url(images/watermark.png) center #fff no-repeat contain fixed;}
```

Gdybym pominął wartość jakiejś właściwości (powiedzmy dla przykładu, że `no-repeat`), to wykorzystana byłaby jej domyślna wartość (w tym przypadku `repeat`).

Inne właściwości tła w CSS3



Modernizr to biblioteka JavaScript, która wykrywa obsługę funkcji HTML5 i CSS3 w przeglądarce użytkownika. Więcej przeczytasz na stronie <http://modernizr.com>.

W CSS3 pojawiło się kilka nowych właściwości `background`, które pokrótkce omówię. Nie są one powszechnie obsługiwane, więc jeśli zdecydujesz się z nich skorzystać, to sprawdź też, jak Twoja strona wygląda bez nich, lub skorzystaj z narzędzia Modernizr do sprawdzenia ich obsługi i uzyskania zastępczego kodu CSS dla przeglądarek, które ich nie obsługują.



Więcej o tych nowych właściwościach dowiesz się na stronie <http://www.w3.org/TR/2001/WD-css3-background-20010924>.

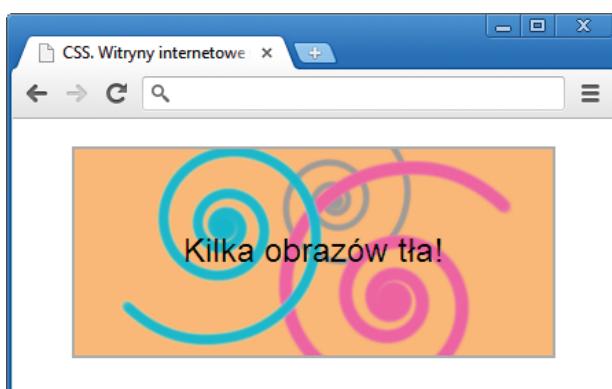
- **background-clip** określa, gdzie obraz jest wyświetlany, np. tylko pod obszarem treści, a nie dopełnienia. Tło domyślnie rozciąga się na cały obszar elementu, w tym marginesu.
- **background-origin** wskazuje punkt początkowy w innych miejscach niż w lewym górnym rogu pola elementu, np. w lewym górnym rogu pola treści.
- **background-break** pozwala na określenie sposobu wyświetlania dzielonych elementów, takich jak pola liniowe, które dzielą się na wiele wierszy.

Większa liczba obrazów tła

CSS3 umożliwia zamieszczanie w tle elementu więcej niż jednego obrazu. Zrobię to przy użyciu właściwości zbiorczej **background**, czego efekt widać na **rysunku 3.39**.

```
p {height:150px; width:348px; border:2px solid #aaa;
margin:20px auto; font:24px/150px helvetica, arial, sans-serif;
text-align:center; background:
url(images/turq_spiral.png) 30px -10px no-repeat,
url(images/pink_spiral.png) 145px 0px no-repeat,
url(images/gray_spiral.png) 140px -30px no-repeat, #ffbd75;}
```

RYSUNEK 3.39. W tle można nałożyć na siebie wiele obrazów. Pierwszy obraz podany w kodzie CSS znajduje się najwyżej



W kodzie CSS zamieściłem każdy obraz w osobnym wierszu, aby uwidoczyć, że wszystkie obrazy tła wraz z ustawieniami położenia i powtórzenia oddzielone są od siebie przecinkami. Na końcu dodałem także wartość **background-color** (która wyróżniłem w kodzie), żeby element nie miał domyślnego, przezroczystego tła, w przypadku gdyby obrazy się nie załadowały. Zauważ, że obraz podany na początku występuje na szczycie stosu, czyli najbliżej pierwszego planu.

Prefiksy

Aby zachęcić producentów przeglądarek do szybkiego wdrożenia specyfikacji CSS3 W3C, wprowadzono do użycia prefiksów.

Te prefiks'y, dodawane przed nazwami właściwości CSS, dają producentom pole do eksperymentowania z projektowanymi przez W3C właściwościami CSS. Producenci mogą z łatwością dodawać obsługę nowych właściwości do przeglądarek, jednocześnie wskazując, że mamy do czynienia z przejściowymi, niepełnymi bądź eksperymentalnymi implementacjami specyfikacji, z których można korzystać wyłącznie na własną odpowiedzialność.

Oto przykład zaleconej przez W3C składni właściwości `transform`:

```
transform: skewX(-45deg);
```

Właściwość CSS3 `transform` wciąż jest jednak w produkcji, więc aby mieć pewność, że będzie działać w możliwie jak największej liczbie przeglądarek, w których jej implementacja jest w fazie eksperymentalnej, powinieneś także dodać wszelkie prefiks'y przeglądarek, które mają obsługiwać Twój kod. Przeglądarka zwyczajnie używa tego kodu, który rozumie.

- `-moz-transform:skewX(-45deg); /* Firefox */`
- `-webkit-transform:skewX(-45deg); /* Chrome i Safari */`
- `-ms-transform:skewX(-45deg); /* Microsoft Internet Explorer */`
- `-o-transform:skewX(-45deg); /* Opera */`
- `transform:skewX(-45deg); /* standardową deklarację W3C umieść na końcu */`

Jak widzisz, prefiks'y zawsze zaczynają się od myślnika, po którym znajduje się nazwa prefiks'u i kolejny myślnik. Dopiero po nich umieszcza się właściwą nazwę właściwości W3C. Zauważ też, że tak jak w powyższym przykładzie, właściwa deklaracja właściwości W3C powinna znaleźć się po wszelkich deklaracjach z prefiksami, tak aby mogła definiować właściwość kiedyś w przeszłości, kiedy będzie ona obsługiwana w ostatecznej wersji bez prefiksów. Jak widzisz, prefiks `-webkit-` używany jest zarówno przez Safari, jak i przez Chrome, ponieważ obydwie te przeglądarki używają silnika Webkit.

Z prefiksami trzeba zapisywać następujące właściwości CSS3:

- `border-image`,
- `linear-gradient`,
- `radial-gradient`,
- `transform`,
- `transform-origin`,
- `translate`,
- `transition`,
- `background*`,
- `background-image*`.

* Kiedy używasz więcej niż jednego obrazu tła lub gradientów.

W celu zaoszczędzenia miejsca nie zawsze będę podawać pełny zestaw prefiksów w każdym z przykładów, w którym należałoby z nich skorzystać, tylko będę wskazywać, że są one wymagane. Wraz z rozwojem przeglądarki zapotrzebowanie na prefiks'y będzie się zmieniać. Najnowsze informacje o CSS3 i prefiksach znajdziesz na stronie <http://caniuse.com>. Do automatycznego dodawania prefiksów skorzystaj ze skryptu `-prefix-free`, o którym przeczytasz w „Dodatku”.

Gradienty tła



Gradienty to obrazy tła, które można wygenerować przy użyciu CSS. Można je dodawać przy użyciu właściwości `background-image` lub, tak jak to zrobię w kolejnych przykładach, używając właściwości zbiorczej `background`.

Gradient jest kolorowym wypełnieniem, będącym przejściem między dwoma lub więcej kolorami. Zanim pojawił się CSS3, gradienty trzeba było tworzyć w edytorsach graficznych takich jak Adobe Photoshop i zamieszczać na stronach jako obrazy tła. Obecnie jednak można je tworzyć przy użyciu CSS.

Istnieją dwa rodzaje gradientów: liniowe i promieniste. Gradienty liniowe rozciągają się od jednego końca elementu do drugiego, a promieniste od jednego punktu w obrębie elementu do jego krawędzi.

Zacznijmy od kilku prostych gradientów liniowych.

Oto kod HTML:

```
<div class="gradient1"></div>
<div class="gradient2"></div>
<div class="gradient3"></div>
```

i CSS:

stylizuje pole elementu

```
div {height:150px; width:200px; border:1px solid #ccc;
float:left; margin:16px;}
```

domyślnie przebiega z góry na dół

```
.gradient1 {background:linear-gradient(#e86a43, #fff);}
```

od lewej do prawej

```
.gradient2 {background:linear-gradient(left, #64d1dd, #fff);}
```

od lewego górnego rogu

```
.gradient3 {background:linear-gradient(-45deg, #e86a43, #fff);}
```

do prawego dolnego rogu

RYSUNEK 3.40 Trzy proste gradienty liniowe



Na rysunku 3.40 widać trzy proste przykłady gradientów liniowych. W przykładzie 1. podałem kolor początkowy i końcowy, gdzie gradient gładko przechodzi od jednego do drugiego w domyślnym kierunku, czyli z góry na dół. W przykładzie 2. podałem słowo kluczowe wartości początkowej `left`, wobec czego gradient zaczyna się po lewej i kończy po prawej. Wartość początkowa w przykładzie 3. to `-45deg` (stopni), co przenosi punkt początkowy z domyślnego środka górnej krawędzi do lewego górnego rogu.

Punkty pośrednie

Punkty pośrednie są punktami, w których definiowane są kolory i wartości krycia. Wygląd gradientu przejścia określony jest zmianą konieczną do uzyskania podanej wartości koloru w kolejnym punkcie pośrednim. Możesz określić tyle punktów pośrednich, ile chcesz. Położenie punktu pośredniego wyrażone jest zwykle procentem długości gradientu. Na **rysunku 3.41** widnieją cztery gradienty z punktami pośrednimi.

pojedynczy punkt pośredni w 50% długości gradientu

```
.gradient1 {background: linear-gradient(#64d1dd, #fff 50%, #64d1dd);}
```

przejście zaczyna się w punkcie 20%, a kończy w punkcie 80%

```
.gradient2 {background: linear-gradient(#e86a43 20%, #fff 50%, #e86a43 80%);}
```

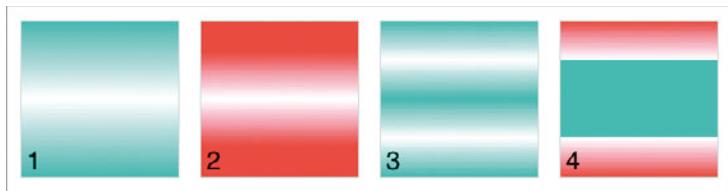
trzy punkty pośrednie: 25%, 50% i 75%

```
.gradient3 {background: linear-gradient(#64d1dd, #fff 25%, #64d1dd 50%, #fff 75%, #64d1dd);}
```

podwójne punkty pośrednie tworzą mocne przejścia

```
.gradient4 {background: linear-gradient(#e86a43, #fff 25%, #64d1dd 25%, #64d1dd 75%, #fff 75%, #e86a43);}
```

RYSUNEK 3.41. Gradienty liniowe z punktami przejściowymi



Kiedy położenie punktów pośrednich nie jest określone wartościami procentowymi lub innymi, kolory rozmieszczone są równomiernie na linii gradientu — w tym wypadku są to punkty na 0%, 50% i 100% jego długości.

Na **rysunku 3.41 przykład 1.** przedstawia pojedynczy punkt pośredni w 50% długości gradientu, wobec czego kolor przechodzi gładko od początku do koloru punktu pośredniego (czyli białego), a następnie gładko przechodzi od niego do koloru końcowego. Zauważ, że położenie punktu początkowego i końcowego określone jest domyślnie jako 0% i 100%, jeżeli nie podano innych wartości.

Przykład 2. ukazuje, co się dzieje, gdy punkt początkowy i końcowy umieszczone są w innych miejscach niż w 0% i 100% długości gradientu. Aż do pierwszego punktu pośredniego (20%) gradient składa się z jednolitego koloru określonego dla tego punktu, a następnie przechodzi w barwę kolejnego punktu pośredniego. Ten sam efekt widać przy ostatnim punkcie pośrednim w 80% długości: jego jednolity kolor ciągnie się do samego końca elementu.

W **przykładzie 3.** widać naprzemienne przejścia między dwoma kolorami określonymi wieloma punktami pośrednimi, podczas gdy **przykład 4.** ukazuje, że podając dwa kolory w tym samym punkcie, można uzyskać mocne przejście.

GRADIENTY PROMIENISTE

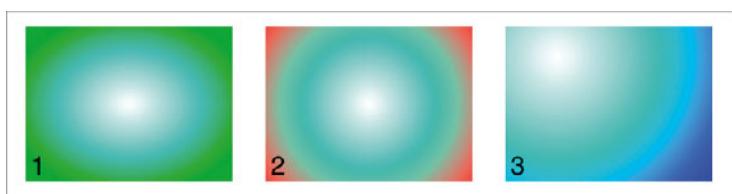
Gradienty promieniste są nieco bardziej skomplikowane niż liniowe, ponieważ istnieje więcej możliwości ich kontrolowania. Jeżeli jesteś programistą, to pewnie poznałeś po nawiasach, że właściwości gradientów są funkcjami. Innymi słowy, właściwość przyjmuje szereg wartości, zwanych argumentami, które wykorzystuje przy generowaniu gradientu. Przy tworzeniu gradientów promienistycznych możesz używać argumentów definiujących kształt, położenie, wielkość, kolor i przezroczystość.

W poniższych przykładach każdemu gradientowi przypisuję trzy kolory.

```
.gradient1 {background: -webkit-radial-gradient(#fff, #64d1dd, #70aa25);}
.gradient2 {background: -webkit-radial-gradient(circle, #fff, #64d1dd, #e86a43);}
.gradient3 {background: -webkit-radial-gradient(50px 30px, circle, #fff, #64d1dd, #4947ba);}
```

 Podaję tu jedynie prefiks `-webkit-`. W ramce „Prefiksy” przeczytasz o pozostałych wymaganych prefiksach.

RYSUNEK 3.42. Trzy trójkolorowe gradienty promieniste. Pierwszy jest gradientem dopasowanym domyślnie do kształtu elementu, drugi kolistym gradientem, a trzeci kolistym, przemieszczonym gradientem.



Na **rysunku 3.42 przykład 1.** przedstawia domyślny kształt gradientu, gdzie przejście dopasowane jest do kształtu elementu — w tym przypadku prostokąta. Gdyby element miał kwadratowy kształt, to gradient byłby kolisty.

W **przykładzie 2.** słowo kluczowe kształtu `circle` sprawia, że przejście rozchodzi się równomiernie i zatrzymuje się na najbliższej krawędzi, nadając mu kształt koła. Reszta miejsca na dłuższej osi wypełniona jest kolorem końcowym. W **przykładzie 3.** argumenty położenia, `50px 30px`, przenoszą punkt startowy gradientu bliżej lewego górnego rogu.

Znasz już podstawy pracy z gradientami. W dalszych rozdziałach przedstawię Ci więcej przykładów, gdy będę używać gradientów w różnych układach graficznych.

Więcej szczegółów związanych z tymi przykładami i z przemieszczaniem gradientów promienistycznych znajdziesz w moim e-booku, *Visual Stylin' with CSS*, opublikowanym przez Peachpit Press w roku 2012.

Podsumowanie

W tym rozdziale dowiedziałeś się o działaniu modelu polowego oraz nauczyłeś się definiować marginesy, dopełnienia i obramowania. Zobaczyłeś, jak tworzenie pływających elementów umożliwia oblewanie tekstu wokół obrazów, a także umieszczanie elementów blokowych obok siebie w poziomie. Wiesz już, jak oczyszczać pływające elementy i zmuszać kontenery do obejmowania swoich pływających dzieci. Zapoznałeś się z właściwościami `position` i `display` i dowiedziałeś się, jaką dają Ci kontrolę nad rozmieszczeniem elementów na stronie. Wreszcie, dowiedziałeś się, jak nadawać warstwie tła elementu kolory, obrazy i gradienty.

Zajmę się teraz ustawieniami CSS odpowiedzialnymi za fonty i tekst i pokażę Ci techniki, które pozwolą Ci uzyskać profesjonalne rezultaty typograficzne.

ROZDZIAŁ 4

Stylizowanie fontów i formatowanie tekstu

Webdesign to w znacznej mierze praca z tekstem zawartym w akapitach, nagłówkach, listach, menu i formularzach, więc zapoznanie się z przedstawionymi w tym rozdziale właściwościami CSS jest kluczowe dla tworzenia stron o profesjonalnym wyglądzie. Tekst w stopniu większym niż niemal wszystkie inne czynniki świadczy o jakości tego, co strona oferuje. Grafika jest wisienką na torcie, ale dobry design opiera się na typografii.

W tym rozdziale dowiesz się czegoś o fontach i tekście oraz właściwościach CSS służących do ich stylizacji. Przedstawię Ci również niezwykły świat fontów internetowych, które użytkownicy pobierają wraz ze stronami internetowymi. Teraz już nie musisz liczyć na to, że użytkownik będzie miał u siebie zainstalowane różnorakie fonty, i możesz mieć pewność, że Twój tekst zawsze będzie wyświetlany tak, jak tego chcesz. Zaczniemy od fontów.

Fonty

Występujące na Twoich stronach internetowych kroje mogą pochodzić z trzech źródeł:

- Fonty zainstalowane w systemie użytkownika. (Do niedawna były to jedyne fonty, których można było używać na stronach).
- Fonty znajdujące się na zewnętrznych serwisach, głównie na Typekit i Google, które możesz podłączyć do swojej strony przy użyciu znacznika [link](#).
- Fonty znajdujące się na Twoim serwerze i przekazywane przeglądarce użytkownika za pośrednictwem strony, z wykorzystaniem reguły `@font-face`.

Przykłady podane w dalszych opisach właściwości fontów będą odnosić się do pierwszego z tych źródeł: fontów znajdujących się na komputerze użytkownika. W podrozdziale „Fonty internetowe” znajdziesz omówienie pozostałych źródeł.

Font a tekst

Każdy font jest zbiorem liter, liczb i symboli o wyjątkowym wyglądzie. Fonty dzielą się na kolekcje, do których zaliczane są na podstawie ich ogólnego wyglądu, np. szeryfowe, bezszeryfowe czy maszynowe. Fonty przynależą do rodzin, takich jak Times czy Helvetica. Do rodzin fontów należą już konkretne fonty, które mają swoje odmiany, np. Times Roman, Times Bold, Helvetica Condensed albo Bodoni Italic.

Tekst odnosi się z kolei do słów i znaków. Tekstem jest ten akapit i tytuł rozdziału, a to, jakim fontem są zapisane, to zupełnie inna sprawa.

CSS obsługuje szereg właściwości odnoszących się do fontów, a także właściwości odnoszące się do tekstu. Właściwości fontów odnoszą się do wielkości i wyglądu kolekcji. Określają rodzinę (np. Times lub Helvetica), wielkość albo to, czy krój jest pogrubiony lub pochylony. Właściwości tekstu odnoszą się do tego, jak krój jest wykorzystany. Definiują interlinię i odstępy międzyliterowe. Określają, czy ma być podkreślony lub wcięty, oraz wiele innych cech.

Myśl o tym tak, że style fonta, takie jak pogrubienie i kursywę, możesz nadawać pojedynczym znakom, ale właściwości tekstu, takie jak interlinia i wcięcie, mają sens jedynie w odniesieniu do jakiegoś bloku tekstu, takiego jak nagłówek lub akapit.

Przyjrzyjmy się teraz sześciu właściwościom odnoszącym się do stylu fontów:

- `font-family`,
- `font-size`,
- `font-style`,
- `font-weight`,
- `font-variant`,
- `font` (zbiorcza).

Właściwość `font-family`

Przykład: `h2 {font-family:times, serif;}`

Właściwość `font-family` wskazuje krój, w jakim element tekstoowy jest wyświetlany. Podstawowy font zazwyczaj nadaje się całej stronie, a style `font-family` jedynie tym elementom, które chcesz wyświetlać innymi fontami. Aby określić font całej strony, należy zdefiniować właściwość `font-family` elementu `body`:

`body {font-family:verdana, sans-serif;}`



Z własnego doświadczenia wiem, że przy podawaniu nazw fontów jako wartości `font-family` nie trzeba zwracać uwagi na wielkość liter. Nie zmienią jednak wielkości liter nazw generowanych przez Google'a i inne tego typu serwisy, gdyż fonty mogą się przez to nie wyświetlać.

Właściwość `font-family` jest dziedziczona, toteż jej wartość przekazywana jest wszystkim potomkom. W przypadku elementu `body` są to wszystkie elementy w kodzie.

Ponieważ fonty muszą się albo znajdować na komputerze użytkownika, albo zostać przekazane za pośrednictwem sieci, zawsze istnieje ryzyko, że podany przez Ciebie font nie będzie dostępny. Z tego względu fonty zawsze podaje się w listach.

PODAWANIE ZAINSTALOWANYCH FONTÓW W LIŚCIE

Z fontów zainstalowanych na systemie operacyjnym danego urządzenia korzystać mogą wszystkie działające na nim aplikacje.

Standardowy system operacyjny dostarczany jest z ograniczonym zakresem fontów. Z kolei użytkownik może je dodawać i usuwać, wobec czego nigdy nie możesz mieć całkowitej pewności, które z nich będą dostępne przy wyświetlaniu Twoich stron. Z tego powodu, kiedy podajesz font, w jakim tekst ma być wyświetlany, musisz podawać także fonty zapasowe na wypadek, gdyby system użytkownika nie dysponował fontem preferowanym. Preferowane i zastępcze fonty podaje się w formie listy.

Listy fontów mają na celu zapewnić, że tekst strony zostanie wyświetlony w wybranym foncie, o ile jest on zainstalowany na systemie użytkownika, a jeśli nie, to żeby wykorzystany został w tym celu dopuszczalny zamiennik.

```
body {font-family: "trebuchet ms", tahoma, sans-serif;}
```

Lista fontów tym samym mówi przeglądarce: „Wyświetl ten dokument w foncie Trebuchet MS, a jeśli nie ma go w systemie, to w Tahoma; a jeśli żadnego z tych fontów nie ma, to użyj dowolnego fonta bezszeryfowego, jaki znajdziesz”. Bardzo ważne jest to, by ostatnia pozycja deklaracji `font-family` określała font ogólnie, co zwykle się robi, podając wartości `serif` lub `sans-serif` jako ostatnią deskę ratunku.

Istnieje pięć ogólnych wartości `font-family`:

- **Serif** — litery krojów szeryfowych mają drobne szczegóły na końcówkach znaków (co możesz zobaczyć w tekście tej książki).
- **Sans-serif** — litery krojów bezszeryfowych nie mają drobnych szczegółów na końcówkach znaków (co możesz zobaczyć na przykładzie nagłówków w tej książce).
- **Monospace** — litery krojów maszynowych zajmują jednakową ilość miejsca w poziomie (co możesz zobaczyć w przykładach kodów w tej książce).



Ponieważ nazwa fonta Trebuchet MS składa się z ponad jednego wyrazu, trzeba ją zwrzeć w cudzysłowie.

- **Cursive** — litery zapisane kursywą wyglądają, jakby je pisano odręcznie (tak jak w nagłówku przykładowego fragmentu *Psa Baskerville’ów* w dalszej części rozdziału).
- **Fantasy** — są to kroje, które nie pasują do innych kategorii (zwykle są dość dziwaczne).

Takie ogólne deklaracje krojów mają zapewnić, że jeśli żaden ze wskazanych fontów nie będzie dostępny, to krój służący do wyświetlenia tekstu dokumentu będzie przynajmniej właściwego rodzaju.

Wybór fontów do listy warto potraktować skrupulatnie.

Dreamweaver na przykład wyświetla menu różnych list fontów za każdym razem, kiedy wpisujesz **font-family** w pliku CSS, ale powadane przez niego fonty nie są idealnymi zamiennikami. Oto jedna z list Dreamweavera:

`verdana, arial, helvetica, sans-serif;`


Wysokość małej litery x (ang. x-height) obejmuje główną część liter, z wyłączeniem górnych i dolnych wydłużen obecnych w literach takich jak d i p. Litera x takich wydłużen nie ma, więc stąd nazwa.

Verdana jest dość tegim krojem o dużo większej wysokości x niż Arial. Jeśli więc na komputerze użytkownika nie ma Verdany, strona zostanie wyświetlona w Arialu, czyli foncie mniejszym od tego, który chciałeś. W każdym wierszu będzie się mieścić więcej słów, a pionowa wysokość bloków tekstowych może być mniejsza.

Dobrym sposobem na przetestowanie strony jest obejrzenie jej z kolejnymi fontami z listy jako pierwszymi w kolejności, aby przekonać się, jak layout się zmienia wraz z przechodzeniem na kroje zapasowe.

Lepszym zamiennikiem Verdany jest Tahoma, krój o takiej samej, dużej wysokości **x**.

`helvetica, tahoma, sans-serif`


Więcej na temat wybierania fontów przeczytasz na stronie <http://unitinteractive.com/blog/2008/06/26/better-css-font-stacks>.

W liście lżejszych krojów bezszeryfowych mógłś z kolei podać **`helvetica, arial, sans-serif`**

Oto lista fontów szeryfowych, zaczynająca się od kroju, którym użytkownik może nie dysponować:

`{font-family: "hoefer text", georgia, times, serif;}`

W takich przypadkach zawsze warto kończyć listę fontami, które są dostarczane z większością systemów operacyjnych. W tym wypadku jest to Georgia, Times, a na końcu widnieje ogólny font szeryfowy.

Jakich fontów używają wszystkie przeglądarki?

Jest to częste pytanie, na które nie ma jednoznacznej odpowiedzi. Istnieje jednak duże prawdopodobieństwo, że na każdym komputerze, maku czy pececie, znajdują się następujące fonty:

Szeryfowe

Georgia
Palatino/Book Antiqua
Times New Roman

Bezszerfowe

Arial
Arial Black
Arial Narrow
Tahoma
Trebuchet MS
Verdana

Maszynowe

Courier New
Lucida Console/Monaco

Kursywowe

Comic Sans MS

Ozdobne

Impact

Ze względu na to, że w nowoczesnych telefonach i tabletach można się spotkać z dość niecodziennymi fontami, szczególnie ważne staje się umieszczanie ogólnych fontów zapasowych w listach. Jeżeli chcesz skorzystać z konkretnego kroju, użyj fonta internetowego dostępnego na zewnętrznym serwisie lub takiego, który można pobrać z Twojego serwera. Więcej na ten temat przeczytasz w podrozdziale „Fonty internetowe”.

Właściwość font-size

Przykład: `h2 {font-size:18px;}`

Każdy element tekstowy HTML ma ustawioną domyślną wartość `font-size` w arkuszu przeglądarki. Kiedy więc określasz `font-size` elementu, przesłaniasz wartość domyślną. Wielkość tekstu może ulegać nieprzewidywalnym zmianom, jeśli nie orientujesz się, jak na dziedziczenie wielkości tekstu wpływa wybór jednostek wielkości. Właściwości `font-size` możesz nadawać wartości w dwóch różnych jednostkach: bezwzględnych, takich jak piksele i punkty, oraz względnych, takich jak procenty i jednostki em. Wy tłumaczę teraz, na czym polega różnica.

Właściwość `font-size` jest dziedziczona, więc zmiana wielkości tekstu elementu pociąga za sobą proporcjonalną zmianę wielkości tekstu jego potomków. Innymi słowy, jeśli właściwości `font-size` elementu `body` nadasz wartość `200%`, to tekst wszystkich elementów na stronie będzie dwukrotnie większy.

Efekt ten występuje, ponieważ wielkość tekstu wszystkich elementów jest określona w arkuszu stylów przeglądarki względnią jednostką em. Tekst elementu `h1` ma na przykład wielkość `2em`, `h2` wielkość

1.5em, a elementu **p** (akapit) wielkość **1em**. **1em** jest domyślnie równy 16 pikselom. Domyślnie zatem tekst **h1** ma wielkość 32 pikseli ($16 \times 2\text{em} = 32$ piksele), **h2** wielkość 24 pikseli, a **p** wielkość 16 pikseli.

Nadając tekstuowi **body** wielkość **20px**, zmieniasz podstawową wielkość tekstu, wobec czego wielkość tekstu elementu **h1** zmienia się na **40px** (20 pikseli \times 2em = 40 pikseli), **h2** na 30 pikseli, a **p** na 20 pikseli. Tym niemniej dziedziczenie właściwości **font-size** nie zachodzi w przypadku potomków, których wielkość tekstu zdefiniowano jednostkami bezwzględnymi, takimi jak piksele. Takie elementy zawsze są wyświetlane we wskazanej wielkości.

Dowiedzmy się teraz więcej o definiowaniu wielkości tekstu, przyglądając się poszczególnym metodom.

BEZWZGLĘDNE USTAWIANIE WIELKOŚCI TEKSTU



Wielkość tekstu można również zmieniać słowami kluczowymi, takimi jak **x-small**, **medium** czy **x-large**. Wartość **medium** jest równa wielkości podstawowej, a pozostałe słowa kluczowe tworzą większy i mniejszy tekst. Ponieważ słowa kluczowe pozwalają na uzyskanie ograniczonego zakresu wielkości, nie są one szeroko stosowane. Więcej na ich temat przeczytasz na stronie http://cssdiscuss.incutio.com/wiki/Using_Keywords.

Ustawianie wielkości tekstu jednostkami bezwzględnymi w rodzaju pikseli, pica czy cali jest łatwe. Kiedy określasz wielkość tekstu elementu jednostkami bezwzględnymi, pozostaje ona taka sama niezależnie od wielkości tekstu przodków elementu. Minusem ustawiania wielkości tekstu jednostkami bezwzględnymi jest to, że jeśli zechcesz proporcjonalnie zmienić wielkość ogólną tekstu na stronie, będziesz musiał ręcznie podmienić każdą bezwzględną wartość **font-size** w arkuszu. Dopracowanie strony z wielkościami definiowanymi bezwzględnie wymaga większego wysiłku.

Mówiąc krótko, jeśli zmienisz wielkość tekstu znacznika **body**, elementy, których wielkości tekstu zdefiniowano bezwzględnie, pozostaną niezmienione, ale elementy, których rozmiar nie zdefiniowano w arkuszu CSS, zmienią wielkość proporcjonalnie do wielkości tekstu **body**.

WZGLĘDNE USTAWIANIE WIELKOŚCI TEKSTU

Określanie wielkości tekstu jednostkami względnymi, takimi jak procenty, em bądź rem, jest nieco bardziej skomplikowane. Kiedy nadajesz elementowi wielkość przy użyciu takich jednostek, wielkość tekstu określana jest względem najbliższego przodka o zdefiniowanej wielkości tekstu.

Spójrzmy na fragment prostego kodu HTML

```
<body>
  <p>To jest <strong>bardzo ważne!</strong></p>
</body>
```



Jeśli chcesz korzystać z jednostek em, ale musisz też zdefiniować określone wielkości w pikselach, dobrym rozwiązaniem jest nadanie właściwości `font-size` znacznika `body` wartości 62,5%. W ten sposób zmieniasz wielkość podstawową tekstu z 16 na 10 pikseli ($16 \times 62,5\% = 10$), dzięki czemu przekładanie jednostek em na piksele staje się łatwiejsze: 1em to 10px, 1.5em to 15px, 2em to 20px itd.



Kiedy korzystasz z jednostek względnych, definiowanie wielkości tekstu zaczynaj od samej góry hierarchii, przechodząc coraz niżej.



Przeglądarka IE9 i wcześniejsze wersje skalują wyłącznie tekst zdefiniowany jednostkami względnymi (a nie bezwzględnymi, takimi jak piksele), kiedy użytkownik zmienia wielkość tekstu przy użyciu funkcji w menu przeglądarki Widok/Rozmiar tekstu. Korzystanie z jednostek rem wiąże się zatem z drobną wadą: jeżeli użytkownik IE7 lub IE6 zechce powiększyć tekst, będzie musiał skorzystać z opcji Widok/Powiększenie, by zwiększyć całość tekstu strony. To kolejny powód, dla którego warto zmienić przeglądarkę na nowszą.

i CSS

```
p {font-size:.75em;}  
strong {font-size:.75em;}
```

W tym przykładzie tekst znacznika `p` miałby wielkość 12 pikseli (czyli wielkość podstawowa `body`, 16 pikseli \times 0,75 = 12). Ponieważ `strong` jest dzieckiem `p`, wielkość jego tekstu wynosi 9 punktów. Jak widać, względne wielkości kumulują się wraz z przechodzeniem na coraz niższe poziomy hierarchii. Wielkość tekstu znacznika `strong` to 16 pikseli \times 0,75 \times 0,75 = 9 pikseli. Opanowanie jednostek względnych wymaga praktyki, skoro — w odróżnieniu od jednostek bezwzględnych — zmiana względnej wielkości tekstu elementu po-ciąga za sobą proporcjonalną zmianę wielkości tekstu jego dzieci.

Korzystając z jednostek względnych, masz możliwość dostosowania wielkości wszystkich elementów proporcjonalnie poprzez zmianę wielkości `body` albo wybranej grupy elementów poprzez zmianę wielkości tekstu jakiegoś wspólnego przodka. Pozwala to na zaoszczędzenie czasu przy eksperymentowaniu z layoutem, ale z tego samego względu wymaga planowania. Zmiana właściwości `font-size` elementu wpływa także na wszystkich jego potomków.

Wielkości tekstu nie możesz dostosować w ten sposób, kiedy prakcyjesz z bezwzględnymi jednostkami `font-size` — każdy element, którego wielkość określona jest bezwzględnie, musi być dostosowywany indywidualnie. Rzecz jasna, kiedy korzystasz z jednostek względnych, możesz zmienić wielkość elementu, nie przejmując się niepożdanym zwykle efektem zmiany wielkości wszystkich jego potomków.

Mając jednak wzrok na to, z jakim szerokim zakresem wymiarów ekranów trzeba się dziś liczyć — od wielkich monitorów, po wyświetlacze małych telefonów — potrzeba łatwości skalowania tekstu sprawia, że definiowanie jego wielkości jednostkami względnymi jest lepszym rozwiązaniem.

JEDNOSTKI REM

Teraz w CSS3 jednostka względna rem (ang. *root em*) wzbudza nieniąłą ekscytację w społeczności programistów internetowych. Kiedy określasz wielkość elementu jednostkami rem, jego wielkość jest względna, ale jedynie w stosunku do elementu głównego HTML. To zachowanie łączy w sobie największe zalety skalowania bezwzględnego i względnego: ogólną wielkość tekstu możesz zmieniać proporcjonalnie poprzez zmianę wielkości tekstu danego elementu HTML jednostką względną, z tym że zmiany na kolejnych poziomach hierarchii nie kumulują się tak jak w przypadku jednostek em. Jednostki rem obsługiwane są przez wszystkie aktualne przeglądarki.

ki, ale nie przez IE8 i wcześniejsze wersje. Istnieje jednak prosta metoda na ominięcie tej niezgodności: wystarczy podać wartość bezwzględną w pikselach tym przeglądarkom, które nie rozumieją jednostek rem:

IE8 i wcześniejsze wersje
używają wartości 14px

`p {font-size:14px; font-size:.875rem;}`

Przyjrzyjmy się teraz innym właściwościom CSS związanym z fontami.

Właściwość font-style

Wartości: `italic`, `oblique`, `normal`

Przykład: `h2 {font-style:italic;}`

Właściwość `font-style` określa, czy tekst jest pochylony. Zamiast wartości `italic` możesz podać `oblique` — efekt jest jednakowy.

Właściwość `font-style` ma tylko dwie użyteczne wartości: `italic` sprawia, że standardowy tekst zapisywany jest kursywą, a `normal` służy do tego, by zmienić jakiś fragment pochyłego tekstu z powrotem w „pionowy”. Kod poniższego przykładu

```
p {font-style:italic;}  
span {font-style:normal;}  
<p>Oto pochyły tekst <span>z fragmentem zwykłego</span>  
pośrodku.</p>
```

daje rezultat widoczny na **rysunku 4.1**.

RYSUNEK 4.1. Wartość normal właściwości font-style sprawia, że wskazany fragment pochylonego tekstu wyświetlany jest jak zwyczajny tekst

Oto pochyły tekst z fragmentem zwykłego pośrodku.

Zauważ, że tekst pochyły służy głównie do wyróżniania tekstu, np. „Dziś jest naprawdę gorąco!”. Jeśli chcesz coś wyróżnić, użyj znacznika `em`, który domyślnie pochyla tekst.

Wartość normal

Wartość `normal` zapobiega wyświetleniu na ekranie wszelkich efektów, jakie właściwość standardowo wywołuje. Ale do czego ona właściwie służy?

Opcja ta jest dostępna po to, abyś mógł wybiórczo dezaktywować domyślne lub ustawione przez Ciebie właściwości globalne. Nagłówki od `h1` do `h6` są domyślnie pogrubione, więc jeśli chcesz usunąć pogrubienie z elementu `h3`, to możesz podać kod `h3 {font-weight:normal;}`. Jeżeli w arkuszu stylu podany jest kod `a {font-variant:small-caps;}`, by wszystkie odnośniki wyświetlane były kapitalikami, a chcesz, by pewien zbiór odnośników zapisywany był normalnie wielkimi i małymi literami, to możesz podać deklarację `a.specjalneodnosniki {font-variant:normal;}`.

Właściwość font-weight

Przykładowe wartości: `100`, `200` i tak dalej do `900` albo `lighter`, `normal`, `bold` i `bolder`.

Przykład: `a {font-weight:bold;}`

Pomimo że można podać tyle różnych wartości liczbowych, przeglądarki wyświetlają jedynie dwa rezultaty wszelkich wartości `font-weight`: tekst pogrubiony i zwykły. Ponieważ interpretacja wartości liczbowych różni się w zależności od przeglądarki, moment przejęcia między tekstem zwykłym a pogrubionym też bywa różny, choć zwykle następuje gdzieś około wartości `400`. Najlepiej unikać korzystania z wartości innych niż `bold` i `normal`, co robię na rysunku 4.2.

```
p.shows_weight {font-weight:bold;}  
p.shows_weight span {font-weight:normal;}  
<p class="shows_weight"> Oto pogrubiony tekst <span>z  
fragmentem zwykłego</span> pośrodku.</p>
```

RYSUNEK 4.2. Wartość `normal` właściwości `font-weight` sprawia, że wskazany fragment pogrubionego tekstu wyświetlany jest normalnie

Oto pogrubiony tekst z fragmentem zwykłego pośrodku.

Zauważ, że pogrubiony tekst ma przede wszystkim służyć wskazywaniu, że coś jest ważne, np. „Ostrożnie!”. Ważny tekst oznaczaj znacznikiem `strong`, który domyślnie pogrubia tekst.

Właściwość font-variant

Wartości: `small-caps`, `normal`

Przykład: `blockquote {font-variant:small-caps;}`

Ta właściwość obsługuje tylko jedną wartość (poza `normal`), czyli `small-caps`. Sprawia to, że wszystkie małe litery zapisywane są jako kapitaliki.

```
h3 {font-variant:small-caps;}
```

Rezultat zastosowania powyższego kodu widać na rysunku 4.3.

RYSUNEK 4.3. Oto nagłówek z kapitalikami. Zauważ, że pierwsza litera tekstu jest zapisana jako wielka litera w kodzie i pozostaje niezmieniona na ekranie

Oto tekst po zastosowaniu wartości `small-caps` właściwości `font-variant`.

Wartości `small-caps` często używam z pseudoelementem `::first-line`, tak jak pokażę to w przykładzie z *Psem Baskerville’ów* pod koniec rozdziału. Korzystam z niego sporadycznie, ponieważ tekst zapisany kapitalikami czyta się trudniej ze względu na brak wskazówek wizualnych, których zwykle dostarczają górne i dolne wydłużenia małych liter.

Właściwość font

Przykład: `p {font: bold italic small-caps .9em helvetica, arial, sans-serif;}`

`<p>Oto tekst oznaczony każdą możliwą właściwością.</p>`

Powyższy kod tworzy efekt widoczny na rysunku 4.4.

RYSUNEK 4.4. Pogrubienie, pochylenie, kapitaliki, zdefiniowana wysokość oraz właściwość `font-family` w jednej regule CSS



Pozwól sobie nieco uprzedzić fakty: we właściwości `font-size` możesz również zawrzeć właściwość `line-height`, która odnosi się raczej do tekstu niż do samego fonta, podając wartość w rodzaju `12px/1.5`. Więcej na temat właściwości `line-height` dowiesz się w kolejnym podrozdziale, „Właściwości tekstu”.

O TO TEKST OZNACZONY KAŻDĄ MOŻLIWĄ WŁAŚCIWOŚCIĄ.

Właściwość `font` jest właściwością zbiorczą, która umożliwia zdefiniowanie wszystkich ustawień fonta w pojedynczej deklaracji, zmniejszając tym samym ilość kodu CSS, jaki musisz napisać. Musisz jednak przestrzegać dwóch zasad, aby przeglądarka mogła zinterpretować właściwości poprawnie.

Zasada 1.: Wartości `font-size` i `font-family` zawsze muszą być podane.

Zasada 2.: Kolejność wartości jest następująca:

1. `font-weight`, `font-style`, `font-variant` (w dowolnej kolejności)
2. `font-size`
3. `font-family`

Właściwości tekstu

Skoro już omówiliśmy właściwości fontów, czas przyjrzeć się właściwościom tekstowym. Kiedy chcesz nadać akapitowi wcięcie, umieścić znak w indeksie górnym, np. 6 w 10^6 , zwiększyć odstępy międzyliterowe w nagłówkach lub wykonać wiele innych czynności związanych z formatowaniem tekstu, musisz skorzystać z właściwości tekstowych CSS.

Oto najbardziej przydatne właściwości tekstu CSS:

- `text-indent`,
- `letter-spacing`,
- `word-spacing`,
- `text-decoration`,
- `text-align`,
- `line-height`,
- `text-transform`,
- `vertical-align`.

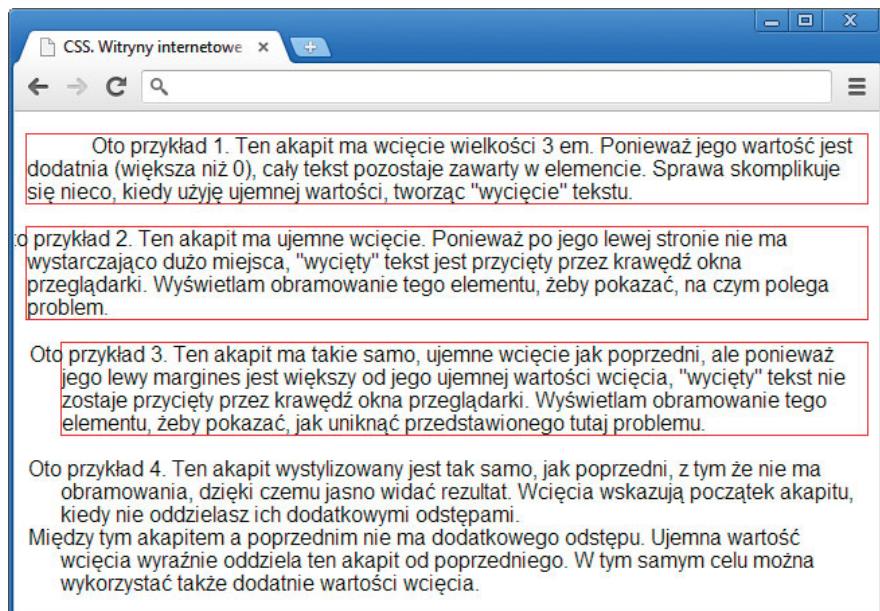
Właściwość `text-indent`

Wartości: dowolna wielkość (dodatnia lub ujemna)

Przykład: `p {text-indent:3em;}`

Ta właściwość określa punkt początkowy pola tekstowego względem kontenera. Domyślnie jest to lewy górny róg kontenera.

Jeżeli właściwości `text-indent` nadasz dodatnią wartość, to tekst przesunie się w prawo, tworząc wcięty akapit (**rysunek 4.5, przykład 1.**).



RYSUNEK 4.5. Cztery przykłady zastosowania właściwości `text-indent`

Wartości dziedziczone i obliczane

Musisz zapamiętać pewną ważną rzecz: właściwość `text-indent` jest dziedziczona przez dzieci. Jeśli na przykład nadasz elementowi `div` właściwość `text-indent`, to wszystkie zawarte w nim akapity odziedziczą jego wartość. Podobnie jednak **jak z wszystkimi dziedziczonymi wartościami CSS**, przekazywana jest nie zdefiniowana wartość, lecz wartość obliczona. Omówię teraz, co z tego zjawiska wynika.

Powiedzmy, że masz `div` z tekstem szerokim na 400 pikseli i 5-procentowym wcięciem. W tym przypadku wcięcie tekstu wynosi 20 pikseli (czyli 5% z 400). W znaczniku `div` znajduje się akapit szeroki na 200 pikseli. Jako dziecko, akapit ten dziedziczy wartość `text-indent`, więc sam ma wcięcie, ale jego odziedziczona wartość jest wynikiem obliczenia przeprowadzonego na wartości rodzica, czyli 20 pikselach, a nie 5%. Wskutek tego akapit również ma 20-pikselowe wcięcie, pomimo że sam jest połowy szerokości swojego rodzica. To zachowanie sprawia, że wszystkie akapity mają jednakowe wcięcia, niezależnie od swojej szerokości. Rzeczą jasna, możesz zniwelować to zachowanie, nadając dzieciom konkretne wartości `text-indent`.

Jeśli jednak nadasz jej wartość ujemną, pierwszy wiersz wyjdzie poza lewą krawędź kontenera; w takiej sytuacji musisz dopilnować, by wystarczyło miejsca na tekst. Jeżeli po lewej stronie znajduje się inny element, to tekst może zacząć na niego nachodzić; jeśli z kolei znajduje się tam krawędź okna przeglądarki, to może zostać przyjęty (**rysunek 4.5, przykład 2.**). Temu problemowi można zapobiec, podając dodatnią wartość lewego marginesu wyższą od ujemnej wartości wcięcia. W **przykładzie 2.** ujemne wcięcie ma wartość `-1.5em`, ale w **przykładzie 3.** znajduje się również lewy margines o wielkości `2em`. Zapisuje się to następująco:

```
p {text-indent:-1.5em; margin-left:2em; border:1px solid red;}
```

Wcięcia nadają tekstowi profesjonalny wygląd, a także przejrzystie wskazują czytelnikowi punkty, w których bloki tekstu się zaczynają. Pamiętaj, by wcięcia i powiązane z nimi marginesy określać jednostkami em, tak jak ja to zrobiłem, aby wcięcie zachowywało proporcję względem szerokości wiersza, kiedy użytkownik zmienia wielkość tekstu (lub sam to robisz).

Właściwość letter-spacing

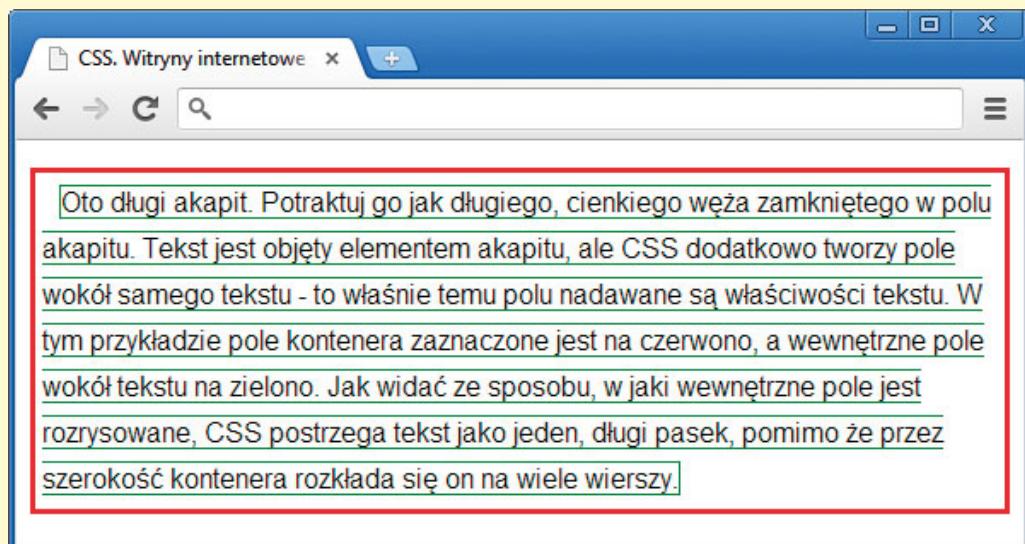
Wartości: dowolna wielkość (dodatnia lub ujemna)

Przykład: `p {letter-spacing: .2em;}`

Dodatnie wartości `letter-spacing` zwiększą odstępy między literami, a ujemne zmniejszą je. Do określania odstępów należy zawsze używać wartości względnych, nawet jeśli wielkość tekstu określona jest pikselami, aby proporcje odstępów były zachowywane przy zmianie wielkości tekstu. Przykłady widać na **rysunku 4.7**.

Jak działa węzyk?

Istotną kwestią z zakresu tego, jak CSS obsługuje tekst, jest to, że CSS tworzy niewidzialne pole wokół zawartego w elemencie tekstu. Blok tekstowy zamieszczony w elemencie `p` postrzegany jest przez CSS jako dłuża linijka tekstu, nawet jeśli sam tekst jest rozłożony na wiele wierszy, by dostosować się do kontenera. Dla jasności, na **rysunku 4.6** obramowanie kontenera (czyli akapitu) oznaczone jest na czerwono, a ramka pola tekstowego na zielono. Właściwości tekstowe nadawane są tekstowi z zielonej ramki.



RYSUNEK 4.6. Tekst zawarty w długiej, wąskiej linii często rozkładany jest na wiele wierszy

Kod tekstu z tego przykładu wygląda następująco:

```
<p><span>Oto długi akapit...</span></p>
```

i korzysta z następujących stylów.

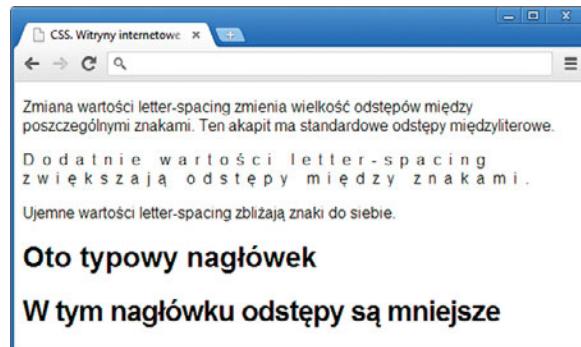
```
p {border:3px solid red;}  
span {border:1px solid green;}
```

Zauważ, że rozłożone na wiele wierszy pole tekstowe zamknięte jest jedynie na początku pierwszej linijki i na końcu ostatniej. Świadomość tego pozwoli Ci na szybsze uzyskanie pożądanych efektów. Jeśli na przykład chcesz utworzyć wcięcie pierwszego wiersza akapitu, to możesz zastosować właściwość `text-indent`, tak jak zrobiłem to na **rysunku 4.5**, aby przesunąć punkt początkowy pola tekstowego. Kolejne wiersze nie mają wcięć, ponieważ dla CSS są jedynie częścią jednego, długiego ciągu tekstowego.

Jeśli chcesz, by cały akapit był wcięty, to musisz zdefiniować jego właściwość `margin-left`. Innymi słowy, musisz przesunąć cały kontener w prawo. Wystarczy, byś pamiętał, że właściwości tekstowe nadawane są długiemu, cienkiemu, węzykowatemu polu tekstu, a nie polu jego kontenera.

Właściwość `letter-spacing` decyduje o trackingu. Tracking jest pojęciem z zakresu typografii, które odnosi się do odstępów międzyliterowych pomiędzy wszystkimi znakami w bloku tekstu. Różni się to od kerningu, który odnosi się do dostosowywania odstępów między dwoma konkretnymi znakami.

RYSUNEK 4.7. Oto jak zmiana wartości `letter-spacing` wpływa na wygląd tekstu



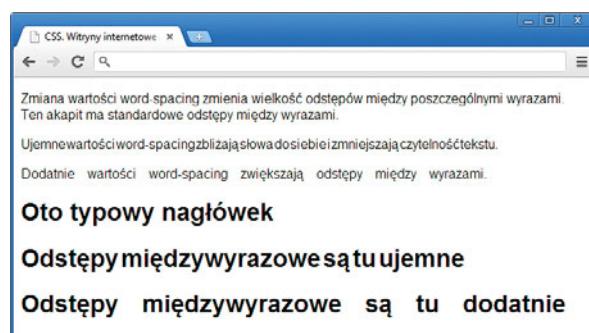
Domyślne odstępy międzyliterowe wydają się większe, gdy tekst jest zapisany większym fontem, więc ścieśnienie odstępów w nagłówkach dodaje stronie finezji. Zauważ, że odstępy międzyliterowe tekstu i nagłówka z **rysunku 4.7** pomniejszyłem jedynie o `.05 em` (czyli jedną dwudziestą firetu); gdybym je ścieśnił bardziej, litery zaczęłyby na siebie nachodzić.

Właściwość `word-spacing`

 Kiedy używasz dużych odstępów międzyliterowych, to trudniej odróżnić odstępy międzywyrazowe. W takiej sytuacji warto także nieco zwiększyć odstępy międzywyrazowe.

RYSUNEK 4.8. Akapity i nagłówki z normalnymi, ujemnymi i dodatkowymi wartościami odstępów

Regulowanie odstępów międzywyrazowych jest bardzo podobne do określania odstępów międzyliterowych. CSS traktuje każdy znak lub grupę znaków otoczonych białymi znakami jako słowo. Z regulacją odstępów międzywyrazowych można przesadzić nawet bardziej niż przy określaniu odstępów międzyliterowych, znacznie tym samym zmniejszając czytelność tekstu (**rysunek 4.8**).



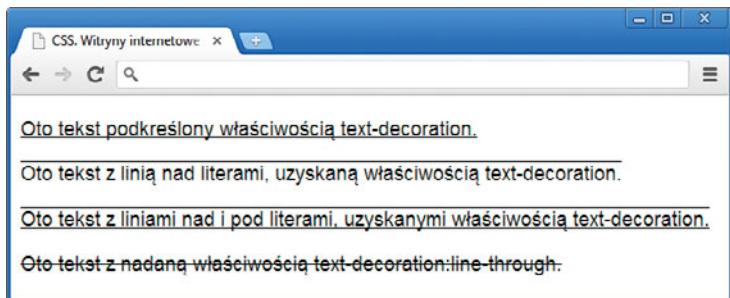
Właściwość `text-decoration`

Wartości: `underline`, `overline`, `line-through`, `blink`, `none`

Przykład: `.retailprice {text-decoration:line-through;}`

Przykłady zastosowania powyższych wartości, z wyjątkiem `blink`, widać na **rysunku 4.9**. Wartość `blink`, która sprawia, że tekst migą, jest niezmiernie irytująca, więc należy jej używać rzadko, a najlepiej wcale.

RYSUNEK 4.9. Oto różne wartości `text-decoration`, która to właściwość najbardziej się jednak przydaje do określania podkreśleń odnośników



 Zauważ, że użytkownicy są tak przyzwyczajeni do oznaczania odnośników podkreśleniami, że wywołasz u nich frustrację i zmusisz do bezsensownego klikania, jeśli stworzysz podkreślony tekst, który wcale odnośnikiem nie jest.

Tej właściwości używa się przede wszystkim do określania podkreślenia odnośników. Poniżej widnieje przykład, w którym usuwam podkreślenia z odnośników w pasku nawigacyjnym. To oczywiste, że tekst takiego paska można kliknąć, więc podkreślenie jedynie zaśmiecałoby układ graficzny. Pojawia się ono jednak wtedy, kiedy użytkownik najeżdża kursem na odnośnik, żeby dać mu jakąś informację zwrotną co do jego działań.

```
nav a {text-decoration:none;}  
a:hover {text-decoration:underline;}
```

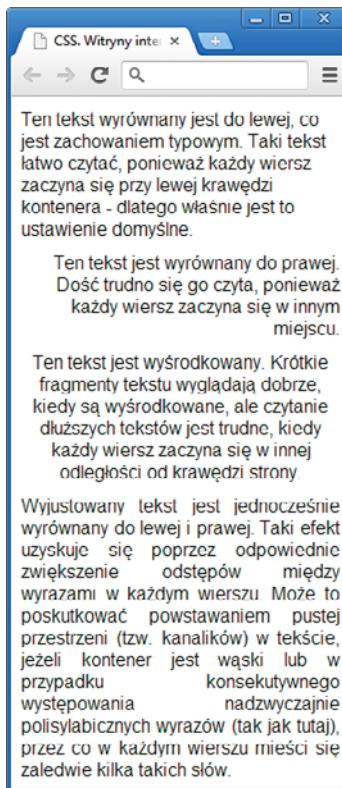
Właściwość `text-align`

Wartości: `left`, `right`, `center`, `justify`

Przykład: `p {text-align:right;}`

Ta właściwość obsługuje jedynie cztery wartości: `left`, `right`, `center` i `justify` i odpowiada za rozmieszczanie tekstu elementu w poziomie. Zauważ, że wartość `center` może także służyć do wyśrodkowania mniejszego elementu o określonej szerokości lub obrazu w obrębie większego elementu. Na **rysunku 4.10** widać cztery wartości właściwości `text-align` w działaniu.

RYSUNEK 4.10. Cztery wartości `text-align`



Właściwość `line-height`

 Termin „leading” wziął się z tego, że u poczatków druku do tworzenia odstępów między wierszami używano pasków z ołówkiem (ang. „lead”).

Wartości: dowolna wielkość (nie trzeba podawać jednostki)

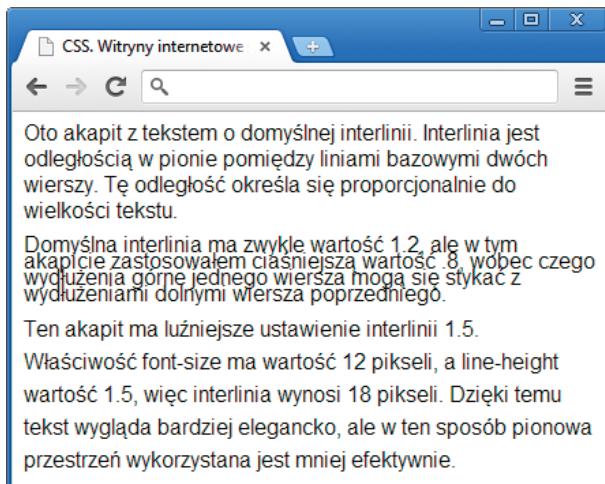
Przykład: `p {line-height: 1.5;}`

Właściwość `line-height` jest odpowiednikiem typograficznego pojęcia *leadingu* w CSS. Leading tworzy odstępy między wierszami bloku tekstu.

Interlinia rozmieszczana jest ponad i pod tekstem. Jeśli więc masz tekst o wielkości 12 pikseli z interlinią o wielkości 20 pikseli, to przeglądarka dodaje 4-pikselowy odstęp pod i nad wierszem, aby uzyskać łączną wysokość 20 pikseli.

W przypadku pojedynczego wiersza tekstu, takiego jak nagłówek, właściwość `line-height` działa jak margines, a duże nagłówki (takie jak `h1` i `h2`) mają domyślnie dość sporą interlinię. Warto o tym pamiętać, ponieważ czasami nawet po usunięciu marginesów i dopełnieniu wciąż nie da się usunąć całej przestrzeni nad i pod nagłówkiem. W tym celu należy także zmniejszyć interlinię, wręcz tak, by była mniejsza od tekstu, tj. miała wartość mniejszą niż 1.

RYSUNEK 4.11. Zmiana standardowej wielkości interlinii jest prostym sposobem, by nadać stronie charakterystyczny wygląd



Jak widać na **rysunku 4.11**, domyślną interlinię najprościej zmienić przy użyciu właściwości zbiorczej **font**, dla której podaje się zestawioną wartość dla właściwości **font-size** i **line-height**. Oto przykład:

```
div#intro {font:1.2em/1.4 helvetica, arial, sans-serif;}
```

W tym przypadku wartość leadingu wynosi 1,4 wielkości fonta, czyli **1.2em**. Zauważ, że nie musisz podawać jednostki w rodzaju em czy pikseli przy tej części wartości, która odnosi się do **line-height** — wystarczy liczba. W tym przypadku CSS bierze obliczoną liczbę pikseli równającą się podanej wartości **1.2em** i mnoży ją przez 1,4, by uzyskać wielkość interlinii. Jeśli później zmienisz wielkość tekstu na 1,5 em, to interlinia wciąż będzie wielkości 1,4 pikseli obliczonych z wartości 1,5 em. Zauważ, że jeśli zdefiniujesz interlinię jednostkami bezwzględnymi w rodzaju pikseli i zwiększasz rozmiar tekstu, to wiersze mogą zacząć na siebie nachodzić.

Właściwość **text-transform**

Wartości: **none**, **uppercase**, **lowercase**, **capitalize**

Przykład: **p {text-transform:capitalize;}**

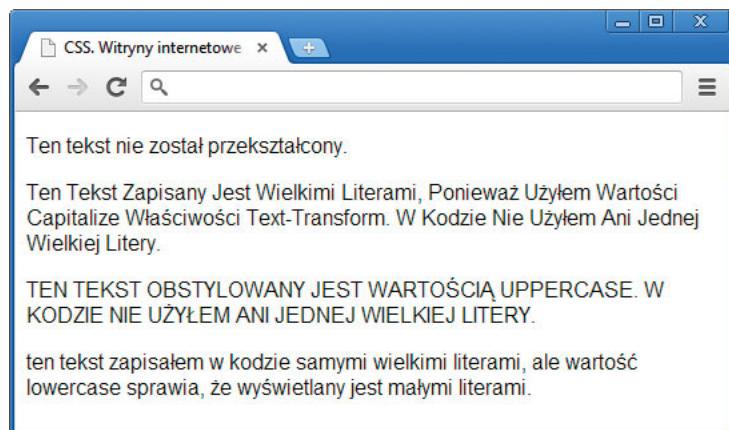
Właściwość **text-transform** zmienia wielkość liter tekstu w obrębie elementu. Możesz sprawić, by pierwsze litery wyrazów w linijce tekstu były zapisywane wielkimi literami albo żeby cały tekst był zapisany wielkimi lub małymi literami. Przykłady znajdziesz na **rysunku 4.12**.

Jeśli chcesz uzyskać tekst, którego słowa pisane są od wielkich liter, musisz podać wartość **font-variant:capitalize**.

Wartość **capitalize** sprawia, że każdy wyraz zapisywany jest od wielkiej litery. Pozwala to na uzyskanie stylu właściwego nagłówkom w angielskich reklamach, gazetach i czasopismach, z tym że przy ręcznej stylizacji tekstu pomniejsze słowa w rodzaju „of”, „as” czy „and”, tak jak w nagłówku „Tom and Jerry Go to Vegas”, zapisy-

wane są od małych liter. Automatyczna stylizacja CSS tworzy natomiast tekst „Tom And Jerry Go To Vegas”.

RYSUNEK 4.12. Właściwość `text-transform` pozwala stylizować tekst w sposób przypominający angielskie nagłówki



Właściwość `vertical-align`

Wartości: dowolna wielkość, `sub`, `sup`, `top`, `middle`, `bottom`

Przykład: `span {vertical-align:60%;}`

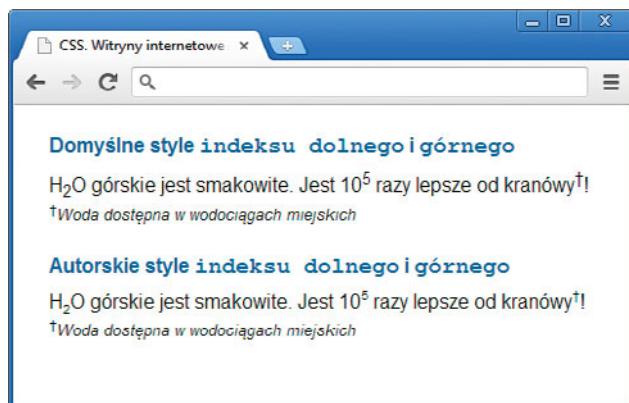
Właściwość `vertical-align` przesuwa tekst w górę lub w dół względem linii bazowej; zauważ jednak, że wpływa ona jedynie na elementy liniowe. Jeżeli chcesz wyrównać pionowo element blokowy, to musisz także zmienić właściwość `display` na `inline`. Właściwości tej używa się najczęściej do zapisywania znaków w indeksie dolnym i górnym we wzorach i wyrażeniach matematycznych, np. $x^4 - y^{-5}$ czy N_3O . Jest to również prawidłowy sposób zapisywania w tekście gwiazdek i innych oznaczeń przypisów. Nie lubię sposobu, w jaki większość przeglądarek domyślnie stylizuje znaki w górnich i dolnych indeksach — jak dla mnie wielkość fonta jest za duża, a same znaki są umieszczone za wysoko (lub za nisko, kiedy mamy do czynienia z indeksem dolnym). Wystarczy taki tekst odrobinę obstylować, aby uzyskać lepsze proporcje, które zachowują większą spójność z przeglądarki na przeglądarce.

Oto kod HTML przykładowy

```
<h4>Domyślne style <code>indeksu dolnego</code>
i <code>górnego</code></h4>
<p>H<sub>2</sub>0 górskie jest smakowite. Jest 10<sup>5</sup>
razy lepsze od kranowy<sup>&dagger;</sup>!</p>
<p class="customsmall"><sup>&dagger;</sup><em>Woda dostępna
w wodociągach miejskich</em></p>
<h4>Autorskie style <code>indeksu dolnego</code>
i <code>górnego</code></h4>
```

```
<p class="custom">H2O górskie jest smakowite.  
Jest 105 razy lepsze od kranowy&dagger;!  
</p>  
  
<p class="customsmall">&dagger;Woda dostępna  
w wodociągach miejskich</p>  
oraz kod CSS  
  
.custom sub {font-size:60%; vertical-align:-.4em;}  
.custom sup {font-size:65%; vertical-align:.65em;}  
.customsmall {font-size:.8em; vertical-align:1em;}
```

RYSUNEK 4.13. Zastosowanie indeksu górnego i dolnego zmienia położenie tekstu w pionie oraz jego wielkość



Chociaż znaczniki HTML `sup` i `sub` automatycznie wyświetlają tekst w indeksie górnym i dolnym, warto korzystać z nich w połączeniu z właściwościami `vertical-align` i `font-size` w celu uzyskania bardziej zadowalających rezultatów (rysunek 4.13). Niniejszym kończymy omówienie właściwości fontów i tekstu w CSS. Przyjrzyjmy się teraz, jak wczytuje fonty na strony internetowe.

Fonty internetowe

Możliwość osadzania fontów na stronie internetowej przy użyciu reguły `@font-face` jest już powszechnie obsługiwana funkcją CSS. Właściwość `@font-face` daje designerom znaczco rozbudowane możliwości doboru fontów poza tymi, które dostępne są na komputerach użytkowników. Możesz teraz zapewnić, że podane przez Ciebie fonty będą dostępne przeglądarce użytkownika, ponieważ będzie je mogła pobrać z serwera — nie musisz zatem liczyć na to, że użytkownik ma je zainstalowane na swoim systemie.

Oto trzy sposoby podawania fontów internetowych:

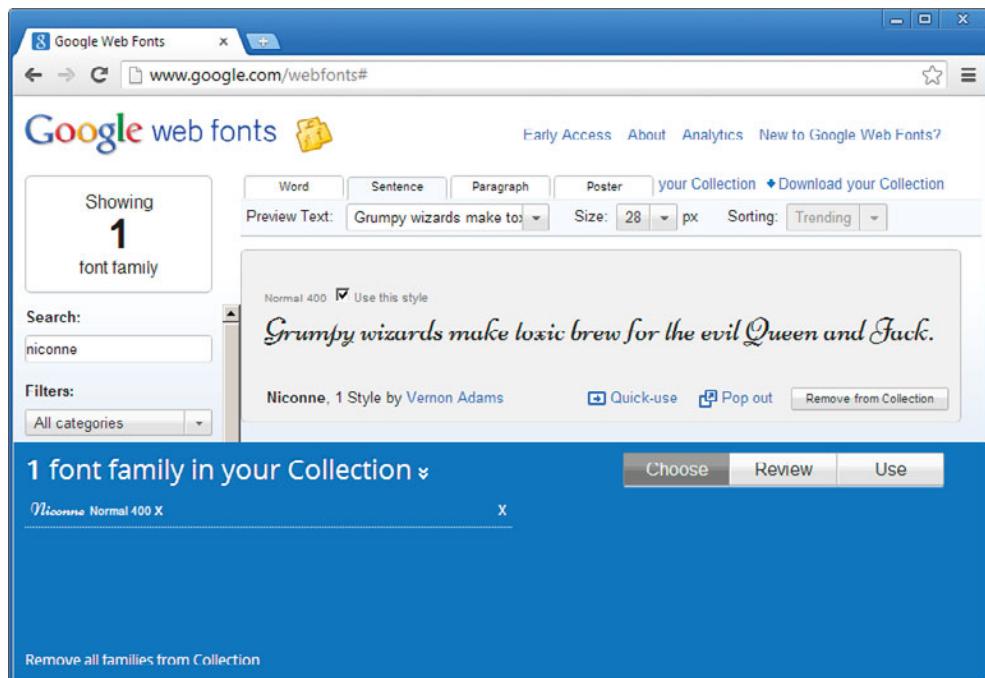
- Skorzystaj z zewnętrznej biblioteki fontów, takiej jak Google Web Fonts lub Typekit Adobe.
- Skorzystaj z gotowego zestawu `@font-face`.
- Stwórz zestaw `@font-face` z własnego fonta przy użyciu Font Squirrel.

Zacznijmy od najprostszej metody, czyli pobierania fonta z zewnętrznej biblioteki.

Internetowe biblioteki fontów

Dwie największe internetowe biblioteki fontów to Google Web Fonts, która oferuje za darmo ponad pięćset fontów, oraz Typekit Adobe, która zapewnia płatny dostęp do 739 rodzin fontów. Obydwie mają proste w obsłudze interfejsy.

Omówię teraz działanie tego procesu na przykładzie Google Web Fonts. Wejdź na stronę <http://www.google.com/webfonts>, znajdź odpowiedni font, kliknij przycisk *Add to Collection*, a następnie *Use* u dołu strony (**rysunek 4.14**). Google generuje wtedy znacznik `link` z odnośnikiem do wybranych fontów, który przeklejasz do znacznika `head` pliku HTML.



RYSUNEK 4.14. Dodaję font Niconne do mojej kolekcji, aby Google wygenerował do niego odnośnik

W pojedynczej linijce kodu można podać odnośniki do wielu fontów. Poniższy znacznik `link` odnosi się do fontów Anton, Niconne i Prata.

```
<link href='http://fonts.googleapis.com/css?  
family=Anton|Niconne|Prata' rel='stylesheet' type='text/css'>
```

Po dodaniu odnośnika do elementu `head` strony możesz korzystać z załączonych fontów tak jak zwykle. Po wyświetleniu strony font jest przekazywany stronie bezpośrednio z serwisu Google'a. Kod

```
h3 {font: 20px "Prata", serif;}
```

wyświetlany jest tak, jak na **rysunku 4.15**.

RYSUNEK 4.15. Użytkownik widzi teraz nagłówek wyświetlany fontem Prata

Ten tekst jest zapisany fontem Prata.

Wykorzystanie internetowej biblioteki fontów jest szybkim i niezawodnym sposobem na rozszerzenie ograniczonej palety fontów systemowych. Dodanie fonta z Google'a to kwestia paru minut, a wykorzystując je w swoich projektach, możesz być pewien, że Twoje strony będą wyświetlane w takich krojach, w jakich chcesz.

Gotowe zestawy @font-face

Drugi sposób osadzania fontów na stronach polega na wykorzystaniu reguły `@font-face`. Konieczne jest wtedy, by fonty były dostępne na serwerze, Twoim lub innym. Przekazywane w ten sposób fonty pobierane są przez przeglądarkę po wczytaniu pierwszej strony, która ją wykorzystuje. Trafiają wtedy do pamięci podręcznej przeglądarki i nie trzeba ich ściągać ponownie. Zauważ, że użytkownik nie może korzystać z takiego fonta w celu innym niż wyświetlanie stron, na których jest użyty.

Wykorzystanie metody `@font-face` wymaga większego nakładu pracy, ale pozwala na skorzystanie z praktycznie dowolnego kroju. Ze względu jednak na prawa autorskie musisz albo zakupić font, albo skorzystać z takiego, który jest wolny od opłat za eksplatację i który można zamieszczać na stronach.

Problemem metody `@font-face` jest to, że różne przeglądarki wymagają różnych formatów fontów. Firefox, oparte na silniku Webkit przeglądarki Safari i Chrome oraz Safari iOS na urządzenia przenośne od wersji 4.1 używają formatów OpenType (OTF) i TrueType

(TTF). Internet Explorer używa formatu Embedded OpenType (EOT), a niektóre inne, takie jak Safari sprzed wersji iOS 4.1, formatu Scalable Vector Graphics (SVG). W gotowych zestawach zwykle jednak dostępne są różne formaty, a poza tym możesz je wygenerować z fonta, który masz na komputerze; oczywiście, musisz się upewnić, że wolno Ci korzystać z niego w ten sposób.

Font Squirrel (www.fontsquirrel.com) oferuje rozległą bibliotekę fontów, dostarczanych w gotowych zestawach. W każdym zestawie znajduje się dany font we wszystkich formatach oraz powiązany kod CSS, który ma zapewnić, że każda przeglądarka otrzyma go w odpowiednim formacie. FontSquirrel udostępnia także konwerter, który pozwala na wczytanie i przekształcenie dowolnego fonta w zestaw.

Oto przykład kodu CSS `@font-face` dla kroju Ubuntu Titling Bold z FontSquirrel. Podany tu format kodu działa jednak również z fontami z innych źródeł.

```
@font-face {  
    font-family: 'UbuntuTitlingBold';  
    src: url('UbuntuTitling-Bold-webfont.eot');  
    src: url('UbuntuTitling-Bold-webfont.eot#iefix')  
        format('embedded-opentype'),  
        url('UbuntuTitling-Bold-webfont.woff')  
        format('woff'),  
        url('UbuntuTitling-Bold-webfont.ttf')  
        format('truetype'),  
        url('UbuntuTitling-Bold-webfont.  
            svg#UbuntuTitlingBold')  
        format('svg');  
    font-weight: normal;  
    font-style: normal;  
}
```

nazwa fonta, do którego odniesiesz się w liście fontów



Istnieje również opracowany przez guru webdesignu Paula Irisha, działający na różnych przeglądarkach, interesujący wariant kodu `@font-face`, który zapobiega myleniu fontów zainstalowanych w systemie użytkownika z tymi, które są pobierane, a mają jednakowe nazwy (co samo w sobie rzadko się zdarza). Znajdziesz go na stronie <http://paulirish.com/2009/bulletproof-font-face-implementation-syntax>.

Po dodaniu tego kodu do strony możesz się do niego normalnie odnieść przy użyciu reguły `font-family`, używając nazwy zdefiniowanej wartością `font-family` w regule `@font-face`.

Własne zestawy @font-face



Jeżeli chcesz lepiej zrozumieć działanie @font-face, polecam wpis blogowy Tima Browna How to Use CSS @font-face (<http://nicewebtype.com/notes/2009/10/30/how-to-use-css-font-face>).

Czasami musisz wykorzystać w projekcie konkretny font — często zdarza się to, kiedy klient wymaga użycia firmowego kroju pisma na stronie, którą projektujesz. O ile licencja zezwala na wykorzystanie kroju jako fonta internetowego (sprawdź, czy w licencji znajduje się zgoda, lub zadaj zapytanie firmie, która go stworzyła), możesz go przekształcić w zestaw @font-face na stronie Font Squirrel (<http://www.fontsquirrel.com/fontface/generator>). Wystarczy postępować zgodnie z instrukcją, żeby po paru minutach pobrać zestaw @font-face, który można wczytać prosto na serwer.

Zanim przejdę do przykładów projektowania typograficznego, chciałbym powiedzieć kilka rzeczy o osadzanych fontach. Dopóki wszyscy producenci przeglądarek nie zdecydują się na korzystanie z jednego formatu plików z fontami (a powinien to być OpenType), będziesz musiał sobie radzić z zawiłościami związanymi z korzystaniem z różnych formatów. Wszystkiego o składni deklaracji @font-face z wieloma fontami oraz o tym, jak zapewnić, by dostarczała przeglądarce Internet Explorer plik o wymaganym formacie .eot, dowiesz się z bloga Fontspring (<http://www.fontspring.com/blog/fixing-ie9-font-face-problems>). Fontspring zajmuje się także sprzedażą fontów, których licencja zezwala na korzystanie z nich w ramach deklaracji @font-face.

Od początku istnienia internetu designerzy — o ile nie dokładali wielkich starań — musieli się ograniczać do fontów dostępnych ogólnie na systemach operacyjnych pecetów i macintoshów. Wyczekiwane od dawna wdrożenie @font-face we wszystkich współczesnych przeglądarkach, w tym w IE9 i nowszych wersjach, dało wreszcie webdesignerom dostęp do wszelkich krojów, z których można korzystać w druku. W przypadku starszych przeglądarek, które nie obsługują reguły @font-face, rozwiązanie jest proste: użytkownicy takich przeglądarek widzą strony wyświetlane w kolejnym foncie z listy, więc w drugiej kolejności po fontach preferowanych musisz koniecznie podawać inne, bardziej rozpowszechnione fonty znajdujące się na komputerach użytkowników.

Stylizacja tekstu

Czas wykorzystać zdobytą wiedzę o fontach i tekście w praktyce. Zakończę ten rozdział trzema przykładami tworzenia estetycznych układów typograficznych, od szybkich i prostych, po przemyślane i wysublimowane.

Typografia wiąże się z pewnym rytmem, który określa wygląd tekstu umieszczonego na stronie, zwykle w odniesieniu do jakiejś siatki. Dobry rytm zwiększa płynność, z jaką wzrok porusza się po stronie.

Zacznijmy od prostej i podstawowej stylizacji tekstu. Zamiast korzystać przy rozmieszczaniu tekstu z siatki, zwyczajnie rozmiścimy wszystkie elementy proporcjonalnie do wielkości ich tekstu. To ćwiczenie pokaże Ci, jak odpowiednio szybko uzyskać pożądany rezultat.

Podstawowy układ tekstu

Jak widziałeś w rozdziale 1., domyślne style przeglądarki, które określają wygląd nagłówków, akapitów, list i innych elementów tekstowych, przewidują bardzo szeroki zakres wielkości, a także zbyt wielkie marginesy pionowe. Aby zobrazować zmianę domyślnego stylu w coś bardziej estetycznego, posłużymy się kodem zawierającym powszechnie używane elementy tekstowe.

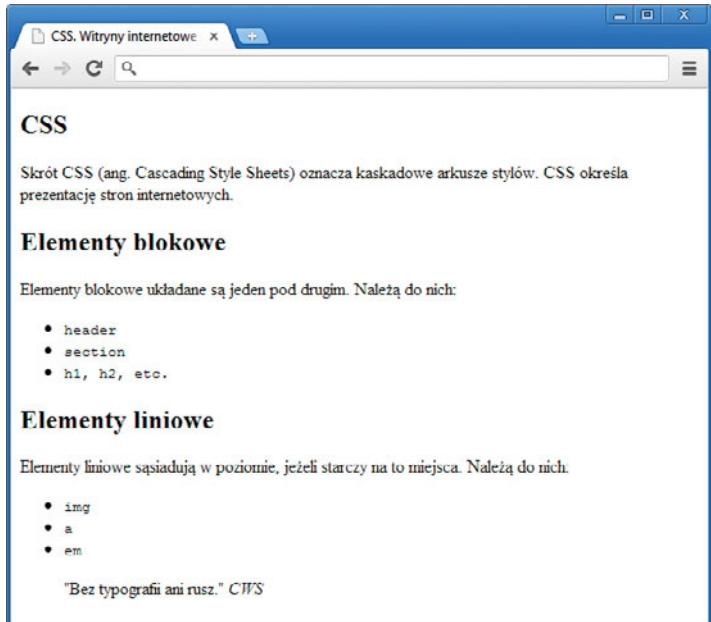
```
<article>
    <h1>CSS</h1>
    <p>Skrót CSS (ang. Cascading Style Sheets) oznacza kaskadowe arkusze stylów. CSS określa prezentację stron internetowych.</p>
    <h2>Elementy blokowe</h2>
    <p>Elementy blokowe układane są jeden pod drugim. Należą do nich:</p>
    <ul>
        <li><code>header</code></li>
        <li><code>section</code></li>
        <li><code>h1, h2 itd.</code></li>
    </ul>
    <h2>Elementy liniowe</h2>
    <p>Elementy liniowe sąsiadują w poziomie, jeżeli starczy na to miejsca. Należą do nich:</p>
    <ul>
        <li><code>img</code></li>
        <li><code>a</code></li>
        <li><code>em</code></li>
    </ul>
```

```
<blockquote>
    <q>Bez typografii ani rusz.</q><cite>CWS</cite>
</blockquote>
</article>
```

Na rysunku 4.16 widać, jak ten kod wygląda w przeglądarce.

RYSUNEK 4.16.
Nieobstyłowany kod nie jest szczególnie atrakcyjny

 Właściwość `font-size` o wartości `1em` zwyczajnie określa domyślną wielkość i niczego jeszcze nie zmienia. Muszę jednak zadeklarować wielkość fonta i jego rodzinę we właściwości zbiorczej `font`. Ponieważ pracuję na tekście o wielkości określonej względową wartością `em`, to jeśli zechcę później zmienić ogólną wielkość tekstu na stronie, będę mógł to zrobić poprzez wprowadzenie w tym miejscu jednej zmiany.



The screenshot shows a browser window with the title "CSS. Witryny internetowe". The page content is as follows:

CSS

Skrót CSS (ang. Cascading Style Sheets) oznacza kaskadowe arkusze stylów. CSS określa prezentację stron internetowych.

Elementy blokowe

Elementy blokowe układane są jeden pod drugim. Należą do nich:

- header
- section
- h1, h2, etc.

Elementy liniowe

Elementy liniowe sąsiadują w poziomie, jeżeli starczy na to miejsca. Należą do nich:

- img
- a
- em

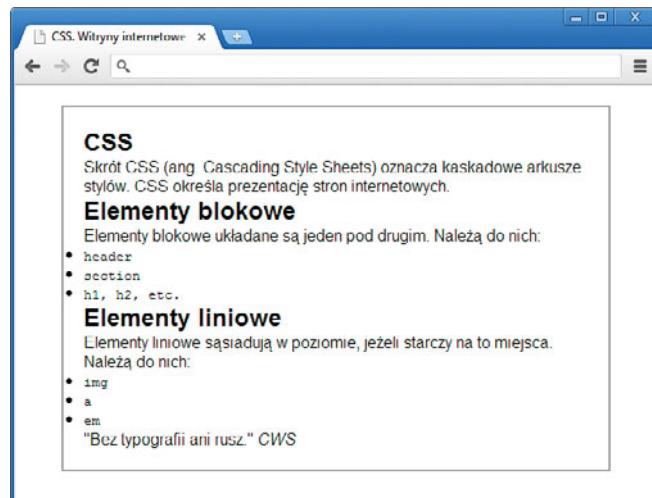
"Bez typografii ani rusz." CWS

Oto kilka fragmentów kodu, które pozwolą na szybkie uzyskanie przyjemniejszej oprawy graficznej. Po pierwsze, usuńmy marginesy, które tworzą straszne odstępy między elementami, określmy ogólny font oraz nadajmy styl znacznikowi `article`, który zawiera wszystkie elementy i pełni rolę kontenera graficznego, obejmującego tekst i wyśrodkowującego go.

usuwa wszystkie marginesy → `* {margin:0; padding:0;}`
 określa rodzinę fonta i jego ogólną wielkość → `body {font:1.0em helvetica, arial, sans-serif;}`
 wyśrodkowane pole → `article {width:500px; margin:20px auto; padding:20px; border:2px solid #999;}`

Na rysunku 4.17 widnieje rezultat.

RYSUNEK 4.17. Usunięcie domyślnych marginesów znacząco zmniejsza wysokość treści



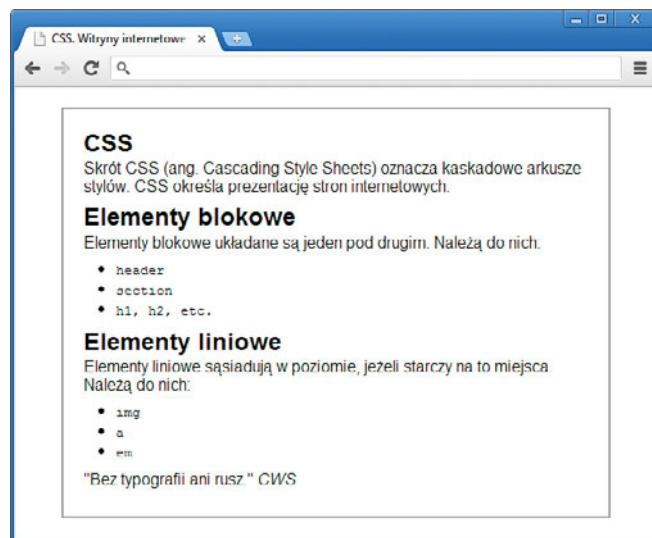
Należy teraz utworzyć odpowiednio rozmiędzone odstępy między elementami. Jako że usunąłem marginesy, punktory list zaczęły na marginesach nachodzić. To też trzeba naprawić.

przestrzeń wokół nagłówków → `h1, h2, h3, h4, h5, h6 {line-height:1.15em; margin-bottom:.1em;}`

przestrzeń wokół innych elementów tekstowych → `p, ul, blockquote {line-height:1.15em; margin-bottom:.75em;}`

wcięcia list → `ul {margin-left:32px;}`

RYSUNEK 4.18. Dodatkowe odstępy pod akapitami



Jak widać na [rysunku 4.18](#), zmniejszyłem wartość `line-height` wszystkich elementów. Interlinia jest teraz zaledwie odrobinę większa od wysokości tekstu. Dzieje się tak, ponieważ odstępy określone wartością `line-height` dodawane są na równi ponad i pod tekstem. Wolę jednak, by dodatkowa przestrzeń znajdowała się jedynie *pod* elementami. W tym celu dodaję marginesy. Muszę jednak zachować jakąś interlinię, żeby kolejne wiersze akapitu (i nagłówki, gdyby zajmowały więcej niż jeden wiersz) nie nachodziły na siebie.

Zauważ, że określiłem tylko dwa ustawienia marginesów, nadając im konkretną wartość odnoszącą się do wielkości fonta elementów. Nagłówkom nadałem bardzo małe dolne marginesy (równe 15% wielkości ich fontów), aby przylegały do następujących po nich elementów. Wszystkim pozostałym elementom tekstowym nadałem większe dolne marginesy (równe 75% wielkości fonta), aby w lay-outcie następowała po nich odpowiednia ilość pustej przestrzeni.

Na koniec, chciałbym lepiej wyważić nagłówki w taki sposób, żeby większe wyróżniały się, a mniejsze nie ginęły w reszcie tekstu; chcę także zwiększyć rozmiar liniowych elementów `code`.

wielkość tekstu nagłówka ————— | `h1 {font-size:1.9em;}`
`h2 {font-size:1.6em;}`
`h3 {font-size:1.4em;}`
`h4 {font-size:1.2em;}`
`h5 {font-size:1em;}`
`h6 {font-size:.9em;}`

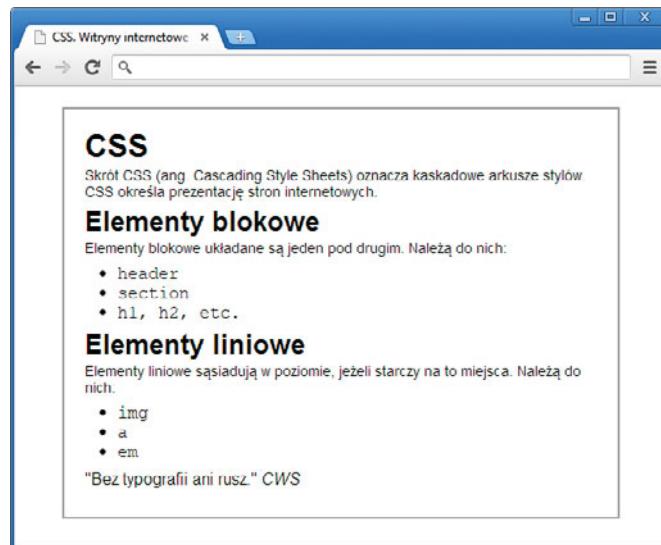
wielkość tekstu akapitu ————— | `p {font-size:.9em;}`

wielkość tekstu znacznika
code (domyślnie za mała) ————— | `code {font-size:1.3em;}`

Choć powyższy przykład nie jest szczególnie skomplikowany, widać, że odrobiną stylizacji tekstu wystarczy, by znacznie ulepszyć wygląd strony i zwiększyć jej czytelność ([rysunek 4.19](#)). Przejrzyjmy się teraz, jak uzyskać bardziej wysublimowany efekt przy użyciu siatek.

RYSUNEK 4.19.

Po powiększeniu nagłówków i tekstu znaczników code strona jest bardziej estetyczna, a hierarchia informacji łatwiejsza do zrozumienia



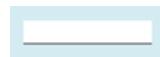
Stylizowanie tekstu w siatce

Wykorzystanie siatki do rozłożenia tekstu nadaje stronie rytm i płynność wizualną. Ponieważ omawiam w tym rozdziale kwestię tekstu, skoncentruję się na wykorzystaniu siatki do stworzenia *pionowego* rytmu tekstu.

W tym przykładzie utworzę layout oparty na pionowej, 18-pikselowej siatce, względem której wyrównam wszystkie elementy. Ponieważ w tle elementu — w tym wypadku mam na myśli **body** — można umieścić obraz, tymczasowo umieszczę na stronie prostą siatkę.

Użyłem Adobe Fireworks (choć sam możesz skorzystać z dowolnego programu) do utworzenia białego prostokąta o wymiarach 100×18 pikseli z szarą, szeroką na piksel linią u dołu. Zapisałem go w formacie *.png* (choć równie dobrze mógłby to być format *.jpg* lub *.gif*), nadając mu nazwę *grid_18px.png*. Na **rysunku 4.20** widać ten obraz na jasnoniebieskim tle.

RYSUNEK 4.20. Obraz, który umieszczę w tle strony. U dołu prostokąta znajduje się cienka, szara linia

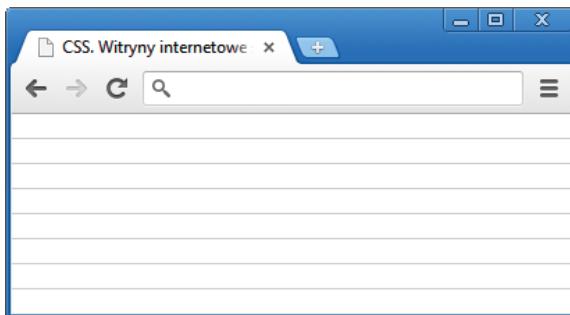


Zamieszczam ten obraz w tle elementu **body**

dodaje linie siatki  **body {background-image:url(images/grid_18px.png);}**

gdzie zostaje powtórzony wzdłuż i wszerz strony (**rysunek 4.21**).

RYSUNEK 4.21. Powtarzający się obraz zamieszczony w elemencie `body` tworzy tło, względem którego można pionowo ułożyć tekst



Mając już poziome linie siatki w tle, mogę się zabrać za rozmieszczenie elementów tekstowych przy ich pomocy.

W tym przykładzie użyłem kilku standardowych elementów tekstowych. Wystarczy jednak przywyknąć do tego rodzaju pracy, żeby z łatwością stworzyć arkusz stylów z pełnym zestawem elementów tekstowych wyrównanych względem siatki, którego można używać jako podstawy dla dowolnej strony.

Zacznę od prostego akapitu

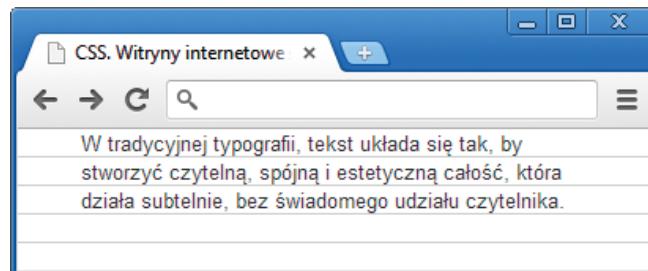
`<p>W tradycyjnej typografii...</p>`

i kodu CSS:

```
usuwa dopełnienia i marginesy ————— * {margin:0; padding:0;}
wszystkich elementów
body {
    dodaje linie siatki ————— background-image:url(images/grid_18px.png);
    definiuje font ————— font:100% helvetica, arial, sans-serif;
    duże marginesy z lewej i z prawej ————— margin:0 40px 0;
    tworzą na potrzeby przykładu
    kolumnę tekstu
    }
    p {
        określa wielkość tekstu ————— font-size:13px;
        określa interlinię o wielkości ————— line-height:18px;
        równej odstępom między
        liniami siatki
    }
```

Zauważ, że wartość `line-height` tekstu dostosowałem do wielkości siatki, czyli 18 pikseli. Po usunięciu wszystkich domyślnych marginesów i dopełnień wiem już, że wiersze będą rozmieszczone w 18-pikselowych odstępach (rysunek 4.22).

RYSUNEK 4.22. 18-pikselowa interlinia sprawia, że odstępy między wierszami zgadzają się z siatką



Kontenerowi `body` nadam teraz 4-pikselowe dopełnienie, aby przepchnąć ten element w dół i wyrównać linię bazową jego tekstu z siatką. Po dorównaniu tego elementu do siatki nietrudno będzie wyrównać kolejne elementy. Dodam jeszcze 22 piksele (4 + 18) u góry elementu `body`, aby uzyskać pusty wiersz i nieco rozluźnić układ.

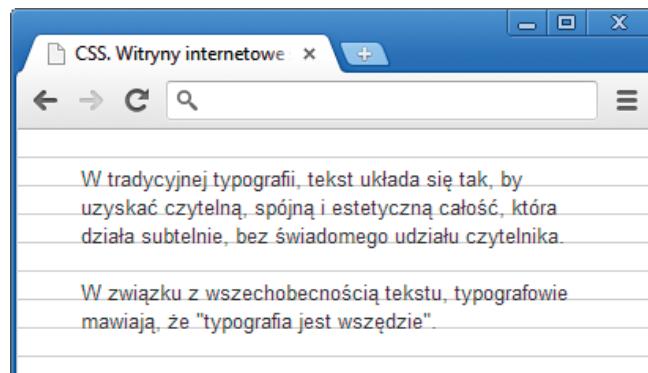
`padding-top:22px;`

Dodam jeszcze następującą deklarację do stylu akapitu:

```
określa wielkość tekstu —————| font-size:13px;
nadaje interlinii wielkość —————| line-height:18px;
odstępów między liniami siatki —————| margin-bottom:18px;
} 
```

Tworzy to dokładnie jeden pusty wiersz siatki pomiędzy akapitami. Dodam drugi akapit, aby pokazać efekt tych dwóch zmian (rysunek 4.23).

RYSUNEK 4.23. Po nadaniu elementowi body dopełnienia tekst jest idealnie wyrównany z siatką



Po wyrównaniu tekstu z siatką i właściwym rozmieszczeniu akapitów określeć wielkość tekstu innych elementów. Zaczynając od znacznika **h3**, któremu nadam wielkość 18 pikseli. Rzecz jasna, jego właściwość **line-height** również będzie miała wartość 18 pikseli, żeby zajmował dokładnie jeden wiersz w siatce. Aby sprawdzić, czy jego położenie jest właściwe, umieszczę go w kodzie pomiędzy dwoma akapitami.

```
<p>W tradycyjnej typografii...</p>
```

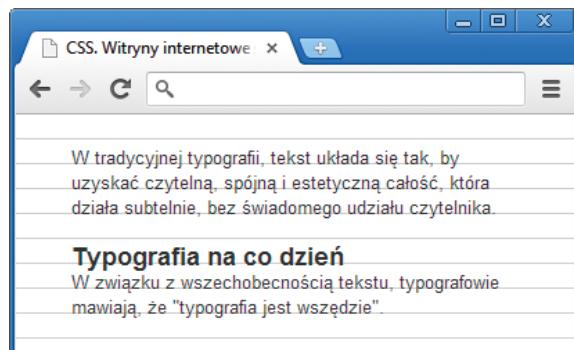
```
<h3>Typografia na co dzień</h3>
```

```
<p>W związku z wszechobecnością tekstu...</p>
```

Oto kod CSS nowego nagłówka:

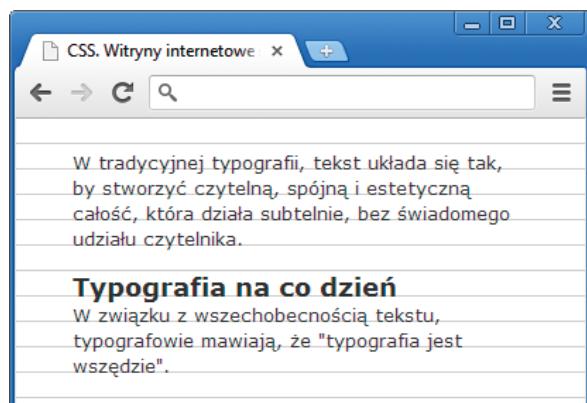
```
h3 {font-size:18px; line-height:18px;}
```

RYSUNEK 4.24. Linia bazowa elementu **h3** znajduje się nieco poniżej linii siatki



Jak widać, linia podstawowa nagłówka znajduje się kilka pikseli poniżej linii siatki, ale — co ciekawe — nie spycha kolejnego akapitu o taką samą odległość w dół (rysunek 4.24). Wynika to z tego, że o ile wartość **line-height** nagłówka jest poprawna, to przy tej wielkości i w tym fontem tekst jest w interlinii nieco przesunięty. Można to zmienić następującym kodem:

RYSUNEK 4.25. Mały ujemny margines górny i równy mu dodatni margines dolny podciągają nagłówki na właściwe miejsce w siatce



```
h3 {font-size:18px; line-height:18px; margin-top:-2px;
margin-bottom:2px;}
```

Ujemny margines górnny podciąga tekst w górę, a taka sama dodatnia wartość marginesu dolnego sprawia, że element pozostaje dokładnie tam, gdzie miał być ([rysunek 4.25](#)).

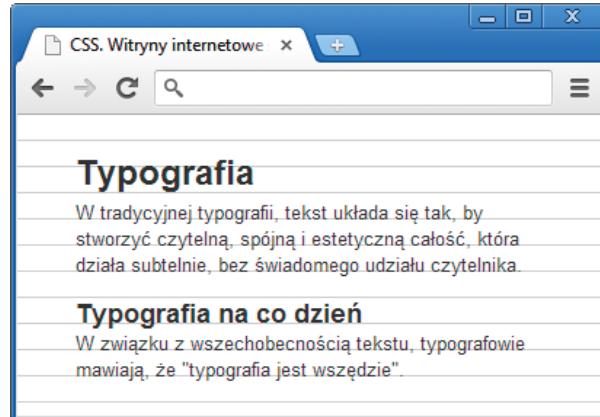
Do rozmieszczania elementów, które są większe od standardowych odstępów siatki, czyli zwykle nagłówków, trzeba użyć innej, ale podobnej techniki. Dla przykładu, zamieszczę w kodzie nagłówek **h1** wielkości 24 pikseli. Rzecznasza, 24-pikselowy tekst zajmuje więcej niż jeden rząd siatki, więc w tym przypadku nadam jego właściwości **line-height** wartość 36 pikseli, czyli wielkość dwóch rzędów. Element **h1** zamieszczę jako pierwszy element na stronie, czyli w miejscu, gdzie zwykle się znajduje.

```
<h1>Typografia</h1>
<p>W tradycyjnej typografii tekst jest układany...</p>
```

Zacznijmy od kodu CSS:

```
h1 {font-size:24px; line-height:36px;}
```

RYSUNEK 4.26. Ponieważ wartość **line-height** jest równa dwóm rzędom siatki, tekst nie opiera się o linię



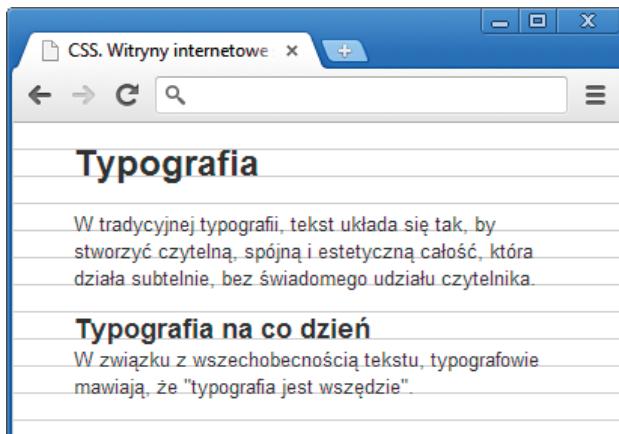
 Innym rozwiązaniem jest zamieszczenie linii bazowej nagłówka dokładnie pośrodku między dwiema liniami siatki. Może to wywołać przyjemny efekt przełamania rytmu, ale trzeba w takim przypadku dopilnować, by kolejny element był już dorównany do siatki.

Duży nagłówek tkwi w nie najlepszej pozycji między dwiema liniami ([rysunek 4.26](#)). Jego wydłużenia dolne naszłyby na tekst akapitu, gdybym przesunął go do najbliższej linii w dół, więc przesunę go w górę. Po serii prób i błędów okazało się, że odpowiednia odległość to 13 pikseli.

```
h1 {font-size:24px; line-height:36px; margin-top:-13px;
margin-bottom:13px;}
```

Pod tym elementem **h1** znajduje się teraz odrobina pustej przestrzeni, która oddziela go od tekstu ([rysunek 4.27](#)). Móglbym zrobić to samo z mniejszym nagłówkiem, ale wydaje mi się, że jednak lepiej wygląda, kiedy znajduje się bliżej następującego po nim elementu.

RYSUNEK 4.27. Nagłówek **h1** jest teraz poprawnie umieszczony w siatce

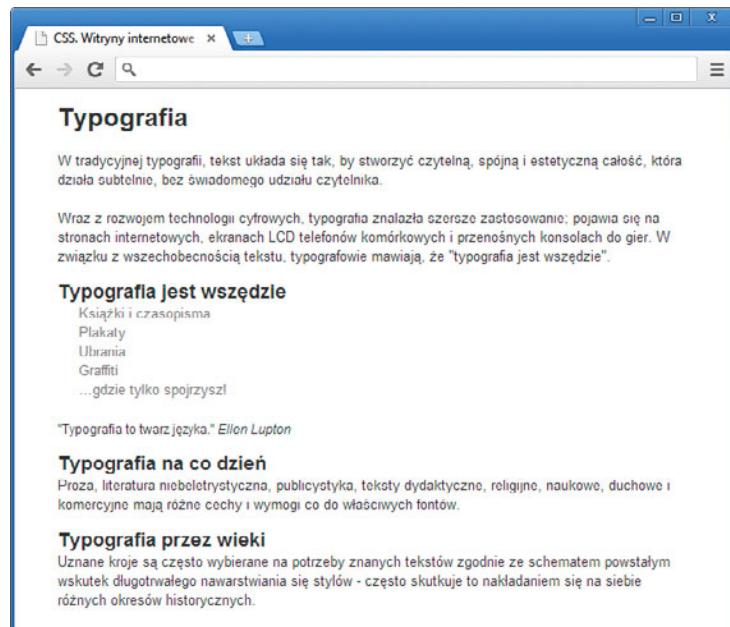


 Kod HTML z tego ćwiczenia znajdziesz na stronie <http://www.helion.pl/ksiazki/csswi3.htm>.

Przykład zakończę, dodając nagłówki o różnych wielkościach, nie-uporządkowaną listę oraz element **blockquote**, aby pokazać, jak wygląda bardziej opracowana strona po usunięciu siatki.

```
* {margin:0; padding:0;}
body {font:100% helvetica, arial, sans-serif;
backgroundimage: url(images/grid_18px.png); margin:0 20px 0;
padding:21px;}
p {font-size:14px; line-height:18px; margin-bottom:18px;}
h1 {font-size:24px; line-height:36px; margin-top:-13px;
margin-bottom:13px;}
h2 {font-size:18px; line-height:18px; margin-top:-2px;
margin-bottom:2px;}
h3 {font-size:16px; line-height:18px; margin-top:-2px;
margin-bottom:2px;}
ul {margin-bottom:18px;}
li {font-size:13px; list-style-type:none; padding:0 20px;
line-height:18px;}
a {color:#777; text-decoration:none;}
blockquote {font-size:12px; line-height:18px;
paddingtop:2px; margin-bottom:16px;}
```

RYSUNEK 4.28. Layout oparty na 18-pikselowej siatce



Jak widać na **rysunku 4.28**, w stronie ustrukturyzowanej na podstawie siatki jest coś bardzo przyjemnego. Stylizując siatkę tekstu na stronie, której treści tworzyć będzie ktoś inny, zyskujesz pewność, że elementy będą rozmieszczane poprawnie, bez względu na ich kolejność.

Typografia klasyczna

Na zakończenie rozdziału sformatuję krótki (przeredagowany na potrzeby przykładu) fragment *Psa Baskerville'ów*, wykorzystując wiele właściwości fontów i tekstu, które opisałem w tym rozdziale. Zobaczysz szereg technik, które pozwalają na uzyskanie wysokiej jakości typografii, w tym zastosowanie encji HTML, tworzenie odstępów międzyliterowych i międzywyrazowych, wykorzystanie kapitalików, pionowej siatki (tym razem 24-pikselowej) oraz fontów internetowych.

Kod jest dość prosty: dwa nagłówki, kilka akapitów i cytat.

<h2>Fragment książki</h2>

<h1>Pies Baskerville'ów</h1>

<p>Holmes wyciągnął rękę po manuskrypt i począł mu się bacznie przyglądać.</p>

<p>— Zwracam pańską uwagę, Watson, na kolejne używanie krótkiego i długiego s. Jest to jedna ze wskazówek, która mi pomogła do określenia daty. </p>

<p>Spojrzałem przez ramię na papier, pożółkły ze starości. Na pierwszej stronicy był nagłówek; „Baskerville-Hall”, a pod spodem data: „1742”. </p>

<p>— To zapewne potwierdzenie jakiejś legendy — wtrącił Holmes.</p>

<p>— Rzeczywiście — przyznał doktor Mortimer. — Legenda tyczy się rodu Baskerville'ów.</p>

<blockquote>

<q>Rozmaite pogłoski krążyły o Psie Baskerville'ów, lecz ja, pochodząc w prostej linii od Hugona Baskerville, słyszałem tę historię z ust mojego ojca…</q>

</blockquote>

<p>Doktor Mortimer skończył czytać tę niezwykłą opowieść, nasunął na oczy okulary i spojrzał na pana Sherlocka Holmesa.</p>

Zaznaczyłem w kodzie cztery różne encje HTML, którymi zapisuję znaki interpunkcyjne: cudzysłów otwierający („), zamykający (”), wielokropka (…) oraz pauzę (—).

Encje HTML

Na poniższych stronach znajdziesz tabele z encjami HTML:

<http://htmlhelp.com/reference/html40/entities/special.html>

<http://code.stephenmorley.org/html-and-css/character-entity-references-cheat-sheet>

Pod pierwszym z powyższych adresów znajdują się zarówno wartości encji HTML, jak i wartości szesnastkowe, których używa się przy podawaniu encji jako treści przy użyciu pseudoelementów `::before` i `::after`. Na przykład, podany w tabeli kod szesnastkowy cudzysłowu otwierającego `„` należy przedstawić na potrzeby pseudoelementu następująco:

```
e::before {content: "\201E";}
```

Zauważ, że wartość poprzedzona jest ukośnikiem. Szczególną wartość cudzysłowu zamykającego to w tym wypadku `\201D`.

W tekście zawartym w kodzie HTML etki i znaki mniejszości (`<`) **zawsze** musisz zastępować encjami, kolejno `&` i `>`. Jest tak, ponieważ znak `&` przeznaczony jest do zapisu encji, a `<` do otwierania znaczników HTML. Przeglądarki oczekują takich właśnie treści, kiedy trafiają na te znaki

KROK 1. — ZDEFINIOWANIE FONTÓW I SIATKI

W tym przykładzie ogół tekstu wyświetlam fontem Neuton z FontSquirrel. Pobrałem zestaw fontów, wczytałem go na swój serwer (ale zachowałem też na komputerze na potrzeby produkcji) oraz dodałem podaną regułę `@font-face` do arkusza stylów. Mogę go teraz podać w regule `font-family`.

```
@font-face {  
    font-family: 'Neuton';  
    src: url('Neuton/Neuton-Regular-webfont.eot');  
    src: url('Neuton/Neuton-Regular-webfont.eot?#iefix')  
        format('embedded-opentype'),  
        url('Neuton/Neuton-Regular-webfont.woff')  
        format('woff'),  
        url('Neuton/Neuton-Regular-webfont.ttf')  
        format('truetype'),  
        url('Neuton/Neuton-Regular-webfont.svg')  
        format('svg');  
    font-weight: normal;  
    font-style: normal;}  
* {margin:0; padding:0;}  
body {font-family:"Neuton", georgia, times, serif;  
background-color:#fff;  
margin:0 10% 0; background-image:url(grid_24px.png);}
```

Następnie wykonuję swoją standardową procedurę usunięcia wszystkich marginesów i dopełnień, wyznaczenia głównego fonta oraz dodania marginesów z lewej i z prawej strony; dodaję także tymczasową siatkę do wyrównywania tekstu, tak jak widać na rysunku 4.29.



RYSUNEK 4.29. Tekst i siatka do wyrównania

KROK 2. — STYLIZACJA NAGŁÓWKÓW

Teraz wszystkie elementy kolejno dorównuję do siatki. Chcę, by pierwszy, drobniejszy nagłówek kontrastował z największym nagłówkiem, któremu chcę nadać wielki, pochyły font. Mniejszy w takim razie zapiszę kapitalikami z jednakowymi odstępami międzyliterowymi.

```
body {font-family:"Neuton", georgia, times, serif;  
background-color:#fff; margin:29px 10% 0; background-  
image:url(grid_24px.png);}  
  
h2 {font-size:18px; line-height:24px; font-weight:bold;  
text-align:center; font-variant:small-caps; word-  
spacing:.5em; letter-spacing:.6em;}
```

Dodatkowe informacje o typografii

<http://ilovetypography.com>. Znajdziesz tu przemyślenia дизайнера Johna Boardleya na temat typografii oraz unikalne układy typograficzne na każdej stronie.

<http://www.thinkingwithtype.com>. Jest to strona towarzysząca książce *Thinking with Type* autorstwa Ellen Lupton. Znajdziesz na niej piękne, klasyczne przykłady wykorzystania typografii oraz informacje o krojach i ich rodzajach.

<http://webtypography.net>. Strona *The Elements of Typographic Style Applied to the Web — A practical guide to web typography* jest przewodnikiem po typografii internetowej. Znajduje się na niej schludny spis treści, przedstawiający popularne wskazówki i problemy z tego zakresu.

Najpierw zdefiniuję właściwość `font-variant` tego nagłówka, by tekst wyświetlany był w kapitalikach, a następnie zastosuję właściwości `word-spacing` i `letter-spacing`, by uzyskać pożądany efekt (rysunek 4.30).

RYSUNEK 4.30. Mały nagłówek jest teraz wyrównany w stosunku do siatki i obstyłowany przy użyciu właściwości fonta i tekstu



Przechodzę następnie do Google Web Fonts, by znaleźć krój pochylony Pinyon, którego wygląd jest adekwatny do tematyki tekstu. Przeklejam do nagłówka dokumentu HTML wygenerowany przez Google Web Fonts znacznik z odnośnikiem, aby móc się teraz odnieść do fonta w deklaracji `font-family`. Po raz kolejny korzystam z triku z ujemnymi i dodatnimi marginesami (rysunek 4.25), aby dokładnie wyrównać tekst z siatką. Rezultat widać na rysunku 4.31.

Zauważ, że właściwości `line-height` nadalem czterokrotną wartość odległości między liniami siatki.

RYSUNEK 4.31. Duży nagłówek jest teraz obstyłowany i dorównyany do siatki

```
<link href='http://fonts.googleapis.com/css?family=Pinyon+Script' rel='stylesheet' type='text/css'>
h1 {font-size:60px; line-height:96px; font-family: "Pinyon Script", cursive; margin:4px 0 -4px; text-align:center; font-weight:normal; position:relative;}
```



KROK 3. — OBSTYLOWANIE AKAPITU I CYTATU

Pierwszy akapit znajduje się nieco za wysoko, więc zmieńmy wielkość jego tekstu i — co najważniejsze — interlinię.

```
p {font-size:18px; line-height:24px;}
```

RYSUNEK 4.32. Określenie właściwości `line-height` akapitu pozwala na wyrównanie go względem siatki

Holmes wyciągnął rękę po manuskrypt i począł mu się bacznie przyglądać.
 — Zwracam pańską uwagę, Watson, na kolejne używanie krótkiego i długiego s. Jest to jedna ze wskazówek, która mi pomogła do określenia daty. Spojrzałem przez ramię na papier, pożółkły ze starości. Na pierwszej stronicy był nagłówek; „Baskerville-Hall”, a pod spodem data: „1742”.
 — To zapewne potwierdzenie jakiejś legendy — wtrącił Holmes.
 — Rzeczywiście — przyznał doktor Mortimer. — Legenda tyczy się rodu Baskerville’ów.
 "Rozmaite pogłoski krążły o Psie Baskerville’ów, lecz ja, pochodząc w prostej linii od Hugona Baskerville, słyszałem tę historię z ust mojego ojca..."
 Doktor Mortimer skończył czytać tę niezwykłą opowieść, nasunął na oczy okulary i spojrzał na pana Sherlocka Holmese.

Pierwsze trzy akapity są teraz wyrównane względem siatki, natomiast z kolejnymi tak już nie jest, ponieważ interlinię cytatu znajdującej się pod pierwszym akapitem również trzeba wyrównać (**rysunek 4.32**). Tekst w `blockquote` zawarty jest w liniowym znaczniku `q`, który domyślnie okala treść cudzysłowami. Elementowi `blockquote` nadam wcięcie, ale wielkość fonta i interlinię zdefiniuję dla znacznika `q`, ponieważ to w nim zawarty jest tekst.

```
blockquote {margin:0px 20%;}
```

```
q {font-size:18px; font-style:italic; line-height:24px;}
```

Wcięcie cytatu i zapisanie go kursywą urozmaica stronę. Zauważ, że po wyrównaniu znacznika `blockquote` pozostałe akapity też zostają wyrównane (**rysunek 4.33**).

RYSUNEK 4.33. Cytat został wyrównany do siatki

Holmes wyciągnął rękę po manuskrypt i począł mu się bacznie przyglądać.
 — Zwracam pańską uwagę, Watson, na kolejne używanie krótkiego i długiego s. Jest to jedna ze wskazówek, która mi pomogła do określenia daty. Spojrzałem przez ramię na papier, pożółkły ze starości. Na pierwszej stronicy był nagłówek; „Baskerville-Hall”, a pod spodem data: „1742”.
 — To zapewne potwierdzenie jakiejś legendy — wtrącił Holmes.
 — Rzeczywiście — przyznał doktor Mortimer. — Legenda tyczy się rodu Baskerville’ów.
 "Rozmaite pogłoski krążły o Psie Baskerville’ów, lecz ja, pochodząc w prostej linii od Hugona Baskerville, słyszałem tę historię z ust mojego ojca..."
 Doktor Mortimer skończył czytać tę niezwykłą opowieść, nasunął na oczy okulary i spojrzał na pana Sherlocka Holmese.

KROK 4. — UTWORZENIE INICJAŁU

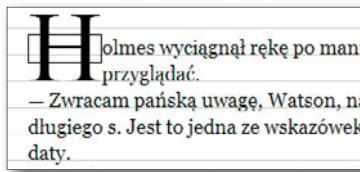
W pierwszym akapicie utworzę charakterystyczny inicjał z pierwszego znaku. Inicjały mogą przyjmować różne formy, ale w tym wypadku wyrównam jego górną krawędź z górną linią pierwszego wiersza akapitu, a dolną krawędź z linią bazową trzeciego wiersza.

Taki efekt zazwyczaj osiąga się poprzez umieszczenie pierwszej litery w elemencie `span`, ale nie jest to ani praktyczne, ani pewne rozwiązanie, gdy tekst pobierany jest z systemu zarządzania treścią. Opisana poniżej technika nie wymaga wprowadzenia jakichkolwiek zmian w kodzie HTML.

Muszę wybrać pierwszą literę pierwszego akapitu, który znajduje się po nagłówku `h1`. Używam w tym celu połączenia dwóch selektorów: pseudoelementu `::first-letter` i selektora brata `+`. Wybrany znak można powiększyć i zmienić w płynący pseudoelement, by przesunąć go w odpowiednie miejsce.

```
h1 + p::first-letter {font-family:times, serif; font-size: 90px; float:left; border:1px solid;}
```

RYSUNEK 4.34. Włączyłem widoczność ramki, dzięki czemu widać, że właściwość `line-height` inicjału odziedziczyła wartość mniejszej interlinii akapitu

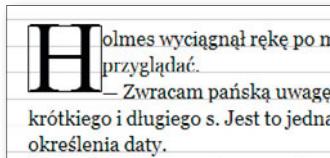


Właściwości `line-height` nadajełam wartości mniejszą od 1, aby interlinia szczelnie obejmowała inicjał i tym samym nie wymuszała na czwartym wierszu, by go oblewał.

Pierwszy znak jest teraz powiększony, ale nie znajduje się tam, gdzie bym chciał (rysunek 4.34). Postanowiłem sobie pomóc, wyświetlając ramkę pseudoelementu, ponieważ to właśnie jej pozycję muszę zdefiniować. Z wyglądu ramki wynika, że jej rozmiar pozwala jedynie na zawinięcie dwóch wierszy wokół niej, ponieważ dziedziczy ona swoją wielkość i położenie po akapicie. Muszę zmienić `line-height` pseudoelementu, aby jego ramka w całości okalała inicjał.

```
h1 + p::first-letter {font-family:times, serif;  
font-size:90px; float:left; line-height:.65; border:1px  
solid;}
```

RYSUNEK 4.35. Pole elementu ściśle teraz otacza inicjał

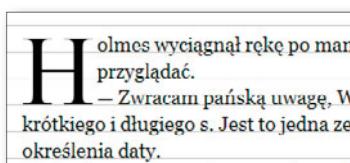


Zwiększenie wartości `line-height` sprawiło, że trzeci wiersz również zaczął oblewać inicjał (**rysunek 4.35**). Pozostaje jedynie utworzyć górne dopełnienie, aby przesunąć inicjał w dół i wyrównać go do linii bazowej trzeciego wiersza.

```
h1 + p::first-letter {font-family:times, serif; font-size:-  
90px; float:left; line-height:.65; padding:.085em 3px 0 0;}
```

Zauważ, że dodaję także 3 piksele prawego dopełnienia, aby utworzyć odstęp między inicjałem a tekstem akapitu. Ponadto usuwam obramowanie, ponieważ nie jest mi już potrzebne (**rysunek 4.36**).

RYSUNEK 4.36. Gotowy inicjał



KROK 5. — STYLIZACJA PIERWSZEGO WIERSZA

Inicjał jest gotowy, ale chciałbym uzyskać efekt płynnego przejścia między wielkim inicjałem a małym tekstem akapitu. Pierwszy wiersz tekstu akapitu zapiszę więc kapitalikami.

```
h1 + p::first-line {font-variant:small-caps;  
letter-spacing:.15em;}
```

Do zapisania pierwszego wiersza kapitalikami ponownie użyjęm selektora brata, tym razem w połączeniu z pseudoelementem `::first-line`. Dzięki użyciu pseudoelementu zamiast zapisu pierwszych kilku słów wielkimi literami zapis kapitalikami będzie się dostosowywał do zmian długości wiersza. Uzyskane w ten sposób estetyczne przejście między inicjałem a resztą akapitu widać na **rysunku 4.37**.

RYSUNEK 4.37. Pierwszy wiersz akapitu został zapisany kapitalikami



KROK 6. — WYKOŃCZENIE

Kiedy między akapitami nie ma odstępów, trudno oszacować, gdzie się który akapit zaczyna. Aby zachować konwencję typografii książkowej, zamiast tworzyć odstępy, akapitom następującym po innych akapitach nadam drobne wcięcia; akapit znajdujący się na początku nie potrzebuje wcięcia. Nie podobały mi się też bezbarwne cudzysłowy w elemencie `q`, więc zmienię domyślne pseudoelementy `::before` i `::after` tego znacznika na cudzysłowy z fonta Neuton. Poza tym nie potrzebuję już siatki, więc usunę ją ze znacznika `body`.

nadaje wcięcie każdemu akapitowi, który znajduje się po innym akapicie

`p + p {text-indent:14px;}`

cudzysłów otwierający

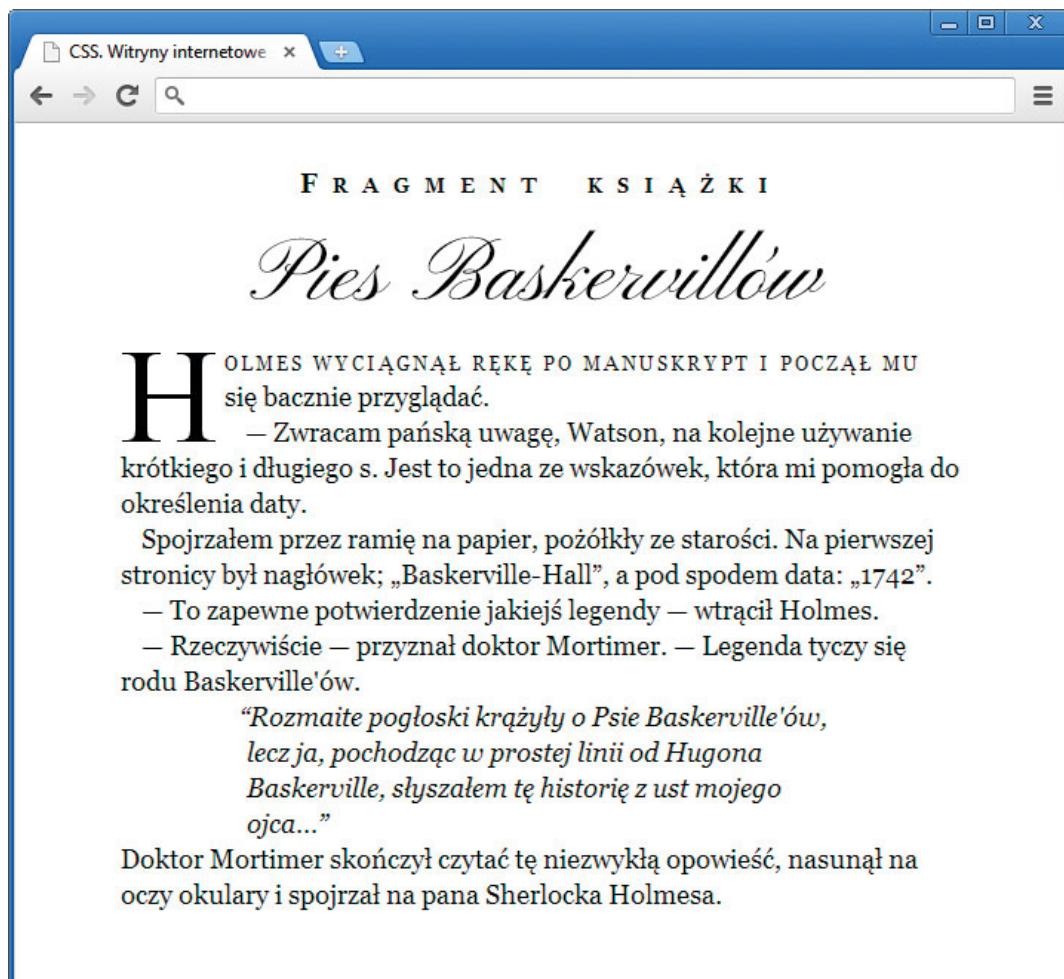
`q::before {content:"\201E"}`

cudzysłów zamkujący

`q::after {content:"\201D"}`

Warto obydwie te zmiany omówić szerzej. Wcięcia uzyskałem przy użyciu selektora brata, dzięki czemu są one nadawane jedynie tym akapitom, które następują po innych akapitach. Akapit zaczynający się od „Doktor Mortimer skończył czytać...” nie znajduje się za innym akapitem, tylko za elementem `blockquote`, więc nie ma wcięcia. Przejście od cytatu do tego akapitu jest natomiast dość płynne dzięki temu, że akapit jest wyrównany do marginesu ([rysunek 4.38](#)).

Cudzysłowy dodane przy użyciu pseudoelementów `::before` i `::after` trzeba zdefiniować encjami wyrażonymi wartościami szesnastkowymi. Wartości `content` nie mogę wyrazić normalnymi encjami HTML, ponieważ nie działałyby w tej sytuacji. Można tu używać jedynie encji szesnastkowych, w dodatku w nieco zmodyfikowanej postaci. Więcej szczegółów na temat encji znajdziesz w ramce „Encje HTML” we wcześniejszej części rozdziału.



RYSUNEK 4.38. Oto ukończona strona

Po naniesieniu tych drobnych poprawek layout jest gotowy. Może się wydawać, że sformatowanie tak małego fragmentu wymagało strasznie dużo pracy, ale style te można z łatwością nadać całej książce.

Podsumowanie

W tym rozdziale poznaleś wiele właściwości CSS odnoszących się do fontów i tekstu. Nauczyłeś się różnych sposobów podawania stronom fontów oraz zetknąłeś się z różnymi sposobami formatowania tekstu. W kolejnym rozdziale omówię szerzej przedstawione layouty i pokażę techniki tworzenia layoutów wielokolumnowych.

ROZDZIAŁ 5

Layouty

W TYM ROZDZIALE POKAŻĘ CI, jak tworzyć layouty wielokolumnowe. Na większości stron internetowych kolumny służą do tego, by zatrzymać możliwie najwięcej informacji „nad zgęściem”, czyli na obszarze widocznym bez przewijania strony. Zazwyczaj korzysta się z dwóch lub trzech kolumn, ale można się też spotkać z cztero-kolumnowymi layoutami. Niezależnie od konkretnej liczby kolumn w layoucie istnieją pewne podstawowe techniki i założenia, z którymi wiąże się tworzenie stron w ogóle. Omówię je w tym rozdziale.

Przedstawię Ci kilka różnych sposobów tworzenia layoutów stron. Poznasz metodę opartą na wykorzystaniu płynących kolumn, zawierających elementy `div`, której stosowanie było przez wiele lat powszechnie. Zademonstruję też kilka nowych technik, których można używać nawet przy tworzeniu wielkich stron, jako że obsługa CSS3 staje się coraz bardziej powszechna, a przeglądarki nieobsługujące CSS3 wychodzą z użycia. Możesz teraz korzystać z właściwości `box-sizing` zamiast osadzonych elementów `div` oraz z właściwości CSS3 `display`, które sprawiają, że elementy zachowują się jak tabele (bez konieczności wstawiania tabel do kodu HTML), aby bez problemu tworzyć płynne layouty z pełnoprawnymi kolumnami.

Omówię wady i zalety wymienionych powyżej i innych technik, a także kod zapasowy i skrypty potrzebne, by starsze przeglądarki (mam głównie na myśli IE8 i wcześniejsze wersje) mogły je obsługiwać. Dzięki temu sam będziesz mógł zdecydować, z których technik będziesz korzystać w pracy. Zaczniemy od kilku podstawowych projektów związanych z layoutami.

Podstawy tworzenia layoutów

 Popularny framework do tworzenia layoutów w CSS, 960 Grid (www.960.gs), jest, jak sama nazwa wskazuje, oparty na 960-pikselowej siatce.

Istnieją trzy podstawowe rodzaje wielokolumnowych layoutów: layouty o stałej szerokości, płynne i elastyczne.

- **Layouty o stałej szerokości** nie zmieniają rozmiarów, gdy użytkownik zmienia szerokość przeglądarki. Mają zwykle szerokość od 900 do 1100 pikseli. 960 pikseli to bardzo popularna szerokość, ponieważ mieści się na wszystkich nowoczesnych monitorach, a sama liczba 960 jest podzielna przez 16, 12, 10, 8, 6, 5, 4 i 3, dzięki czemu obliczenia jednolitej szerokości kolumn i innych elementów dają przyjemne, okrągłe wyniki.
- **Płynne layouty** zmieniają swoją szerokość, kiedy użytkownik zmienia szerokość okna przeglądarki. Choć pozwala to na sprawniejsze skalowanie layoutu na dużych monitorach, to sam tracisz pełną kontrolę nad layoutem, jako że przy zmianach wielkości okna zmienia się też interlinia oraz relacje między elementami. Środkowy obszar layoutu Amazon.com jest płynny — przy poszerzaniu kolumn wokół elementów treści pojawia się dodatkowa pusta przestrzeń, a obecnie boczny pasek nawigacyjny zmienia się w rozwijane menu, aby zapewnić treści właściwej odpowiednio dużo miejsca, kiedy użytkownik zwęża okno poniżej pewnej wartości.
Upowszechniająca się obsługa kwerend CSS, które pozwalają na podawanie przeglądarkom różnych stylów CSS w zależności od szerokości okna, sprawia, że płynne layouty zastępowane są layoutami, które przyjmują określone rozmiary w zależności od tego, jakiej wielkości jest okno przeglądarki użytkownika. Tworzenie stron, które mogą się dostosować do największych i najmniejszych ekranów, nazywamy „projektowaniem skalowalnym”, którego techniki omówię w rozdziale 8.
- Layouty **elastyczne** są podobne do płynnych. Kiedy użytkownik poszerza okno przeglądarki, takie layouty nie tylko zmieniają swoją szerokość, ale także wielkość wszystkich swoich elementów. Wywołuje to efekt ogólnego powiększenia. Nie zetknąłem się z wieloma dobrze przygotowanymi layoutami tego typu, a obsługa ich jest skomplikowane, więc nie będę ich omawiał. Skoncentruję się w tym rozdziale na layoutach o stałej szerokości i layoutach płynnych.

Przyjrzyjmy się teraz różnicy między wysokością a szerokością strony.

Wysokość i szerokość layoutu

Zanim zabiorę się za omówienie tworzenia layoutów, chciałbym Ci zwrócić uwagę na kwestię wysokości i szerokości layoutów, ponieważ zarządza się nimi zupełnie inaczej.

WYSOKOŚĆ LAYOUTU

Na ogół nie trzeba definiować wysokości strukturalnych elementów layoutu ani layoutów w ogóle. Określania wysokości elementów należy wręcz unikać. Jeśli już to zamierzasz zrobić, to musisz koniecznie mieć dobry powód, np. taki, że chcesz utworzyć na stronie element o położeniu zdefiniowanym wartościami bezwzględnymi.

Elementowi zazwyczaj najlepiej pozostawić domyślną wysokość `auto` dlatego, że dzięki temu będzie się on mógł powiększać w pionie, żeby zatrzymać w sobie dodawane treści. Powiększający się w ten sposób element może zepchnąć w dół elementy znajdujące się pod nim, wobec czego layout dostosowuje się odpowiednio do umieszczanych na stronie treści. Nadając elementowi określona wysokość, sprawiasz, że nadmiar treści będzie ucinały lub będzie wylewał się poza swój kontener, w zależności od ustawienia właściwości `overflow`.

SZEROKOŚĆ LAYOUTU

W odróżnieniu od wysokości, szerokość layoutu należy uważnie kontrolować, aby pasowała do szerokości odpowiednio wielkiego okna przeglądarki, a wiersze tekstu nie były ani za długie, ani za krótkie. Nieograniczone dodawanie dopełnień, obramowań i dużych elementów może sprawiać, że płyniące elementy będą się „wślizgiwać” pod siebie, jeśli będą szersze od swoich kontenerów, określających rozmiary layoutu.

O ile jednak zależy Ci na określeniu szerokości kolumn, nie powinieneś definiować szerokości zawartych w nich elementów z treścią, tylko pozwolić im zajmować całą szerokość kolumny. Jak już dobrze wiesz, elementy blokowe robią to domyślnie. Pozwól po prostu, by szerokość kolumn określała szerokość zawartych w nich elementów. Wszystko powyższe będzie bardziej zrozumiałe, kiedy pokażę, jak tworzyć layouty. Na razie jednak zapamiętaj, że z reguły należy kontrolować szerokość layoutu, ale treści elementów pozwolić na określenie wysokości.

Tworzenie kolumn

W tym podrozdziale pokażę, jak stworzyć layout z trzema kolumnami. Kiedy zrozumiesz sposób działania takiej struktury, będziesz w stanie tworzyć layouty o dowolnej liczbie kolumn. Każdej z kolumn layoutów nadam kolorowe tła, żeby dokładnie było widać, co się dzieje na ekranie, ponieważ w tym rozdziale zajmujemy się właśnie strukturą strony.

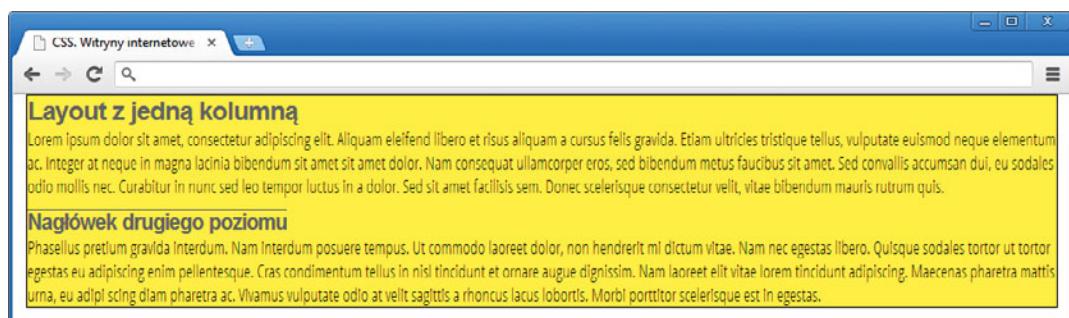
Zacznijmy od czegoś prostego — pojedynczej kolumny o stałej szerokości, wyśrodkowanej na stronie. Nadałem tekstowi kilka prostych stylów, żeby ulepszyć wygląd layoutu. W celu zaoszczędzenia miejsca nie podaję tutaj kodu CSS, którym się posłużyłem do obstylowania tekstu; nie ma on żadnego wpływu na technikę, którą tutaj demonstruję. W kodzie HTML utworzyłem kontener, który określa szerokość layoutu, a w nim kolejny kontener z zawartością kolumny.

```
<div id="wrapper">
    <article>
        <!-- elementy tekstowe -->
    </article>
</div>
```

Oto kod CSS:

```
#wrapper {width:960px; margin:0 auto; border:1px solid;}
article {background:#ffed53;}
```

Definiując szerokość elementu `wrapper` i nadając mu po bokach marginesy wartością `auto`, wyśrodkowałem layout w oknie (**rysunek 5.1**). Jego wysokość może się odpowiednio zwiększać wraz z dodawaniem nowej treści. Zawarty w nim element `article` zachowuje się jak typowy element blokowy o nieokreślonej szerokości, rozszerzając się w poziomie, aby wypełnić swój kontener. W drugiej kolumnie zamieszczę teraz element nawigacyjny.



RYSUNEK 5.1. Elementowi `article` nadałem kolorowe tło i obramowanie, aby wyraźnie ukazać, że jest to wyśrodkowana, pojedyncza kolumna

```
<div id="wrapper">  
  <nav>  
    <!-- nieuporządkowana lista -->  
  </nav>  
  <article>  
    <!-- jakiś tekst -->  
  </article>  
</div>
```

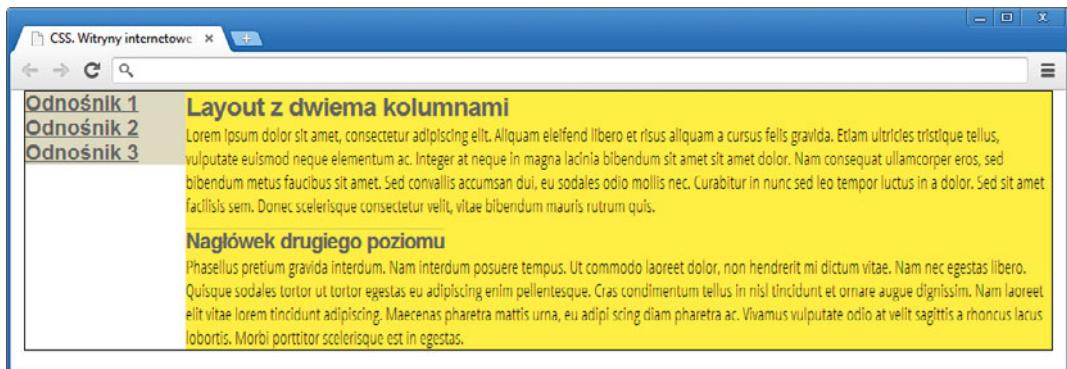
Muszę przekształcić obydwa kontenery w elementy pływające, aby umieścić je obok siebie. O tej metodzie wspomniałem już w rozdziale 4.

```
#wrapper {width:960px; margin:0 auto; border:1px solid;}  
nav {  
  width:150px;  
  float:left;  
}  
nav li {  
  list-style-type:none;  
}  
article {  
  width:810px;  
  float:left;  
  background:#ffed53;  
}
```



W punkcie „Udawane kolumny” w dalszej części rozdziału dowiesz się, jak uzyskać efekt kolumny o pełnej wysokości.

Jak widać na **rysunku 5.2**, nadając dwóm elementom szerokości łącznie równe szerokości kontenera ($150 + 810 = 960$) i przekształcając je w elementy pływające, sprawiasz, że zostają umieszczone obok siebie jako dwie kolumny. Wysokość obydwu z nich określona jest ilością zawartych w nich treści. Łatwo w ten sam sposób utworzyć trzecią kolumnę (albo i więcej kolumn).



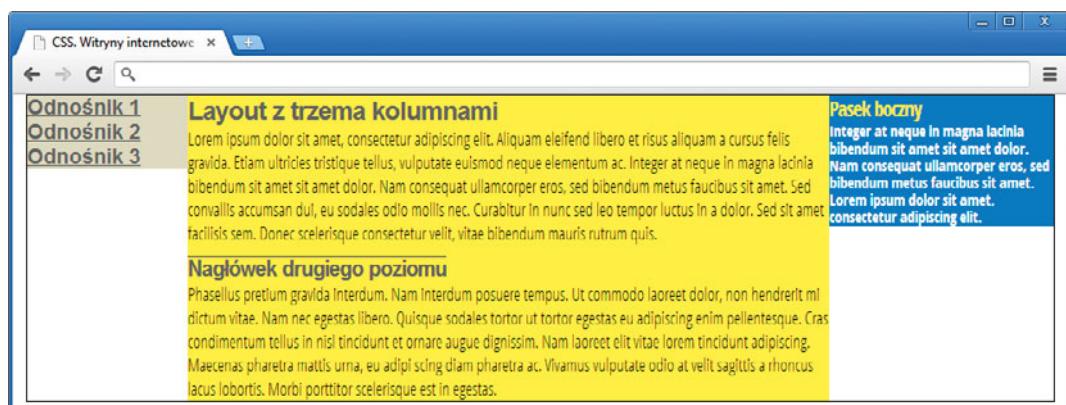
RYSUNEK 5.2. Dodałem drugą kolumnę

```
<div id="wrapper">
    <nav>
        <!-- nieuporządkowana lista -->
    </nav>
    <article>
        <!-- jakiś tekst -->
    </article>
    <aside>
        <!-- jakiś tekst -->
    </aside>
</div>
```

Zmienię szerokość kolumny `article` tak, by nie zabrakło miejsca na nową kolumnę.

```
#wrapper {width:960px; margin:0 auto; border:1px solid;}
nav {
    width:150px;
    float:left;
    background:#dcd9c0;
}
```

```
nav li {  
    list-style-type:none;  
}  
  
article {  
    width:600px;  
    float:left;  
    background:#ffed53;  
}  
  
aside {  
    width:210px;  
    float:left;  
    background:#3f7ccf;  
}
```



RYSUNEK 5.3. Layout składa się teraz z trzech płynących kolumn

Zmieniając trzy kontenery w elementy płynające i nadając im szerokości łącznie równe szerokości nadzawanego kontenera ($150 + 600 + 210 = 960$), otrzymuję szkielet layoutu trzykolumnowego (rysunek 5.3). W ten sposób mogę dodać tyle kolumn, ile chcę, o ile ich łączna szerokość jest równa szerokości ich kontenera. Wielokolumnowe layouty zwykle mają nagłówek i stopkę o pełnej szerokości, które teraz dodamy.

```
<div id="wrapper">
  <header>
    <! -- nagłówek tekstowy -->
  </header>
  <nav>
    <!-- nieuporządkowana lista -->
  </nav>
  <article>
    <! -- jakiś tekst -->
  </article>
  <aside>
    <! -- jakiś tekst -->
  </aside>
  <footer>
    <! -- jakiś tekst -->
  </footer>
</div>
```

Chcę, żeby nagłówek i stopka były szerokie na cały kontener, a — jak widać na **rysunku 5.4** — domyślnie takie właśnie są. Pozostaje mi jedynie nadać im kolorowe tła, żeby od razu było widać, gdzie się znajdują.

```
header {background:#f00;}
footer {background:#000;}
```



RYSUNEK 5.4. Nagłówek wygląda nieźle, ale stopka przenosiła się w góre pod pływające kolumny

Nagłówek jest szeroki na cały kontener, wysoki jak jego treść i wygląda dobrze, ale znajdująca się pod pływającymi elementami stopka przesuwa się tak wysoko, jak może. Rozwiązanie tego problemu jest proste ([rysunek 5.5](#)).

```
footer {clear:both;}
```



Rysunek 5.5. Po oczyszczeniu elementu stopka przenosi się pod najwyższą kolumnę

Nadając stopce właściwość `clear:both` (chociaż `clear:left` też by zadziałało, ponieważ mamy tu do czynienia tylko z elementami pływającymi w lewo), uniemożliwiamy jej przesunięcie się nad dolną krawędź pływających elementów. Ta prosta modyfikacja daje nam pewność, że stopka będzie się odtąd zawsze znajdować pod najdłuższą kolumną.

Oto kod CSS (poza stylami tekstu), który jak dotąd utworzyliśmy dla podanego wcześniej kodu HTML.

```
* {margin:0; padding:0;}  
#wrapper {width:960px; margin:0 auto; border:1px solid;}  
header {background:#f00;}  
nav {  
    width:150px;  
    float:left;  
    background:#dcd9c0;  
}  
nav li {  
    list-style-type:none;  
}  
article {  
    width:600px;  
    float:left;  
    background:#ffed53;  
}  
aside {  
    width:210px;  
    float:left;  
    background:#3f7ccf;  
}  
footer {clear:both; background:#000;}
```

Zajmiemy się teraz dwoma rzucającymi się w oczy problemami, jakie ma ten layout. Po pierwsze, treść styka się z bokami kolumn. Po drugie, kolumny mają wysokość zawartej w nich treści, a layout wyglądałby lepiej, gdyby miały one jednakową wysokość. Zaczniemy od utworzenia odrobiny przestrzeni wokół treści. Jak się zaraz przekonasz, to wcale nie jest takie proste.

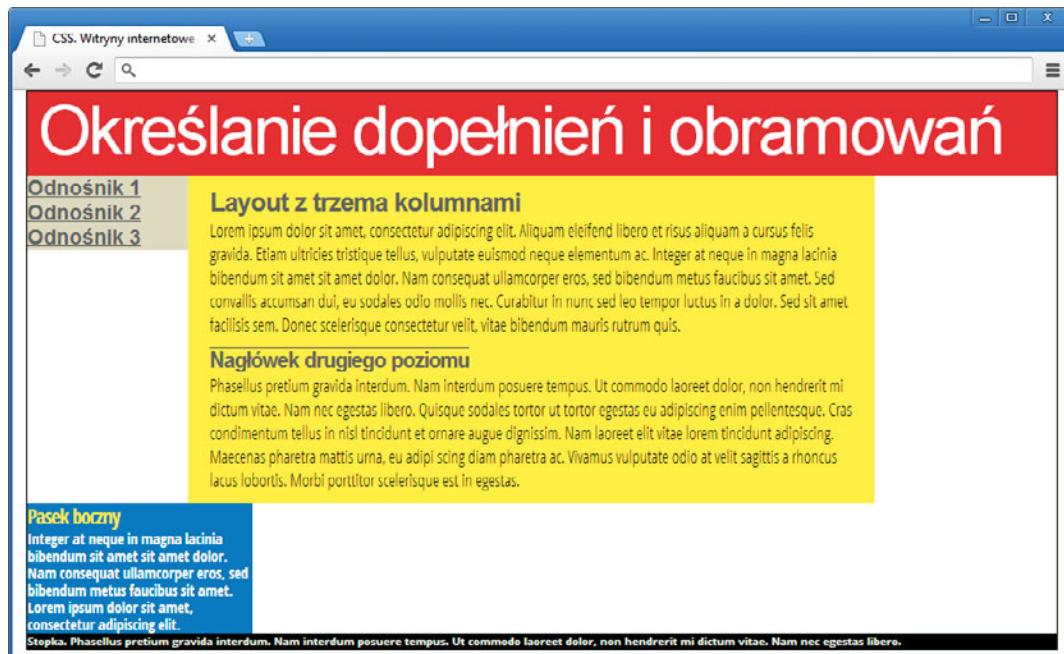
Nadawanie kolumnom dopełnień i obramowań

Podczas obrabiania treści zawartej w kolumnach layout może stać się szerszy od kontenera, a prawa kolumna przesunąć się pod lewą. Ten problem zwykle pojawia się z jednego z następujących powodów:

- Nadanie kolumnom marginesów i dopełnień w poziomie w celu odsunięcia ich treści od boków czy też nadanie marginesów samym kolumnom w celu odsunięcia ich od siebie (a zwykle powinieneś wprowadzić któryś z tych zmian na samym początku stylizowania layoutu) zwiększa szerokość layoutu i powoduje „poślizg” płynących elementów. Prawej płynącej kolumnie brak już miejsca, by siedzieć obok pozostałych, więc przesuwa się w dół, pod lewą kolumnę.
- Dodanie wielkich obrazów lub długich ciągów znaków bez spacji, takich jak adresy URL, może sprawić, że szerokość kolumny przekroczy szerokość layoutu. To też może spowodować przesunięcie prawej kolumny pod lewą.

Zobaczmy, co się stanie, kiedy dodam małe dopełnienie w celu odsunięcia treści od krawędzi którejś z tych kolumn. Zrobię to z kolumną środkową.

```
article {  
    width:600px;  
    float:left;  
    background:#ffed53;  
    padding:10px 20px;  
}
```



RYSUNEK 5.6. Zawartość środkowej kolumny nie dotyka już krawędzi kontenera, ale jej zwiększena szerokość sprawia, że prawa kolumna przenosi się w dół, pod lewą kolumnę

Zawartość środkowej kolumny, jak widać na **rysunku 5.6**, jest teraz ładnie oddzielona od krawędzi. Tym niemniej jej zwiększona szerokość sprawia, że prawej kolumnie brakuje miejsca, by znaleźć się w swoim docelowym położeniu, wobec czego przenosi się pod lewą kolumnę.

Jak pamiętasz z omówienia modelu polowego w rozdziale 3., dodanie elementowi o stałej szerokości poziomych marginesów, obramowań i dopełnień poszerza go. Zwiększenie w taki sposób szerokości pływających kolumn niemal zawsze wywołuje problem z „poślizgiem”, który widać na **rysunku 5.6**. Są jednak trzy sposoby, w jakie można temu zapobiec:

- Zmniejsz zadeklarowaną szerokość elementu o sumę szerokości poziomych marginesów, obramowań i dopełnień, które mu nadajesz.
- Nadaj dopełnienia i marginesy elementom zawartym w kontenerze zamiast samemu kontenerowi.
- Zmień sposób skalowania pól przy użyciu właściwości CSS3 `box-sizing: section {box-sizing:border-box;}`. Kiedy określasz właściwość `box-sizing` elementu `section`, jego szerokość pozostaje niezmieniona, gdy przypisujesz mu obramowania i dopełnienia; zamiast tego treść się ścieśnia.

Omówmy te trzy możliwości po kolei.

1. DOSTOSOWANIE SZEROKOŚCI DO OBRAMOWAŃ I DOPĘLENIEŃ

Nadajmy 600-pikselowej kolumnie dopełnienia o szerokości 20 pikseli z dwóch stron. Aby dostosować jej szerokość do dopełnienia, zmniejszamy ją o 40 pikseli do 560 pikseli. Prawa kolumna wraca teraz do swojego właściwego położenia. Zmianie szerokości elementów layoutu przy każdej zmianie marginesów i dopełnień bywa jednak dość mącejące. Choć sprawdza się to w praktyce, nie jest to najlepsze rozwiążanie. Przy dostosowywaniu dopełnień i obramowań bardzo łatwo przypadkiem zepsuć cały layout.

2. NADAWANIE DOPĘLENIEŃ I OBRAMOWAŃ ELEMENTOM ZAWARTYM W KONTENERZE

Nadawanie marginesów i dopełnień elementom z treścią *rzeczywiście się sprawdza*. Jeżeli nie mają one zadeklarowanej szerokości, ich treść zwyczajnie ścieśnia się wraz z dodawaniem do nich marginesów i dopełnień. Jak wynika z modelu polowego, element o nieokreślonej szerokości wypełnia swojego rodzica w poziomie, a jego treść ścieśnia się, kiedy nadaje się mu marginesy, obramowania i dopełnienia.

W kolumnie może się jednak znajdować wiele różnych rodzajów elementów z treścią. Jeśli zdecydujesz się później zmienić odległość treści od krawędzi kontenera, będziesz musiał ją dostosować dla wszystkich elementów, co jest czynnością mącejącą i podatną na pomyłki. Ponadto, jeżeli chcesz obstylować obramowanie kolumny, które także zwiększa jej szerokość, nie możesz tego dokonać poprzez obstylowanie poszczególnych elementów treści w jej obrębie.

Zamiast więc dodawać marginesy wszystkim elementom kontenera, najlepiej umieścić w kolumnie element `div` o nieokreślonej szerokości, który będzie zawierał wszystkie elementy treści, i jemu właśnie nadać obramowania i dopełnienia. Pozwoli Ci to przesunąć wszystkie elementy treści o jednakową odległość od krawędzi kolumny, określając jedno ustawienie tego wewnętrznego znacznika `div`, które będziesz mógł później w razie konieczności zmodyfikować. Szerokości wszelkich elementów z treścią, które w nim później umieścis, będą określone szerokością tego znacznika.

Kod prezentacyjny

HTML pełni z założenia funkcję semantyczną, tj. nadaje znaczenie treści, a CSS ma obejmować całość kodu odpowiedzialnego za prezentację treści — i słusznie. Tymczasem jedne znaczniki prezentacyjne są szkodliwe, ale inne nie mają zupełnie żadnego znaczenia. Tworzenie wielokolumnowych layoutów na podstawie tabel i oddzielanie akapitów znacznikami `
` zamiast poprawnego stosowania znaczników `p` to bardzo złe przyzwyczajenia, ponieważ taki kod wpływa na uniwersalność treści. Jeśli na przykład treść jest rozmieszczona w layoucie opartym na tabeli z trzema komórkami, będzie ona wyświetlana w ten sposób wszędzie, nawet na smartfonach, gdzie taki layout jest zupełnie nie na miejscu. Kiedy dodajesz kod prezentacyjny, który definiuje wygląd elementów w określony sposób i nie daje możliwości jego zmiany przy użyciu kodu CSS, albo kiedy elementy są wyświetlane w taki sposób, kiedy kod CSS nie jest dostępny, to nie używasz HTML poprawnie. Jednak używanie neutralnych elementów w rodzaju `div` i `span`, które nie mają domyślnych stylów i — co za tym idzie — konkretnego wyglądu, jeśli się go nie sprecyzujesz w CSS, jest, moim zdaniem, w pełni dopuszczalną metodą na uzyskanie konkretnego efektu prezentacyjnego.

Jedynym ewentualnym problemem, poza koniecznością pracowania na kilku dodatkowych elementach `div` w kodzie, jest konieczność zmierzenia się z krytyką purystów, według których przenigdy nie należy tworzyć kodu służącego wyłącznie celom prezentacyjnym. Moje przemyślenia na ten temat przeczytasz w ramce „Kod prezentacyjny”, a o kodzie, którym można się posłużyć, zamiast wstawiać wewnętrzne elementy `div`, w ramce „Selektor gwiazdkowy dzieci”. Zademonstruję Ci zastosowanie techniki z wewnętrznymi elementami `div` na przykładzie rozwiązania problemu zilustrowanego **rysunkiem 5.6**.

```
<article>
  <div class="inner">
    <!-- jakiś tekst -->
  </div>
</article>
```

Usunę teraz niefortunne dopełnienie z kolumny i żeby ukazać sprawność tej techniki, nowemu, wewnętrznemu elementowi `div` nadam nie tylko dopełnienie, ale także marginesy i obramowanie.

```
article {
  width:600px;
  float:left;
  padding:10px 20px;
  background:#ffed53;
}
```

```
article .inner {
    margin:10px;
    border:2px solid red;
    padding:20px;
}
```



RYSUNEK 5.7. Kiedy nadajesz marginesy, obramowanie i dopełnienie wewnętrznemu elementowi div o nieokreślonej szerokości, szerokość środkowej kolumny pozostaje niezmieniona, a prawa kolumna pozostaje na swoim miejscu

Jak widać na **rysunku 5.7**, w wyniku tego szerokość kolumny pozostała niezmieniona, a szerokość treści została zredukowana ołączną szerokość marginesów, obramowań i dopełnień dodanych do wewnętrznego znacznika `div`. Morał z tego taki, że po określaniu szerokości pływających kolumn nie należy ich już ruszać. Rozmieszczenie ich treści należy natomiast definiować przy użyciu wewnętrznego znacznika `div`.

Skoro już wszystko zademonstrowałem, usunę marginesy, dopełnienia i obramowania z kolumny środkowej. Zamiesczę terazewnętrzne elementy `div` w pozostałych dwóch kolumnach i zwyczajnie nadam kolumnom dopełnienia.

```
<div id="wrapper">
```

```
<header>
    <!-- nagłówek tekstowy -->
</header>
<nav>
    <div class="inner">
        <ul>
            <!-- odnośnik -->
        </ul>
    </div>
</nav>
<article>
    <div class="inner">
        <!-- treść tekstowa -->
    </div>
</article>
```

Selektor gwiazdkowy dzieci

Jest to zestawienie selektorów, które służy do określania marginesów wszystkich elementów kolumny bez konieczności wykorzystania wewnętrznych znaczników `div`.

Selektor gwiazdkowy wybiera wszystkie elementy, wobec czego selektor, po którym znajduje się gwiazdka, np. `tenSelektor *`, wybiera wszystkich potomków elementu oznaczonego selektorem `tenSelektor`. Selektor dziecka `>` odnosi się do dzieci elementów. Kiedy zatem stawiam selektor dziecka przed gwiazdką, np. `tenSelektor > *`, to wybieram wszystkie bezpośrednie dzieci danego elementu, ale nie jego dalszych potomków. Właśnie to chcę zrobić, aby odsunąć wszystkie bezpośrednie dzieci kontenera od jego krawędzi. W przypadku kolumny `section` muszę podać deklarację `section > * {margin:0 10px;}`. Ta deklaracja dodaje 10-pikselowe marginesy po lewej i prawej stronie wszystkich dzieci, ale nie innych potomków. Przy korzystaniu z selektora gwiazdkowego dzieci trzeba wziąć pod uwagę dwie rzeczy. Po pierwsze, przy definiowaniu pionowych odstępów między elementami dziećmi należy korzystać z właściwości `margin-top` i `margin-bottom` — nie możesz użyć właściwości zbiorczej `margin`, gdyż usuwa ona poziome marginesy nadane elementom przy użyciu selektora gwiazdkowego dzieci. Jeżeli chcesz bardziej wciąć treść któregoś dziecka, nadaj mu dopełnienie.

Po drugie, zastosowanie gwiazdkowego selektora dzieci może wpływać na wydajność strony, jako że do znalezienia pasujących elementów przeglądarka musi przemierzyć cały DOM. Sam stwierdzam, że to praktycznie nieistniejący problem. Jeżeli jednak Twoja strona jest olbrzymia i znajdują się na niej tysiące elementów, to możesz sprawdzić wpływ tego selektora na szybkość wczytywania strony przy użyciu ySlow lub innego narzędzia do pomiaru wydajności.

```

<aside>
  <div class="inner">
    <!-- treść tekstu -->
  </div>
</aside>
<footer>
  <!-- treść tekstu -->
</footer>
</div>

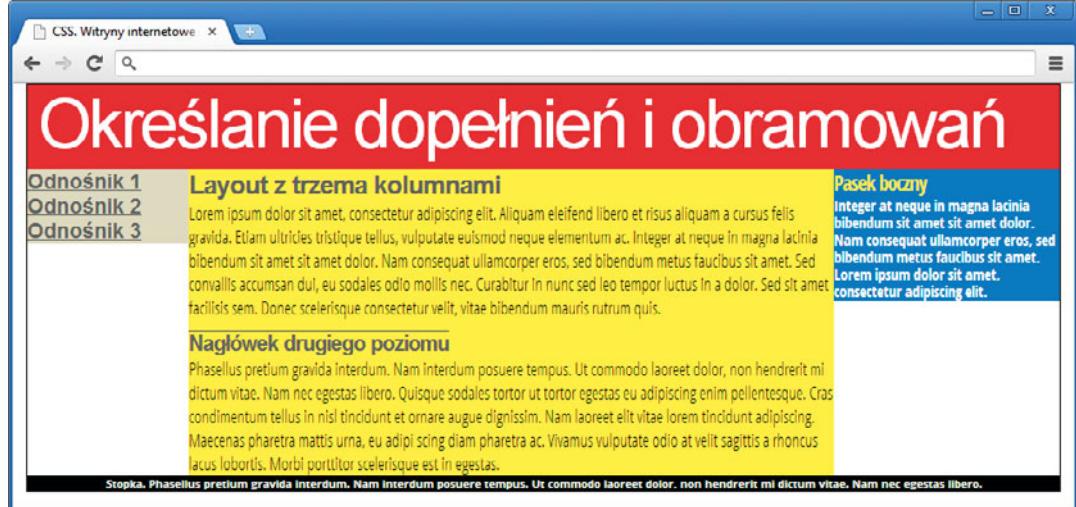
```

Użyję teraz elementów `div` do utworzenia przestrzeni wokół treści wszystkich trzech kolumn, a także wyśrodkuję tekst w elemencie `footer`.

```

nav .inner {padding:10px;}
article .inner {padding:10px 20px;}
aside .inner {padding:10px;}
footer {text-align:center}

```



RYSUNEK 5.8. Po nadaniu dopełnienia wewnętrznym elementom `div` szerokość layoutu nie ulega zmianie

Nadane trzem kolumnom dopełnienie tworzy odrobinę nieodzownej przestrzeni wokół tekstu, co widać na [rysunku 5.8](#). Dodatkowo wyśrodkowałem tekst w znaczniku [footer](#).

Te zmiany zdecydowanie ulepszyły wygląd layoutu względem tego, co było widać na [rysunku 5.5](#) — wystarczy kilka prostych czynności, aby layout uzyskał bardziej wysublimowaną oprawę. Kluczem do pracy z kolumnami i ich wewnętrznymi elementami [div](#) jest przekształcenie kolumn w elementy pływające i określenie ich szerokości, jednak bez określania szerokości któregokolwiek z zawartych w nich elementów treści. Te ostatnie powinny rozszerzać się tak, by wypełniać swoich rodziców (tj. elementy [div](#)); następnie możesz je odsunąć wraz z zawartością od krawędzi kolumn, nadając wewnętrznym elementom [div](#) marginesy lub dopełnienia.

Zauważ, że górne i dolne marginesy elementów [div](#) ulegają scaleniu, o ile górne i dolne obramowania kontenera nie są widoczne. Jeśli spotkasz się z tym problemem, zwyczajnie zdefiniuj pionowe dopełnienia samego kontenera. Musisz jednak uważać, by tym samym nie nadać mu poziomych dopełnień. Użyj kodu w rodzaju [article {padding: 20px 0;}](#), który określa jedynie dopełnienie górne i dolne.

3. ZASTOSOWANIE WŁAŚCIWOŚCI BOX-SIZING:BORDER-BOX

Trudno o prostszą technikę. Deklarację [box-sizing:border-box](#) dodaję do kodu CSS każdej z trzech pływających kolumn, aby móc następnie nadać ich polom dopełnienia i marginesy bez konieczności dostosowywania szerokości kolumn lub dodawania wewnętrznych znaczników [div](#). Kiedy dodam dopełnienia i marginesy, pole pozostanie tej samej wielkości, a jego zawartość zostanie ściśnięta. Oto schludny, prosty kod bez wewnętrznych znaczników [div](#).

```
<div id="wrapper">  
  <header>  
    <!-- nagłówek tekstowy -->  
  </header>  
  <nav>  
    <ul>  
      <!-- odnośnik -->  
    </ul>  
  </nav>
```

```
<article>
    <!-- treść tekstowa -->
</article>
<aside>
    <!-- treść tekstowa -->
</aside>
<footer>
    <!-- treść tekstowa -->
</footer>
</div>
```

A oto kod CSS:

```
* {margin:0; padding:0;}
#wrapper {width:960px; margin:0 auto; border:1px solid #000;
overflow:hidden;}
header {background:#f00;}
nav {box-sizing:border-box;
width:150px;
float:left;
background:#dcd9c0;
padding:10px 10px;
}
usuwa punktory -----| nav li {list-style-type:none;}
article {box-sizing:border-box;
width:600px;
float:left;
background:#ffed53;
padding:10px 20px;
}
```

```
aside {box-sizing:border-box;
width:210px;
float:left;
background:#3f7ccf;
padding:10px 10px;
}
footer {clear:both; background:#000;}
```



RYSUNEK 5.9. Właściwość `box-sizing` pozwala na bezpośrednie nadawanie kolumnom dopełnień



O kodach zastępczych i tym, dlaczego są potrzebne, przeczytasz w „*Dodatku*”.

Na **rysunku 5.9** widać, że po bezpośrednim nadaniu dopełnienia kolumnom treść zostaje ściśnięta, a layout nie przechodzi niepożądanych zmian. Jak to często bywa z prostymi rozwiązaniami, jest tu pewien haczyk: IE6 i IE7 nie rozumieją właściwości `box-sizing`. Istnieje jednak skrypt `borderBoxModel.js`, który możesz dodać do strony w komentarzu warunkowym (słującym temu, by ze skryptu korzystały jedynie IE6 i IE7) znajdującym się za całym kodem HTML, a przed zamkającym znacznikiem `body`, aby DOM mógł go wczytać przed wyświetleniem strony:

```
<body>
<!-- cała treść HTML-->
```

Ochrona przed przerośniętymi elementami

Projektowanie dynamicznych witryn, które inni ludzie będą później aktualizować, wymaga dodatkowego planowania. Wstawianie przerośniętych elementów jest jednym z problemów, na które trzeba się przygotować. Zamieszczenie w kolumnie obrazu przekraczającego jej szerokość może poszerzyć layout i sprawić, że pusta kolumna przesunie się do kolejnego rzędu. Prostym rozwiązaniem jest dodanie stylu `inner img {max-width:100%}`, aby ograniczyć szerokość obrazów do szerokości ich rodziców — w tym wypadku wewnętrznego elementu `div`.

Innym rozwiązaniem jest nadanie wartości `overflow:hidden` wszystkim kolumnom lub wewnętrznym elementom `div`, jeżeli z nich korzystasz. W wyniku tego obrazy (i jakiekolwiek inne przerośnięte elementy) nie będą jednak zmieniać wielkości, lecz będą przycinane na krawędzi kontenera.

Drugim problemem, który może wystąpić na dynamicznej witrynie, jest to, że HTML zawija wiersze jedynie w tych miejscach, gdzie znajdują się odstępy między wyrazami. Długie adresy URL, albo nawet — jeśli kolumny są dość wąskie — długie wyrazy, mogą poszerzyć kolumny. Z tego względu możesz rozważyć nadanie kontenerowi kolumn właściwości `word-wrap:break-word`, aby wszystkie kolumny wraz z zawartością dziedziczyły ją. Bądź świadom, że ustawienie `word-wrap` rozbija słowa zupełnie arbitralnie i nie wstawia łączników. Włącza się tylko wtedy, kiedy jest potrzebne, i może uchronić layout przed zmianami wywoływanymi wstawianiem długich adresów URL. Warto sprawdzić wytrzymałość layoutu poprzez wstawianie długich adresów URL, dużych obrazów i za dużych ilości tekstu do wszystkich kolumn, aby ujawnić miejsca narażone na zmiany.



Najnowszą wersję skryptu `border-BoxModel.js` oraz informacje o jego zastosowaniu i ograniczeniach znajdziesz na stronie <https://github.com/albertogasparin/borderBoxModel>. Artykuł twórcy skryptu na temat właściwości `box-sizing` przeczytasz na stronie <http://albertogasparin.it/articles/2012/02/start-using-css-box-sizing-today>.

```
<!-- wczytuje się tylko, jeśli IE jest w wersji starszej niż 8 -->
<!--[if lt IE 8 ]>
    <script src="helpers/borderBoxModel.js"></script>
<![endif]-->
</body>
```

Dzięki powyższemu rozwiążaniu IE6 i IE7 właściwie określają szerokość kolumn, zgodnie z zachowaniem `border-box`. Po długich latach rozmieszczania treści przy użyciu osadzonych elementów `div` zacząłem — jak dotąd z powodzeniem — eksperymentować z nadawaniem właściwości `box-sizing` nie samym płynącym kolumnom, ale wszystkim elementom przy użyciu następującego kodu:

```
* {box-sizing:border-box}
```

Mogę teraz pracować na bardziej logicznym modelu polowym, w którym szerokość określa niezmienną szerokość pola elementu, a nie szerokość jego zawartości. Idę tym samym śladami internetywych guru, takich jak Paul Irish, którego artykuł na ten temat przeczytasz na stronie <http://paulirish.com/2012/box-sizing-border-box-ftw>. Przeczytaj również komentarze, ponieważ poruszane są w nich ciekawe kwestie; jak to bywa z nowościami, nie wszystkim przypadają do gustu.

Przyjrzyjmy się teraz kilku metodom tworzenia płynnych layoutów.

Trzykolumnowe layouty z płynną środkową kolumną

Płynną środkową kolumnę można uzyskać na dwa sposoby: przy użyciu ujemnych marginesów do określania położenia prawej kolumny przy zmianie wielkości środkowej albo poprzez sprawienie w kodzie CSS3, by kontenery kolumn zachowywały się jak komórki tabeli. Metoda ujemnych marginesów dobrze się sprawdza na starych przeglądarkach, ale wykorzystanie właściwości CSS3 `table` jest prostsze. Zaprezentuję Ci obydwie te metody.

Trzykolumnowy layout z płynną środkową kolumną i ujemnymi marginesami

Problemem przy tworzeniu trójkolumnowego layoutu z płynnym obszarem środkowym jest zarządzanie prawą kolumną i określanie jej relacji względem całości layoutu podczas zmiany wielkości środkowej kolumny.

Opracowane przez webdewelopera Ryana Brilla rozwiązanie polega na regulowaniu marginesów dwóch kontenerów: pierwszego zawierającego wszystkie trzy kolumny i drugiego, w którym znajduje się lewa i środkowa kolumna. Kod przypomina ten, który służy do tworzenia layoutów o stałej szerokości, więc nie omówię tego przykładu krok po kroku. Zamiast tego podam cały kod z zaznaczonymi kontenerami, kilkoma zrzutami (rysunek 5.10a i b) oraz krótkim opisem.

```
<div id="main_wrapper">
    <header>
        <!-- zawartość nagłówka-->
    </header>
    <div id="threecolwrap">
        <div id="twocolwrap">
            <nav>
                <!-- elementy nawigacyjne -->
            </nav>
            <article>
                <!-- treść właściwa -->
            </article>
        </div>
    </div>
</div>
```

The diagram shows the structure of the layout with red arrows pointing from labels to specific parts of the code. Labels include:

- obejmuje trzy kolumny (points to the outermost div)
- obejmuje lewą i środkową kolumnę (points to the inner div containing nav and article)
- lewa kolumna (points to the first child of the inner div)
- środkowa kolumna (points to the second child of the inner div)

```
koniec kontenera twocolwrap ————— | </div>
prawa kolumna ————— | <aside>
                                         <!--zawartość paska bocznego--&gt;
                                     &lt;/aside&gt;
koniec kontenera threecolwrap ————— | &lt;/div&gt;
                                         &lt;footer&gt;
                                         <!-- zawartość stopki --&gt;
                                     &lt;/footer&gt;
                                 &lt;/div&gt;
A oto kod CSS:
* {margin:0; padding:0;}
body {font:1em helvetica, arial, sans-serif;}
div#main_wrapper{
    min-width:600px; max-width:1100px;
    margin:0 auto;
    background:url(images/bg_tile_150pxw.png) repeat-y #eee;
}
header {
    padding:5px 10px;
    background:#3f7ccf;
}
div#threecolwrap {
    float:left;
    width:100%;
    background:url(images/bg_tile_210pxw.png)
        top right repeat-y;
}
div#twocolwrap {</pre>

wyśrodkowuje layout,  
gdy rozmiar okna przekracza maksymalną szerokość



obraz tła jest domyślnie wyrównany do lewej



kontener musi pływać, by zatrzymać w sobie płynącą kolumnę



obraz tła wyrównany do prawej


```

kontener musi pływać, aby zatrzymać — w sobie płynącą kolumnę
przeciąga prawą kolumnę na
miejsce utworzone marginem — elementu z treścią właściwą

```
float:left;
width:100%;
margin-right:-210px;
}
```

```
nav {
    float:left;
    width:150px;
    background:#f00;
    padding:20px 0;
}
```



W kodzie do pobrania znajdują się kilka dodatkowych linijek kodu CSS z komentarzami na temat obsługi błędów w IE6 i IE7.

odpycha elementy dzieci od krawędzi kolumny

```
nav > * {margin:0 10px;}
```

```
article {
    width:auto;
    margin-left:150px;
    margin-right:210px;
    background:#eee;
    padding:20px 0;
}
```

dodaje przestrzeń po prawej stronie płynnej kolumny środkowej

```
margin-right:210px;
background:#eee;
padding:20px 0;
```

odpycha elementy dzieci od krawędzi kolumny

```
article > * {margin:0 20px;}
```

```
aside {
    float:left;
    width:210px;
    background:#fffed53;
    padding:20px 0;
}
```

odpycha elementy dzieci od krawędzi kolumny

```
aside > * {margin:0 10px;}
```

```
footer {
    clear:both;
    width:100%;
    text-align:center;
    background:#000;
}
```

Trzykolumnowy layout z płynną środkową kolumną

• Odnośnik nawigacyjny 1
• Odnośnik nawigacyjny 2

O layoucie
Ta strona jest stylizowana przy użyciu CSS. Wykorzystuję tu ujemne marginesy, aby uzyskać bogaty w funkcje layout.

Najważniejsze cechy
Zmień wymiary strony, żeby zauważać następujące zachowania.

1. Średkowa kolumna z treścią zmienia szerokość wraz ze zmianą szerokości przeglądarki.
2. Ustawienie min-width sprawia, że treść nie ścisnia się w nieskończoność, po dotarciu do minimalnej szerokości, layout już się nie ścisnia.
3. Ustawienie max-width sprawia, że treść nie staje się zbyt szeroka; po osiągnięciu maksymalnej szerokości, layout już się nie poszerza, a marginesy o wartości auto wyśrodkowują layout w oknie przeglądarki.
4. Stopka znajduje się pod najdłuższą kolumną.

Pasek boczny
Ta kolumna została rozrysowana w elemencie div o pełnej szerokości przy użyciu ujemnego marginesu. Dodatni margines elementu div z treścią zapewnia, że staczy dla niej miejsca.

Szablon CSS autor: CSS. Witryny internetowe sztyte na miarę, Charlesa Wyke-Smitha

RYSUNEK 5.10A i 5.10B. Kiedy przeciągam narożnik okna, środkowa kolumna zmienia szerokość, ale boczne kolumny pozostają niezmienione, co widać na rysunkach

Trzykolumnowy layout z płynną środkową kolumną

• Odnośnik nawigacyjny 1
• Odnośnik nawigacyjny 2

O layoucie
Ta strona jest stylizowana przy użyciu CSS. Wykorzystuję tu ujemne marginesy, aby uzyskać bogaty w funkcje layout.

Najważniejsze cechy
Zmień wymiary strony, żeby zauważać następujące zachowania:

1. Średkowa kolumna z treścią zmienia szerokość wraz ze zmianą szerokości przeglądarki.
2. Ustawienie min-width sprawia, że treść nie ścisnia się w nieskończoność, po dotarciu do minimalnej szerokości, layout już się nie ścisnia.
3. Ustawienie max-width sprawia, że treść nie staje się zbyt szeroka; po osiągnięciu maksymalnej szerokości, layout już się nie poszerza, a marginesy o wartości auto wyśrodkowują layout w oknie przeglądarki.
4. Stopka znajduje się pod najdłuższą kolumną.

Pasek boczny
Ta kolumna została rozrysowana w elemencie div o pełnej szerokości przy użyciu ujemnego marginesu. Dodatni margines elementu div z treścią zapewnia, że staczy dla niej miejsca.

Szablon CSS autor: CSS. Witryny internetowe sztyte na miarę, Charlesa Wyke-Smitha

Na **rysunkach 5.10a i 5.10b** widać płynny środek layoutu. Działa on następująco: prawa kolumna jest szeroka na 210 pikseli. Prawy, 210-pikselowy margines elementu `article` w środkowej kolumnie tworzy miejsce dla prawej kolumny, choć tym samym odpycha prawą kolumnę o 210 pikseli. Ujemny 210-pikselowy prawy margines kontenera obejmującego lewą i środkową kolumnę przyciąga zatem prawą kolumnę z powrotem na miejsce utworzone prawym marginesem `article`. Średkowa kolumna `article` o wielkości 100% wciąż zajmuje całą szerokość uzyskaną po umieszczeniu we właściwym miejscu płynącej lewej kolumny, ale jej prawy margines tworzy miejsce, na które można ściągnąć prawą kolumnę przy użyciu ujemnego marginesu wewnętrznego kontenera. Coś pięknego.

UDAWANE KOLUMNY

Jeżeli zastanawiasz się, jakim cudem kolumny mają teraz jednakową wysokość, to muszę przyznać, że wcale tak nie jest. Zastosowałem technikę udawanych kolumn, która ma wywoływać takie wrażenie. Polega ona na zamieszczeniu w kontenerach obrazów tła o takiej samej szerokości i barwie jak kolumny. Kontenery, w odróżnieniu od samych kolumn, zajmują całą wysokość obszaru treści, więc powielając grafikę tła w pionie, można graficznie rozciągnąć kolumny w dół. W tym przypadku użyłem osobnych obrazów dla lewej i prawej kolumny oraz koloru tła środkowej płynnej kolumny (**rysunek 5.11**).

RYSUNEK 5.11. Do płynnego layoutu trzeba użyć osobnych obrazów tła lewej i prawej kolumny



W przedstawionym powyżej kodzie CSS widać, że do podania obrazu tła lewej kolumny użyłem znacznika `div main_wrapper`, a znacznika `div threecolwrap` do podania obrazu tła prawej kolumny, który umieszczał przy prawej krawędzi tego elementu. Nadałem też tłu elementu `main_wrapper` kolor środkowej kolumny. Wypełnia ona tło całego układu, ale grafika bocznych kolumn oraz nagłówka i stopka znajdują się nad elementem `main_wrapper` (dzieci zasłaniają swoich rodziców), więc widać go jedynie w środkowym obszarze.

Kolejną cechą tego layoutu jest to, że elementom z treścią nadałem poziome marginesy przy użyciu selektora gwiazdkowego dzieci, tj. `nav > * {margin:0 10px;}`, zamiast nadawać dopełnienia wewnętrznym elementom `div`, żebyś mógł zobaczyć zastosowanie tej techniki w praktyce. Przeczytasz o niej więcej w ramce „Selektor gwiazdkowy dzieci”, która znajduje się we wcześniejszej części rozdziału.

Trzykolumnowy layout z płynną środkową kolumną, oparty na właściwościach CSS3 table

O ile tworzenie wielokolumnowych layoutów przy użyciu tabel HTML jest niepożądane, to wykorzystanie CSS, by layout zachowywał się jak tabela, jest w pełni dopuszczalne. Zastosowanie tej techniki nie skutkuje utworzeniem wyglądu tabelowego, którego obstylowania nie można by zmienić np. na potrzeby wyświetlenia treści w jednej kolumnie na urządzeniu przenośnym. Tabele CSS w istocie zachowują się w ramach tworzenia layoutów bardzo sprawnie, do czego zaraz wróć.

Tabele, w swojej podstawowej formie, składają się z trzech elementów: kontenera `<table>`, zawierającego kontenery rzędów `<tr>`, zawierające komórki `<td>`.

```
<table>
  <tr> <td>Komórka 1</td><td>Komórka 2</td><td>Komórka 3</td>
  </tr>
  <tr> <td>Komórka 1</td><td>Komórka 2</td><td>Komórka 3</td>
  </tr>
</table>
```

Właściwość CSS `display` elementu HTML można zmienić na `table`, `table-row` lub `table-cell`, by element symulował zachowanie swojego tabelowego odpowiednika. Zalety oznaczania kolumn CSS jako komórek tabeli są następujące:

- Komórki tabeli rozmieszczone są obok siebie jak kolumny bez potrzeby tworzenia pływających elementów, wobec czego można im bezpośrednio nadawać dopełnienia, nie psując przy tym layoutu.
- Wszystkie komórki w jednym rzędzie tabeli są domyślnie jednakowej wysokości, więc nie trzeba tworzyć udawanych kolumn.
- Wszystkie kolumny o nieokreślonej szerokości są płynne.

Te trzy cechy rozwiązuje problemy z layoutami, z którymi zapoznałeś się w tym rozdziale. Tym niemniej (bo przecież musi być jakiś haczyk) tabele CSS3 nie są obsługiwane przez IE7 i wcześniejsze wersje, a poza tym nie ma żadnego sprawnego kodu zastępczego. Jeżeli jednak jesteś na to (wraz ze swoimi zleceniodawcami) gotów, możesz zaniechać obsługi IE7. Tabele CSS są prostym, solidnym i pełnym rozwiązaniem, które polecam najbardziej.

Zauważ, że nie potrzebujesz nawet kontenerów `div`, które miałyby się zachowywać jak elementy `table` i `tr` — wystarczy zmienić właściwość `display` trzech kolumn na `table-cell`. Kiedy brakuje znacznika rzędu, silniki przeglądarek automatycznie tworzą taki wokół ciągłego zestawienia komórek, a także znacznik tabeli wokół znaczników rzędów, jeśli takiego nie ma. Nie potrzebujesz zatem dodatkowego kodu — nadaj właściwości `display` kolumn wartość `table-cell`, a przeglądarka załatwi resztę.

Do uzyskania layoutu trzykolumnowego z płynnym środkiem wystarczy zatem kod HTML

```
<nav><!-- treść --></nav>
<article><!-- treść --></article>
<aside><!-- treść --></aside>
```

i CSS:

```
nav {display:table-cell; width:150px; padding:10px;
background:#dcd9c0;}
article {display:table-cell; padding:10px 20px;
background:#ffed53;}
aside {display:table-cell; width:210px; padding:10px;
background:#3f7ccf;}
```

Skorzystałem tutaj z layoutu o stałej szerokości z **rysunku 5.5**, jednak bez wewnętrznych znaczników `div`. Jak widać w kodzie CSS, właściwość `float:left` wszystkich kolumn zamiениłem na `display:table-cell` oraz usunąłem ustawienie `width` kolumny środkowej. Oto powstał layout o płynnym środku z kolumnami o pełnej wysokości, którym możesz do woli nadawać dopełnienia i obramowania (**rysunek 5.12**).

Zauważ, że tego prostego i funkcjonalnego layoutu nie można uzyskać w IE7 i IE6 przy pomocy skryptu; nie ma nawet zastępczego kodu, który działałby odpowiednio. W tych przeglądarkach utworzone tą techniką kolumny rozmieszczone są jedna nad drugą, więc jeżeli nie zamierzasz zrezygnować z obsługi starszych wersji IE, musisz korzystać z innych metod tworzenia layoutów, które tutaj opisuję, przynajmniej do czasu, aż te przeglądarki wyjdą z użycia.



RYSUNEK 5.12. W tym płynnym layoucie wykorzystałem właściwość CSS3 `display:table-cell`, aby kolumny zachowywały się jak rząd komórek tabeli

Podsumowując, płynący layout o stałej szerokości z osadzonymi znacznikami `div` jest najbezpieczniejszym rozwiązaniem, jako że jest kompatybilny z nowymi i starymi przeglądarkami, ale jego utworzenie wymaga największego nakładu pracy. Zastosowanie właściwości `box-sizing:border-box` (wraz ze skryptem zastępczym na potrzeby IE7 i IE6) pozwala na uzyskanie bardziej intuicyjnego modelu polowego oraz nie wymaga korzystania z osadzonych elementów `div`. Wreszcie, zastosowanie właściwości `display:table-cell` to rozwiązanie łatwe do wdrożenia i oferujące szeroką funkcjonalność (tj. możesz utworzyć płynny layout z kolumnami o pełnej wysokości bez użycia jakichkolwiek osadzonych znaczników `div`), ale wykorzystanie go przekłada się na porzucenie obsługi przeglądarek IE6 i IE7.

Przedstawione do tej pory layouty składają się z nagłówka, stopki i znajdujących się między nimi trzech kolumn. Rozdział zakończy bardziej złożonym przykładem, w którym wykorzystam omówione już techniki tworzenia wielokolumnowych layoutów.

Layout wielorzędowy i wielokolumnowy

W ostatnim podrozdziale, w ramach przygotowania do stylizacji ukończonej strony internetowej w rozdziale 7., chcę pomóc Ci rozwinąć umiejętności na bazie technik, które przedstawiłem

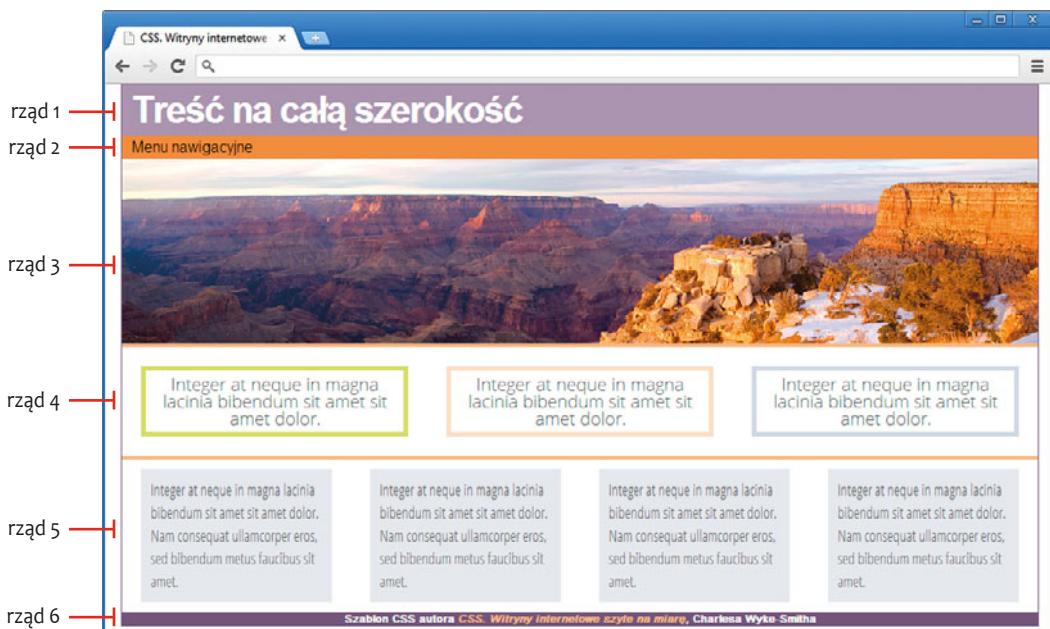
w omówieniach poprzednich layoutów trzykolumnowych. Pokażę Ci bardziej złożony i życiowy szkielet strony; layout, który można wykorzystać jako podstawę szablonu WordPress lub strony firmowej.

W przedstawionych dotąd layoutach znaczniki `article`, `nav` i tym podobne występowały pojedynczo, wobec czego wybieranie ich było proste — wystarczyło podać nazwę. Na bardziej złożonych stronach takie znaczniki pojawiają się wielokrotnie, wobec czego konieczne jest korzystanie z selektorów kontekstowych do ich odróżnienia.

Wykorzystam ten przykład do zademonstrowania poprawnego sposobu dodawania do kodu możliwie najmniejszej liczby klas i identyfikatorów, które pozwalają na dokładne wskazanie dowolnego elementu.

Jak pokazałem we wcześniejszej części tego rozdziału, wykorzystanie osadzonych elementów `div` do tworzenia dopełnienia wokół treści można teraz zastąpić zastosowaniem różnych właściwości CSS3 w rodzaju `box-sizing:border-box`. Tymczasem znaczniki wewnętrzne, czy to `div` czy inne, można także wykorzystywać na różne sposoby w ramach stylizacji, co również zamierzam tutaj ukazać.

Nasz layout, jak widać na **rysunku 5.13**, składa się z sześciu rzędów rozciągniętych na całą szerokość; widać, że w czwartym rzędzie znajdują się trzy kolumny, a w piątym cztery.



RYSUNEK 5.13. Layout z sześcioma rzędami zajmującymi całą szerokość strony oraz dwoma osobnymi zestawami kolumn

Oto kod HTML:

```
<div id="wrapper">
  <header>
    <h1>Treść na całą szerokość</h1>
  </header>
  <nav>
    <p>Menu nawigacyjne</p>
  </nav>
  <section id="branding">
    
  </section><!-- koniec obszaru branding -->
  <section id="feature_area">
    <article>
      <div class="inner">
        <p>Lorem Ipsum</p>
      </div>
    </article>
    <!-- jeszcze dwa elementy article -->
  </section><!-- koniec obszaru feature_area -->
  <section id="promo_area">
    <article>
      <div class="inner">
        <p>Lorem Ipsum</p>
      </div>
    </article>
    <!-- jeszcze trzy elementy article -->
  </section><!--koniec obszaru promo_area -->
  <footer>
    <p>Szablon CSS autora książki <a href="http://www.helion.pl/ksiazki/csswi3.htm"><em>CSS. Witryny internetowe szyte na miarę</em></a>, Charlesa Wyke-Smitha</p>
  </footer>
</div>
```

Widziałeś już większość technik, które wykorzystano do obstylowania kodu w celu uzyskania przedstawionego powyżej layoutu: tworzenie elementów „rzędów”, które domyślnie rozciągają się na całą szerokość kontenera, wprowadzenie płynących elementów składających się na kolumny oraz uzyskiwanie poziomych dopełnień przy użyciu osadzonych znaczników `div` zamiast kontenerów, aby zapobiec problemowi „przesiągania się” kolumn przy poszerzaniu layoutu. Zamiast zapełniać kolejne strony kodem CSS, który już widziałeś, skoncentrujmy się na kilku konkretnych fragmentach kodu. Ja zaś pokażę Ci kilka sposobów na tworzenie selektorów elementów w kodzie HTML, który obejmuje choćby i kilkaset znaczników.



Pełny kod CSS dla tego przykładu znajdziesz na stronie <http://www.helion.pl/ksiazki/csswiz.htm>.



Przegląd różnych selektorów CSS znajdziesz w rozdziale 2.

Praktyczne selektory CSS

Wraz z rozbudową layoutu określone znaczniki HTML (`section`, `article`, `nav` itp.) zaczynają się wielokrotnie pojawiać w kodzie. W powyższym kodzie na przykład znajduje się siedem znaczników `article`. Aby wybrać konkretnie ten, który chcesz, musisz znaleźć sposób na odróżnienie od siebie elementów o jednakowych nazwach. Błędnym rozwiązaniem byłoby przypisanie każdego z nich do innej klasy, co często robią ci, którzy dopiero zaczynają pracę z CSS. Jest to niewłaściwe zastosowanie klas, które służą do oznaczania elementów o wspólnych cechach, a w dodatku prowadzi do zagmatwania kodu i zmniejszenia jego czytelności, ponieważ ciągle trzeba spoglądać jednym okiem na kod HTML, by zorientować się, gdzie co jest.

Dużo lepszym pomysłem jest nadanie identyfikatorów znaczników znajdujących się na najwyższym poziomie poszczególnych sekcji kodu (co jest przykładem prawidłowego zastosowania identyfikatorów, które mają wskazywać unikalne elementy na stronie). Takie identyfikatory służą do wskazywania różnych fragmentów kodu HTML w ramach selektorów kontekstowych określających ścieżkę do potomków, których chcesz wybrać. Takie podejście pozwala na zredukowanie liczby klas i identyfikatorów w kodzie, a używane w ten sposób selektory kontekstowe wskazują jasną ścieżkę do elementów, do których się odnoszą, dzięki czemu kod CSS jest bardziej czytelny.

Przyjrzyj się przedstawionemu wyżej kodowi HTML. Zamieszczone w nim trzy identyfikatory to jedyne punkty zahaczenia, których potrzebuję, żeby móc dokładnie wskazać jakikolwiek element.

Niewykluczone, że nie potrzebowałbym żadnych innych — ewentualnie jeszcze kilku — do oznaczenia elementów treści, które bym jeszcze dodał. Pokażę Ci teraz tę koncepcję w działaniu na przykładzie obstylowania rzędu czwartego z trzema elementami o kolorowych obramowaniach.

Najwyższym w hierarchii elementem tego fragmentu jest `section` o identyfikatorze `id="feature_area"`, który jest kontenerem rzędu z trzema elementami `article` wyświetlonymi jako kolumny. Przyjrzyjmy się teraz kodowi CSS tego rzędu.

nadaje styl czterem kolumnom
article

```
section#feature_area article {  
    float:left;  
    width:320px;  
    padding:10px 0;  
    background:#fff;  
    border-top:4px solid #f7be84;  
}
```

kontenerom kolumn można nadać
jedynie pionowe dopełnienie

```
section#feature_area article .inner {  
    margin:10px 20px;  
    padding:5px;  
    background:#fff;  
    border:5px solid;  
}  
  
section#feature_area article:nth-child(1) .inner {  
    border-color:#d7dd6f;  
}  
  
section#feature_area article:nth-child(2) .inner {  
    border-color:#f6dec5;  
}  
  
section#feature_area article:nth-child(3) .inner {  
    border-color:#d1d8e4;  
}
```

określa style poszczególnych
pół treści

Selektorem `section#feature_area article` wybrałem wszystkie trzy pola, a następnie podałem regułę określającą ich wspólny styl szerokości, dopełnienia i grubości obramowań. Mam jedną deklarację, którą określам wygląd wszystkich trzech pól.

Następnie podaję trzy reguły określające różnice między poszczególnymi polami. Oto reguła, która odnosi się do środkowego wewnętrznego znacznika `div`:

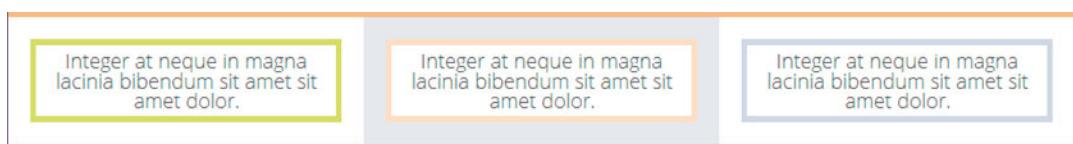
```
section#feature_area article:nth-child(2) .inner {border:5px solid #f6dec5;}
```

Selektor, gdyby przeczytać go od końca, twierdzi: znajdź element o klasie `inner`, który jest drugim dzieckiem typu `article` elementu `section` o identyfikatorze `feature_area`; nadaj mu jasnopomarańczowe obramowanie.

Jak widać, wystarczy mi jeden punkt zahaczenia w postaci identyfikatora elementu na najwyższym poziomie hierarchii, aby wybrać dowolnego jego potomka (a nawet elementy treści, które w nim później osadzę). Równie istotne jest to, że mogę dzięki temu uniknąćomyłkowego obstylowania innej części kodu. Nadając elementom klasy i identyfikatory tylko wtedy, kiedy nie ma innych możliwości wskazywania znaczników, tworzysz czysty i uporządkowany kod HTML, a także czytelniejszy i łatwiejszy w obsłudze kod CSS.

Wewnętrzne elementy `div` w działaniu

Ten layout daje nam również sposobność do przyjrzenia się możliwościom określania położenia i stylu elementów, jakie dają nam wewnętrzne znaczniki `div`. W tym layoucie wewnętrzne znaczniki `div` nie tylko pozwalają na dodawanie dopełnień w poziomie bez psucia layoutu, ale pełnią także ważną funkcję graficzną, jako że każdy z nich tworzy kolorową obwódkę wokół pól z treścią. Ich rodzice `article` nie są widoczni, ale tymczasowo pokoloruję środkowy, żebyś mógł zobaczyć ich położenie (**rysunek 5.14**).



RYSUNEK 5.14. Środkowy element `article` zaznaczony jest kolorem szarym

Zaznaczenie kolorem środkowego elementu `article` pokazuje, że owe elementy wypełniają kontener rzędu `section` i stykają się ze sobą.

Zwróć uwagę na to, jak konkretnie obstylowałem przestrzeń otaczającą wewnętrzne znaczniki `div`. Przestrzeń poziomą tworzy się przy użyciu lewych i prawych marginesów, które odpychają znaczniki `div` od krawędzi ich rodzica `article` (któremu nie mógłbym nadać dopełnienia w poziomie bez zepsucia layoutu). Przestrzeń w poziomie utworzyłem, nadając dopełnienie rodzicowi, ponieważ, jak widziałeś wcześniej, wszelkie pionowe marginesy dzieci znikają, gdy rodzic nie ma rozrysowanych górnych i dolnych krawędzi.

Tym samym kończę omówienie tego layoutu i mam nadzieję, że dzięki temu lepiej rozumiesz, jak korzystać z selektorów kontekstowych w kodzie. Kolejną wersję tego layoutu wystylizuję obszerniej w rozdziale 7.

Podsumowanie

W tym rozdziale dowiedziałeś się, że wielokolumnowe layouty można utworzyć przy użyciu płynących elementów o określonej szerokości oraz poprzez wykorzystanie zachowania tabel CSS. Zobaczyłeś, jak tworzyć stałe, wyśrodkowane na stronie layouty oraz layouty płynne o minimalnej i maksymalnej szerokości. Nauczyłeś się używania wewnętrznych znaczników `div` do tworzenia przestrzeni w płynących layoutach bez zmieniania ich ogólnej szerokości, a także tworzenia udawanych kolumn. Nauczyłeś się także ograniczania liczebności identyfikatorów i klas w kodzie, co polega na uznawaniu ich za kontekstowe punkty zahaczenia, a nie za oznaczenia elementów. W rozdziale 7. zobaczysz, jak omówione tutaj rodzaje layoutów tworzą podstawę pełnoprawnej witryny, kiedy połączę je z komponentami interfejsu, takimi jak menu, formularze i chmurki, o których przeczytasz w kolejnym rozdziale.

ROZDZIAŁ 6

Komponenty interfejsu

KOMPONENTAMI INTERFEJSU NAZYWAM standardowe elementy interfejsu użytkownika (UI), które występują w HTML, takie jak listy i formularze. Do innych powszechnie stosowanych komponentów UI należą rozwijane menu i chmurki. O ile w HTML nie istnieją konkretne elementy będące ich odpowiednikami, to pokażę Ci, jak je stworzyć i stylizować.

Pokażę Ci też, jak pisać gotowy kod takich komponentów, żebyś nie musiał ich za każdym razem tworzyć od podstaw, a mógł je zamieścić w bibliotece arkuszy stylów, które będziesz mógł dostosowywać w zależności od wymogów danego projektu.

Każda strona potrzebuje systemu nawigacji. Na początek przyjrzymy się temu, jak stylizować listy HTML w celu uzyskania atrakcyjnego i funkcjonalnego menu nawigacyjnego.

Tworzenie menu nawigacyjnych

Menu jest zbiorem odnośników. Odnośniki są pogrupowane logicznie w liście HTML, dzięki czemu wyświetlane są prawidłowo, nawet jeśli nie podano żadnego kodu CSS, który określałby ich styl. Ponieważ elementy listy są elementami blokowymi, domyślnie są one rozmieszczone jeden nad drugim.

Pionowe menu

Zacznijmy od bardzo prostego przykładu, abyś zapoznał się z najważniejszymi technikami stylizowania list jako menu.

```
<nav class="list1">  
  <ul>  
    <li><a href="#">Muzyka alternatywna</a></li>
```

```

<ul>
  <li><a href="#">Country</a></li>
  <li><a href="#">Jazz</a></li>
  <li><a href="#">Rock</a></li>
</ul>
</nav>

```

W powyższym kodzie zawarłem tekst każdego punktu listy w odnośniku, aby można je było kliknąć. (Każdy odnośnik ma zastępczy adres URL #, w którego miejscu będzie można później podać właściwy adres).

Zacznę omówienie pierwszego przykładu od wskazania pewnych problemów, z jakimi możesz się zetknąć przy pierwszych próbach stylizowania odnośników nawigacyjnych, oraz sposobów na zaradzenie im. Oto kod, który tworzy czysty, prosty styl (**rysunek 6.1**).

usuwa domyślne dopełnienia i marginesy	* {margin:0; padding:0;}
określa położenie menu na stronie	nav {margin:50px; width:150px;}
pole wokół menu	.list1 ul {border:1px solid #f00; border-radius:3px; padding:5px 10px 3px;}
usuwa punktory menu i dodaje miejsca wokół odnośników	.list1 li {list-style-type:none; padding:3px 10px;}
selektor „niepierwotnego”	.list1 li + li {border-top:1px solid #f00;}
stylizuje odnośniki	.list1 a {text-decoration:none; font:20px Exo, helvetica, arial, sans-serif; font-weight:400; color:#000; background:#ffed53;}
podświetlenie odnośnika po najechaniu kursem	.list1 a:hover {color:#069;}

RYSUNEK 6.1. Oto wystylizowana, nieuporządkowana lista. Jak można wnioskować z wyglądu kurSORA, te obszary elementów listy, w których nie ma tekstu, nie są klikalne



Podany tu styl jest dość prosty (**rysunek 6.1**). Element `nav`, który pojawił się w HTML5, jest semantycznie adekwatnym kontenerem, któremu mogę nadać klasę w celu uzyskania kontekstu przy stylizacji.

WYKORZYSTANIE SELEKTORA „NIEPIERWODNEGO”

Powyższy kod jest ciekawy pod dwoma względami. Po pierwsze, wyróżniony selektor `li + li` odnosi się do „każdego `li` znajdującego się po innym elemencie `li`”.

W każdej nieprzerwanej sekwencji elementów, takiej jak przedstawiona tutaj, taki selektor wybiera wszystkie elementy poza pierwszym. Ten wybór pozwala mi nadać górnego obramowanie każdemu elementowi poza pierwszym, dzięki czemu mogę utworzyć oddzielające je linie. Gdybym zdefiniował górnego obramowanie w deklaracji `li`, na ekranie pojawiłaby się także linia nad pierwszym elementem, co wcale mi nie odpowiada. Selektor „niepierworodnego” jest bardzo przydatny w pracy z listami. Ten sam rezultat można by jednak uzyskać przy użyciu kodu:

definiuje górnego obramowania każdego elementu `li`
usuwa obramowanie pierwszego elementu `li` w grupie

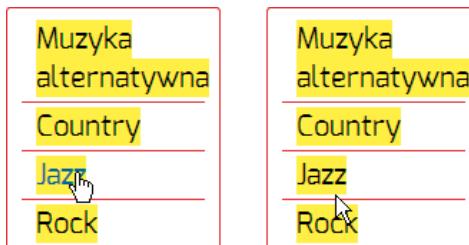
— `li {border-top:1px solid #f00;}`
— `li:first-child {border-top:none;}`

POPRAWNE TWORZENIE KLIKALNYCH LIST

Na rysunkach 6.2a i b tła odnośników są pokolorowane, aby można było zobaczyć ich klikalne obszary. Jak widzisz, klikalny jest sam tekst, ponieważ odnośniki są elementami liniowymi, których pola scisłe obejmują ich zawartość. Z punktu widzenia doświadczenia użytkownika byłoby lepiej, gdyby klikalne były całe obszary między liniami. Pokazuję tu ten problem, ponieważ na wielu stronach działania odnośników pozostaje niezmienione, wobec czego nic się nie dzieje, gdy użytkownik kliką obszar odnośnika, ale nie sam jego tekst.

Ponadto, każdy z elementów `li` ma trzy piksele dopełnienia górnego i dolnego, które tworzą odstępy między odnośnikami a liniami. Sprawia to jednak, że kursor zmienia postać między palcem a strzałką, kiedy trafia między kolejnymi odnośnikami na drobne fragmenty elementu `li`.

RYSUNEK 6.2A i B. Nadałem odnośnikom tymczasowe żółte tło, aby ukazać ograniczoną wielkość klikalnych obszarów. Kursor zmienia postać, kiedy przebywa drobne odstępy między odnośnikami



Załatwmy obydwa te problemy, przenosząc dopełnienia i linie z elementów `li` na odnośniki i sprawiając, by odnośniki całkowicie wypełniały elementy listy.

```
* {margin:0; padding:0;}  
nav {margin:50px; width:150px;}
```

```
.list2 ul {border:1px solid #f00; border-radius:3px;
padding:5px 10px 3px;}
.list2 li {list-style-type:none; padding:3px 10px;}
.list2 li + li a {border-top:1px solid #f00;}
.list2 a {display:block; padding:3px 10px; text-decoration:
none; font:20px Exo, helvetica, arial, sansserif;
font-weight:400; color:#000; background:#ffed53;}
.list2 a:hover {color:#069;}
```

Zmieniając selektor `li + li` na `li + li a`, nadaję teraz górne obramowania *odnośnikom zawartym* w elementach listy, które znajdują się za elementami listy (mam nadzieję, że nadążasz). Przeniosłem także dopełnienie z elementów listy na same odnośniki. Odnośniki stykają się teraz ze sobą w pionie i nie ma między nimi przestrzeni, wobec czego kurSOR już nie zmienia stanu, kiedy przesuwa się między odnośnikami. Wreszcie, nadając właściwości `display` każdego odnośnika wartość `block`, sprawiam, że pole odnośnika nie przylega ścisłe do jego zawartości, lecz rozszerza się, by wypełnić swojego rodzica, zwiększać tym samym klikalny obszar. Te usprawnienia widać wyraźnie na **rysunkach 6.3a i b.**

RYSUNEK 6.3A i B. Każdy odnośnik wypełnia teraz cały obszar swojego elementu listy, który w całości staje się klikalny. Na rysunku 6.3a obszar ten jest tymczasowo oznaczony na żółto. Na rysunku 6.3b widać ukonczone menu bez testowego żółtego tła



Menu poziome

Elementy list domyślnie rozmieszczone są w pionie, ale łatwo je rozmieścić obok siebie, tworząc poziome menu. Robi się to poprzez zmianę elementów listy w elementy pływające. Oto prosta lista:

```
<nav class="list1">
<ul>
<li><a href="#">Koszule</a></li>
<li><a href="#">Spodnie</a></li>
<li><a href="#">Sukienki</a></li>
<li><a href="#">Buty</a></li>
```

```
<li><a href="#">Dodatki</a></li>
</ul>
</nav>
```

oraz kod CSS, który jest bardzo podobny do kodu CSS pionowych list:

```
.list1 ul {
    overflow:hidden;
}
.list1 li {
    float:left;
    list-style-type:none;
}
.list1 a {
    display:block;
    padding:0 16px;
    text-decoration:none;
    color:#999;
}
.list1 li + li a {border-left:1px solid #aaa;}
.list1 a:hover {color:#555;}
```

sprawia, że ul obejmuje pływające elementy li

układa listę w poziomie

usuwa punktory

sprawia, że odnośniki wypełniają elementy li

usuwa podkreślenie odnośnika

wszystkie odnośniki poza pierwszym otrzymują obramowanie po lewej

RYSUNEK 6.4. Prosta stylizacja poziomego menu

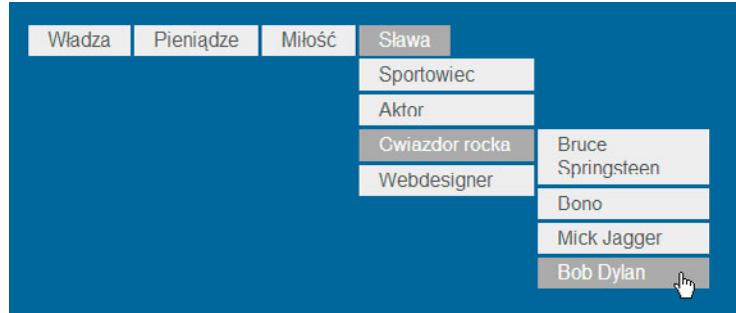


Na **rysunku 6.4** widać nowoczesny styl, który często pojawia się na witrynach z odzieżą i sklepach internetowych. Rzeczy, na które należy zwrócić uwagę: zmiana elementów `li` w elementy pływające w celu utworzenia poziomej zamiast pionowej listy oraz deklaracja `display:block` w stylu odnośników, która sprawia, że wypełniają one elementy `li`, przez co cały ich obszar staje się klikalny. Podobnie jak na **rysunku 6.1**, selektor `li + li a` pozwala na dodanie pionowych linii oddzielających po lewej stronie każdego odnośnika z wyjątkiem pierwszego. Rozwińmy nieco ten styl.

Rozwijane menu

Na żaden inny temat nie dostaję tylu e-maili, co w związku z rozwijanymi menu. Wobec tego omówię je tutaj szczegółowo. Przyjrzymy się, czym one są, jak się je tworzy, oraz rozważmy warianty, które mogą się przydać. Rozwijane menu, które oparte jest na zbiorze osadzonych list, łączy w sobie techniki stylizacji w poziomie i w pionie, z którymi się właśnie zapoznałeś (**rysunek 6.5**).

RYSUNEK 6.5. Trójpoziomowe rozwijane menu



Do obstylowania tego menu nadam kontenerowi listy, który zazwyczaj bywa elementem `nav`, klasę `multi_drop_menu`. Wszystkie style w kodzie CSS tego menu zaczyna się od `.multi_drop_menu`, aby nadano je tylko tym listom, które zawarte są w kontenerze należącym do tej klasy.

Poniższy kod oparty jest na kodzie z poprzednich przykładów tworzenia menu. Na pierwszy rzut oka może się wydawać rozbudowana, ale w rzeczywistości mamy do czynienia z trzema osadzonymi listami.

```
<nav class="multi_drop_menu">
    <ul>
        <li><a href="#">Władza</a></li>
        <li><a href="#">Pieniądze</a></li>
        <li><a href="#">Miłość</a></li>
        <li><a href="#">Sława</a>
            <ul>
```

poziom pierwszy




```
        <li><a href="#">Sportowiec</a>
        <li><a href="#">Aktor</a>
        <li><a href="#">Cwiazdor rocka</a>
        <li><a href="#">Webdesigner</a>
        <li><a href="#">Bruce Springsteen</a>
        <li><a href="#">Dono</a>
        <li><a href="#">Mick Jagger</a>
        <li><a href="#">Bob Dylan</a>
```

poziom drugi




```

<li><a href="#">Sportowiec</a></li>
<li><a href="#">Aktor</a></li>
<li><a href="#">Gwiazdor rocka</a>

poziom trzeci ━━━━ <ul>
<li><a href="#">Bruce Springsteen</a></li>
<li><a href="#">Bono</a></li>
<li><a href="#">Mick Jagger</a></li>
<li><a href="#">Bob Dylan</a></li>

koniec poziomu trzeciego ━━━━ </ul>
</li>
<li><a href="#">Webdesigner</a></li>

koniec poziomu drugiego ━━━━ </ul>
</li>
</ul>

koniec poziomu pierwszego ━━━━ </ul>
</nav>

```

W tym kodzie zamieściłem jedynie te opcje menu, które widać na **rysunku 6.5**, ale sam możesz utworzyć podmenu dowolnego elementu, zamieszczając nieuporządkowaną listę po odnośniku do danej pozycji, ale przed zamkającym znacznikiem **li**. Jak zobaczyłeś, rozbudowywanie opcji menu w ten sposób nie wymaga wprowadzania zmian w CSS. Zaczniemy tworzenie tego menu od najwyższego poziomu, menu poziomego.

NAJWYŻSZY POZIOM MENU

Oto kod CSS najwyższego poziomu menu.

```

początek kodu ze stylami graficznymi ━━━━ .multi_drop_menu {font:1em helvetica, arial, sans-serif;}
sprawia, że odnośnik wypełnia element li ━━━━ .multi_drop_menu a {
kolor tekstu ━━━━ display:block;
kolor tła ━━━━ color:#555;
dopełnienie wokół tekstu odnośnika ━━━━ background-color:#eee;
                           padding:.2em 1em;

```

```
 szerokość linii oddzielającej ━━━━ border-width:3px;  
 kolor linii oddzielającej — może ━━━━ border-color:transparent;  
 być kolorowa lub przezroczysta  
 }  
  
.multi_drop_menu a:hover {  
 kolor tekstu po najechaniu  
 kursorem ━━━━ color:#fff;  
 kolor tła po najechaniu kursorem ━━━━ background-color:#aaa;  
 }  
  
 podświetlenie tła podczas  
 kliknięcia ━━━━ .multi_drop_menu a:active {  
 background:#fff;  
 kolor podświetlenia tekstu  
 podczas kliknięcia ━━━━ color:#ccc;  
 }  
 początek kodu ze stylami  
 określającymi funkcjonalność ━━━━ .multi_drop_menu * {margin:0; padding:0;}  
 sprawia, że elementy ul obejmują ━━━━ .multi_drop_menu ul {float:left;}  
 pływające elementy li  
 .multi_drop_menu li {  
 sprawia, że menu rozłożone  
 jest poziomo ━━━━ float:left;  
 usuwa punktory ━━━━ list-style-type:none;  
 kontekst pozycjonowania podmenu ━━━━ position:relative;  
 }  
  
.multi_drop_menu li a {  
 sprawia, że odnośnik  
 wypełnia element li ━━━━ display:block;  
 nadaje odnośnikom  
 prawe obramowanie ━━━━ border-right-style:solid;  
 sprawia, że tło widnieje jedynie  
 pod dopełnieniem, a nie pod  
 obramowaniem ━━━━ background-clip:padding-box;  
 usuwa podkreślenia odnośników ━━━━ text-decoration:none;  
 }  
.multi_drop_menu li:last-child a {border-right-style:none;}  
 tymczasowo ukrywa niższe  
 poziomy menu ━━━━ .multi_drop_menu li ul {display:none;}
```

RYSUNEK 6.6. Najwyższy poziom menu, tym razem z podświetleniem :active przy kliknięciu



 Kiedy oddzielasz w ten sposób style graficzne od funkcjonalnych, koniecznie dodaj komentarz na temat tego, co robisz.

W kodzie CSS **rysunku 6.6** w oczy rzuca się przede wszystkim oddzielenie stylów graficznych menu (takich jak wielkość fonta oraz kolory obramowań i tekstu) od stylów odpowiedzialnych za „mechanikę funkcjonowania”, która określa układ i zachowanie menu (co opisuję w komentarzach). Warto w ten sposób porządkować złożone komponenty, zwłaszcza jeśli inni członkowie zespołu produkcyjnego mają później modyfikować Twój kod — dzięki temu będą mogli dostosować deklaracje stylów bez naruszania kodu CSS odpowiedzialnego za funkcjonalność.

Jak widzisz, jedyną istotną zmianą stylu elementów `li` jest przekształcenie ich w elementy pływające, aby utworzyć poziome menu zamiast pionowego, oraz przekształcenie rodzica `ul` w element pływający, by elementy `li` były w nim osadzone. Zauważ, że w tym przypadku do osadzenia pływających elementów nie używam właściwości `overflow:hidden`, gdyż dodane później rozwijane menu nie byłyby widoczne — znajdują się poza obszarem kontenera swojego rodzica `ul`, byłyby ukryte.

 Więcej na temat zastosowania właściwości `background-clip` z przezroczystymi obramowaniami przeczytasz na stronie <http://css-tricks.com/transparent-borders-with-background-clip>.

W celu uzyskania najlepszej funkcjonalności stylizacja graficzna — dopełnienia, tła, obramowania itp. — nadana jest odnośnikom `a`, a nie elementom `ul` i `li`, aby możliwie powiększyć kikalne obszary i zapewnić ciągłość reakcji na kurSOR, o której wspomniałem w poprzednim przykładzie. Aby uzyskać ten efekt, ale jednocześnie oddzielić odnośniki graficznie, użyłem deklaracji `background-clip:box-padding`, która sprawia, że kolorowe tła odnośników nie rozciągają się pod obramowaniem, tak jak to standardowo robią. Teraz mogę sprawić, by obramowania były przezroczyste (lub kolorowe), tym samym tworząc odstępy między odnośnikami, w których prześwitują obszary strony. Unikam dzięki temu korzystania z marginesów do oddzielania odnośników. Taka stylizacja pozwala na uzyskanie płynnych przejść między kolejnymi podświetlanymi elementami bez zmiany wyglądu kurSORA. Są one graficznie oddzielone, ale w rzeczywistości sąsiadują ze sobą.

Być może zauważysz, że elementom `li` nadałem właściwość `position:relative`, która przyda się przy dodawaniu podmenu, ale nie jest jeszcze wymagana na tym etapie. W ostatniej linijce kodu CSS ukryłem podmenu — wyświetlanie ich podczas pracy nad menu najwyższego poziomu byłoby za bardzo rozpraszające. W następnych krokach pokażę Ci, jak włączyć ich widoczność przy najeżdżaniu kursorem na menu.

Style nadane menu najwyższego poziomu zostaną odziedziczone przez jego kolejne poziomy, więc większość pracy nad ich stylizacją została już wykonana.

STYLIZACJA ROZWIJANEGO MENU

Choć dziedziczenie stylów menu najwyższego poziomu przez kolejne podmenu jest wygodne, niektóre z tych reguł nie są potrzebne. Na przykład rozwijane menu, które zawiera drugopoziomową listę, ma być rozłożone w pionie, więc nie są mi potrzebne pływające elementy [1](#), które w menu najwyższego poziomu rozłożone są w poziomie. Kolejną różnicą jest to, że w poziomym menu linie oddzielające znajdująły się po prawej stronie elementów, podczas gdy w pionowym menu mają znajdować się nad elementami. Dodatkowo, określ tutaj położenie rozwijanego menu bezwzględnie, aby móc je umieścić bezpośrednio pod jego rodzicem, którego położenie określiłem wcześniej względnie.

Jednocześnie włączę widoczność rozwijanego menu drugiego poziomu, żeby widzieć, nad czym pracuję, ale menu trzeciego poziomu, którym zajmę się później, pozostawię ukryte.

Oto dodatkowy kod CSS, którego potrzeba do wprowadzenia tych zmian:

szerokość menu drugiego poziomu	—→	.multi_drop_menu li ul {width:9em;}
usuwa odziedziczone prawe obramowania	—→	.multi_drop_menu li li a {border-right-style:none;
dodaje obramowania górne	—→	border-top-style:solid;
		}
/* zmiany funkcjonalności */		
tymczasowo włącza widoczność menu drugiego poziomu	—→	.multi_drop_menu li ul {display:block;
określa położenie względem rodzica	—→	position:absolute;
wyrównuje lewą krawędź podmenu do rodzica	—→	left:0;
wyrównuje górną krawędź rozwijanego menu do dolnej krawędzi rodzica	—→	top:100%;
		}
.multi_drop_menu li li {		

usuwa odziedziczone płynawanie — `float:none;`
 — tworzy pionową listę
`}`

ukrywa trzeci poziom — `.multi_drop_menu li li ul { display:none; }`

RYSUNEK 6.7. Obstylowane i zamieszczone pod rodzicem rozwijane menu



Kluczowe dla powodzenia tego etapu jest określenie względnego i bezwzględnego położenia rozwijanego menu. Określając jego górne położenie jako 100% (wobec górnej krawędzi jego wzgólnie rozmieszczonego rodzica `li`), sprawiam, że jego górną krawędź znajduje się bezpośrednio na dolnej krawędzi rodzica; odstęp graficzny pomiędzy rodzicem a rozwijanym menu jest w rzeczywistości obramowaniem pierwszego odnośnika rozwijanego menu (**rysunek 6.7**).

FUNKCJONALNOŚĆ ROZWIJANEGO MENU

Teraz zaczyna się prawdziwa zabawa, czyli puszczenie rozwijanego menu w ruch. Muszę ukryć rozwijaną listę i sprawić, by stawała się widoczna po najechaniu kursem na jej rodzica.

ukrywa poziom drugi — `.multi_drop_menu li ul { display:none; }`
 określa położenie względem rodzica — `position:absolute;`
 wyrownuje lewą krawędź podmenu do rodzica — `left:0;`
 wyrownuje górną krawędź podmenu do dolnej krawędzi rodzica — `top:100%;`
`}`

wyswietla menu, gdy kurSOR znajduje się nad rodzicem — `.multi_drop_menu li:hover > ul { display:block; }`

RYSUNEK 6.8. Po najechaniu kursorem na odnośnik na ekranie pojawia się menu będące jego dzieckiem



ukrywa poziom drugi

Nasze menu będzie działać poprawnie tylko, jeśli je najpierw ukryjemy

`li ul {display:none;}`

i sprawimy, by pojawiało się znowu po najechaniu kursorem na rodzica.

wyswietla poziom drugi

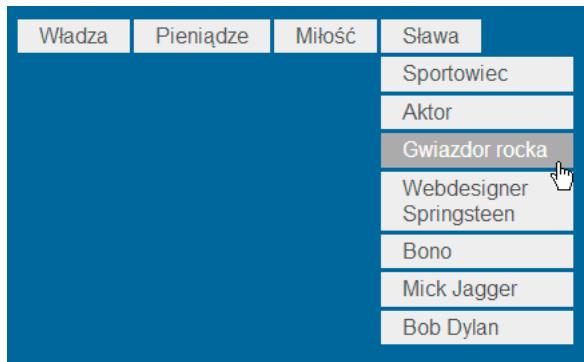
`li:hover > ul {display:block;}`

Ostatni wiersz zwyczajnie stwierdza, żeby po najechaniu kursem na pozycję listy wyświetlana była lista będąca jej dzieckiem. Zauważ, że pseudoklasę `:hover` nadałem elementowi `li`, zawierającemu element odnośnika `a`, a nie samemu odnośnikowi. Zrobiłem tak, ponieważ element, który chcę wyświetlić, czyli osadzony `ul`, nie jest dzieckiem odnośnika, tylko elementu `li` — określone zachowanie musi być zatem wyzwalane po najechaniu kursem na ten element. Ponadto zamieściłem selektor dziecka `>` pomiędzy selektorem pływającego elementu listy a selektorem listy będącej dzieckiem `ul`, tak aby wyświetlana była jedynie lista będąca bezpośrednim dzieckiem (**rysunek 6.8**). Bez selektora dziecka po najechaniu kursem na pozycję w pierwszym menu wyświetlane byłoby zarówno menu drugiego, jak i trzeciego poziomu.

UTWORZENIE TRZECIEGO POZIOMU MENU

Zajmijmy się teraz działaniem trzeciego poziomu menu. W zasadzie on już w pewien sposób działa. Gdybyś otworzył w przeglądarce kod, na którym oparty jest **rysunek 6.8**, i najechał kursem na podmenu, zobaczyłbyś to, co widać na **rysunku 6.9**.

RYSUNEK 6.9. Lista trzeciego poziomu wyświetla się po najechaniu kursem na jej rodzica, ale nie zajmuje właściwego położenia



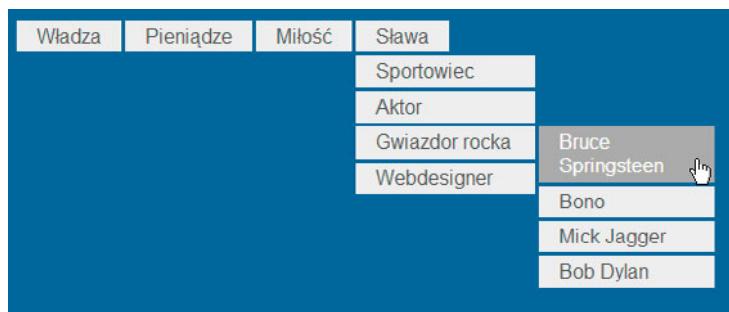
Po najechaniu kursorem na rodzica na ekranie wyświetlna jest lista trzeciego poziomu, jako że podana wcześniej reguła CSS z pseudo-klasą `:hover` odnosi się w równym stopniu do tego elementu z trzeciego poziomu, jak do elementu z drugiego poziomu. Obecnie jednak menu trzeciego poziomu widoczne na **rysunku 6.9** znajduje się za swoim rodzicem. Jak można się spodziewać, jego bieżąca relacja z jego rodzicem jest taka sama, jak relacja między drugim a pierwszym poziomem menu — znajduje się pod elementem, na który najechano kursem. Na rysunku widać, że *Bruce Springsteen* zakryty jest ostatnim elementem menu, *Webdesigner*.

Muszę teraz przenieść menu trzciopoziomowe tak, żeby znajdowało się po prawej stronie rozwiniętego menu, a jego górna krawędź była wyrównana do górnej krawędzi wybranej pozycji.

```
.multi_drop_menu li li ul {
    position: absolute;
    left: 100%;
    top: 0;
}
```

położenie względem rodzica
wyrównuje menu do prawej krawędzi rodzica
wyrównuje menu z górną krawędzią aktywnego elementu

RYSUNEK 6.10. Menu trzeciego poziomu zajmuje teraz prawidłowe miejsce obok menu drugiego poziomu



Menu działa teraz poprawnie (**rysunek 6.10**). Na najwyższym poziomie listy możesz zamieścić więcej elementów, a na niższych poziomach dodatkowe podmenu — wszystkie będą działać bez konieczności wprowadzania zmian w kodzie CSS. Chcę wprowadzić jeszcze dwie zmiany w tym menu. Po pierwsze, aby uzyskać przydatny kod do wielokrotnego użycia, dodaję kilka zamiennych stylów, aby najwyższy poziom menu można było uporządkować w pionie zamiast w poziomie, np. tworząc nawigacyjny pasek boczny. Zacznę od nadania kontenerowi `nav` drugiej klasy, `vertical`.

zwrócić uwagę na spację między nazwami klas w kodzie HTML

```
<nav class="multi_drop_menu vertical">
```

Mogę teraz utworzyć kilka stylów CSS, które będą nadawane tylko w przypadku obecności klasy `vertical`. Zauważ, że każda z tych dodatkowych reguł menu klasy `vertical` zaczyna się od selektora

brak odstępu między nazwami klas w kodzie CSS → `.multi_drop_menu.vertical`

który odnosi się do elementów należących do *obydwu* podanych klas. Zauważ, że w kodzie CSS nie ma spacji między ich nazwami.

szerokość pionowego menu z najwyższego poziomu → `.multi_drop_menu.vertical {width:8em;}`

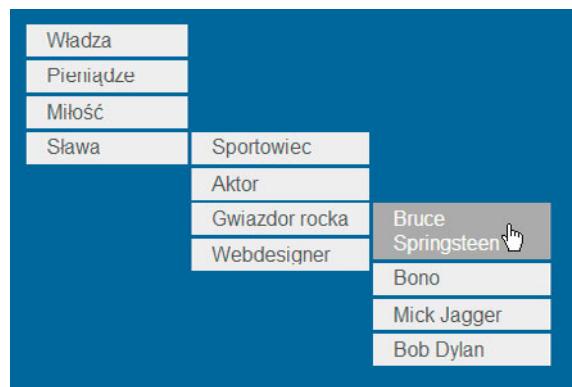
```
.multi_drop_menu.vertical li a {
    border-right-style:none;
    border-top-style:solid;
}
```

poziom trzeci → `.multi_drop_menu.vertical li li a {border-left-style:solid;}`
`.multi_drop_menu.vertical ul,`
`.multi_drop_menu.vertical li {`

sprawia, że menu najwyższego poziomu rozłożone jest pionowo → `float:none;`
`}`

wyrównuje menu poziomu drugiego do prawej krawędzi menu poziomu pierwszego → `left:100%;`
`top:0;`

RYSUNEK 6.11. Po zdefiniowaniu dodatkowych stylów elementy menu najwyższego poziomu rozmieszczone są w pionie



Sprawiłem, że elementy `li` z najwyższego poziomu przestały płynąć, aby menu powróciło do swojego domyślnego stanu, gdzie elementy znajdują się jeden pod drugim. Nie płyną teraz także rodzic `ul`, który wcześniej miał obejmować płynące elementy `li`.

Określiłem także szerokość kontenera `nav`. Bez tego zarówno element `nav`, jak i zawarte w nim menu pierwszego poziomu rozszerzałyby się, by zająć możliwie największą przestrzeń. Wreszcie, zmieniłem relację między menu drugiego poziomu a pierwszego poziomu, aby była taka sama, jak między menu trzeciego a drugiego poziomu, tj. aby znajdowało się po jego prawej stronie, a jego górna krawędź była wyrównana z górną krawędzią jego rodzica w liście. Zauważ, że usunąłem również prawe obramowania elementów z menu pierwszego poziomu i zastąpiłem je obramowaniem górnym (rysunek 6.11).

PODŚWIETLENIE ŚCIEŻKI WYBORU

Na rysunku 6.11 widać, że podświetlony jest tylko element, nad którym znajduje się kursor. Aby wskazać użytkownikowi, że podjął właściwy wybór, muszę podświetlić wybór dokonany na każdym kolejnym poziomie menu. Łatwo to osiągnąć, zastępując kod

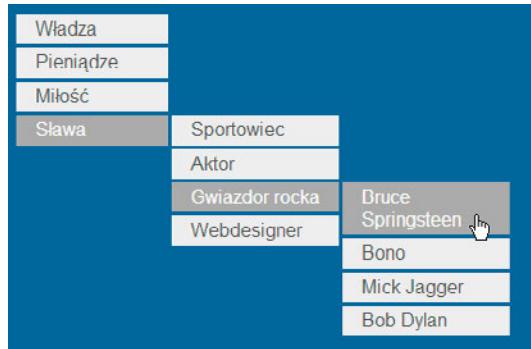
`.multi_drop_menu a:hover` następującą deklaracją:

```
.multi_drop_menu li:hover > a {  
    color:#fff;  
    background-color:#aaa  
}
```

kolor tekstu po najechaniu kursem →

kolor tła po najechaniu kursem →

RYSUNEK 6.12. Podświetlone elementy menu, które składają się na ścieżkę wyboru do elementu, nad którym znajduje się kursor



Ta sprytna sztuczka działa, ponieważ zdarzenie `:hover` „bąbelkuje” w górę hierarchii. Przypisując to zdarzenie elementowi `li`, nadaję je także wszystkim elementom `li`, które są przodkami. Następnie nadaję style `:hover` każdemu z ich bezpośrednich dzieci, które są odnośnikami. Ukończone menu jest dzięki temu wygodniejsze w użyciu ([rysunek 6.12](#)).

Podsumowując, łącząc kod CSS odpowiedzialny za menu ze stroną i nadając klasę `multi_drop_menu` kontenerowi zawierającemu nie-uporządkowaną listę, przekształcam listę w menu, czego rozwinięcie zobaczysz w kolejnym rozdziale.

Formularze



Bardzo polecam książkę Luke'a Wroblewskiego o projektowaniu formularzy: http://www.lukew.com/resources/web_form_design.asp.



Jeżeli chcesz się dowiedzieć więcej o walidacji danych formularzy po stronie serwera, możesz sięgnąć po moją książkę Codin' for the Web, w której omawiam PHP i MySQL.

Formularze pełnią wyjątkową funkcję, jako że *przekazują informacje od użytkownika do serwera*, podczas gdy pozostałe elementy strony wyświetlają treści *przesypane z serwera do użytkownika*. Bez wątpienia zależy nam na tym, by formularze działały sprawnie; chcemy przecież uzyskiwać od użytkowników informacje zwrotne, ich opinie, dane kontaktowe i – rzecz jasna – numery kart kredytowych. Duże formularze bywają skomplikowane, a transakcja z wykorzystaniem koszyka zakupów w sklepie internetowym jest przykładem tego, że bez dobrze zaprojektowanej obsługi procesu zakupów sfrustrowany użytkownik może po prostu zrezygnować z usługi. Jakość designu ma w takiej sytuacji jasno określona wartość wymierną.

Samo przetwarzanie danych formularza jest zagadnieniem wykraczającym poza tematykę tej książki, ale pokażę Ci, jak stylizuje się formularze z myślą o przejrzystości i łatwości w użyciu. Na początek rzućmy okiem na gotowy formularz.

Elementy HTML formularza

Oto gotowy formularz, który stworzę w tym podrozdziale ([rysunek 6.13](#)).

Formularze w HTML5

W HTML5 pojawiło się trzynaście nowych wariantów znacznika `input`, a także szereg usprawnień, w tym atrybut `placeholder`, który wyświetla tekst ze wskazówką (np. „Podaj identyfikator klienta”) bezpośrednio w polu tekstowym i znika, kiedy użytkownik zaczyna pisać.

Więcej o znacznikach i atrybutach formularzy HTML5 oraz ich obsłudze w przeglądarkach przeczytasz na stronach <http://www.wufoo.com/html5> i <http://www.html5rocks.com/en/tutorials/forms/html5forms>.

RYSUNEK 6.13. Wystylizowany formularz, w którym znajdują się typowe elementy form

legend
label
input
type=text
label
input
type=password
label
textarea
label
kontrolka input
type=date
legend
h4
input
type=checkbox (każda kratka ma znacznik label)
h4
input
type=radio (każda kratka ma znacznik label)
label
select
type=submit

Wystylizowany formularz

Sekcja 1 • Podstawowe kontrolki

* wymagane pole

Nazwa użytkownika *

Podaj nazwę użytkownika

Hasło *

Hasło musi mieć minimum 8 znaków

Opis

Wprowadź tutaj opis.

Data

rrrr-mm-dd

Sekcja 2 • Kontrolki wielokrotnego wyboru

Zaznacz dowolną liczbę kratek

Opcja 1

Opcja 2 jest wstępnie zaznaczona

Opcja 3 — dodaj ich, ile chcesz

Musisz wybrać przynajmniej jedną

Wybierz tylko jedną opcję

Opcja 1 jest wstępnie zaznaczona, a jej tekst jest zawijany, jeśli rozciąga się na kilka wierszy.

Opcja 2

Opcja 3

Wybierz

Brak

Zatwierdź

fieldset
fieldset

wymagany znacznik form

```
<form class="styling_form1" action="process_form.php"
      method="post">
<h3>Wystylizowany formularz</h3>
```

Zacznę od przedstawienia całego kodu formularza, a następnie szczegółowo omówię poszczególne elementy HTML.

kontener zbioru kontrolek	—————	<fieldset>
nazwa sekcji formularza	—————	<legend>Sekcja 1 • Podstawowe kontrolki</legend>
jednowierszowe pole tekstowe	—————	<section> <p class="note">* wymagane pole</p> <label for="user_name">Login *</label> <input type="text" id="user_name" name="user_name" /> <p>Podaj nazwę użytkownika</p>
ma taką samą wartość, jak jej identyfikator	—————	</section>
wartość text atrybutu sprawia, że ta kontrolka wyświetlana jest jako pole tekstowe	—————	<section>
początek pola na hasło	—————	<label for="password">Hasło *</label> <input type="password" id="password" name="password" maxlength="20" /> <p>Hasło musi mieć minimum 8 znaków</p>
znaki wpisywane w polu hasła wyświetlane są jako punktory	—————	</section>
wielowierszowe pole tekstowe	—————	<section> <label for="description">Opis</label> <textarea id="description" name="description" placeholder="Wprowadź tutaj opis. "></textarea>
pole daty HTML	—————	</section> <section> <label for="special_date">Data</label> <input type="date" id="special_date" name="event_date" min="2012-09-05" />
		</section>
		</fieldset>
		<fieldset>
		<legend>Sekcja 2 • Kontrolki wielokrotnego wyboru</legend>
		<section>
sekcja z kratkami	—————	



Więcej o elementach

`input` przeczytasz na
stronach <http://htmlhelp.com/reference/html40/forms/input.html> i <http://www.javascript-coder.com/html-form/html-form-tutorial-p1.phtml>.

sekcja z kratkami

```
<h4>Zaznacz dowolną liczbę kratek</h4>
<section>
    <input type="checkbox" id="check1"
        name="checkboxset" value="1" />
    <label for="check1">Opcja 1</label>
</section>
<section>
    <input type="checkbox" checked="checked"
        id="check2"
        name="checkboxset" value="2" />
    <label for="check2">Opcja 2 jest wstępnie
    zaznaczona</label>
</section>
<section>
    <input type="checkbox" id="check3" name="checkboxset"
        value="3" />
    <label for="check3"> Opcja 3 – dodaj ich, ile
    chcesz</label>
</section>
</section>
<section>
    <h4>Wybierz tylko jedną opcję</h4>
    <section>
        <input checked="checked" id="radio1"
            name="radioset"
            type="radio" value="Choice_1" />
        <label for="radio1">Opcja 1 jest wstępnie
        zaznaczona, a jej tekst
        jest zawijany, jeśli rozciąga się na kilka wierszy.
        </label>
    </section>
    <section>
        <input id="radio2" name="radioset" type="radio"
            value="Choice_2" />
    </section>
```

sekcja z przełącznikami



```
        <label for="radio2">Opcja 2</label>
    </section>
    <section>
        <input id="radio3" name="radioset" type="radio"
            value="Choice_3" />
        <label for="radio3">Opcja 3</label>
    </section>
</section>
<section>
    <label for="select_choice">Wybierz</label>
    <select id="select_choice" name="select_choice">
        <option value="0">Brak</option>
        <option value="1">Opcja 1</option>
        <option value="2"> Opcja 2</option>
        <option value="3"> Opcja 3</option>
        <option value="4"> Opcja 4</option>
    </select>
</section>
</fieldset>
<section>
    <input type="submit" value="Zatwierdź" />
</section>
</form>
```

sekcja z rozwijanym menu —————|

przycisk zatwierdzający —————| formularz

Objaśnię teraz strukturę kodu, zaczynając od elementu **form**.

ELEMENT FORM

Kod formularza zawarty jest w znaczniku HTML **form**.

```
<form class="stylin_form1" action="process_form.php"
method="post">
    <!-- kod elementów formularza -->
</form>
```



O przetwarzaniu danych z formularzy przeczytasz na stronie <http://www.javascript-coder.com/htmlform/html-form-tutorial-p1.php>. U dołu znajdziesz odnośniki do innych materiałów związanych z tematem formularzy.



Zamiast nadawać arbitralne wartości atrybutom `name` kontrollek, powinieneś określić ich nazwy we współpracy z programistami i administratorem bazy danych, ponieważ warto używać jednakowych nazw danych, gdziekolwiek się znajdują.

Element `form` musi być opatrzony dwoma atrybutami. Wartość atrybutu `action` to adres URL pliku na serwerze, który przetworzy dane formularza po ich wysłaniu, a atrybut `method` (o wartości `post` lub `get`) określa sposób, w jaki dane mają być serwerowi przekazane.

WYSYŁANIE FORMULARZA

Kiedy użytkownik zatwierdza formularz, podane lub wybrane w nim dane zostają przesłane na serwer. Kontrolkami nazywa się ogólnie elementy formularza, które służą do zbierania danych: pola tekstowe, kratki, przełączniki itp. Dane przesypane są na serwer w postaci pary nazwy i wartości, np. `username=chrisconsumer`, gdzie każdej kontrolce przypisana jest jedna para. *Nazwa* jest tekstem przypisanym atrybutowi `name` kontrolki. *Wartość* to albo informacja, którą użytkownik podaje w polu tekstowym kontrolki, albo wartość boolowska stanu kontrolki w rodzaju kratki (`1`, gdy jest zaznaczona, `0`, gdy nie jest).

FIELDSET

Zbiór powiązanych kontrollek formularza można zawiązać w jednym elemencie `fieldset`. W sklepie internetowym kontrolki służące do pobierania loginu użytkownika i jego adresu można zawiązać w elemencie `fieldset` oznaczonym jako „Adres dostawy”. W drugim elemencie `fieldset` można zawiązać kontrolki odpowiedzialne za pobieranie danych karty kredytowej i tym podobne.

Kod mojego formularza podzielony jest na dwie części przy użyciu elementów `fieldset`. Pierwszy z nich zawiera kilka podstawowych par oznaczeń i kontrollek. Drugi zawiera bardziej złożone zestawienia kontrollek: grupy elementów `input` w postaci kratek i przełączników oraz element `select` z kilkoma opcjami. Pierwszym dzieckiem `fieldset` musi być element tekstowy `legend`, w którym zawarta jest nazwa elementu `fieldset`.

```
<fieldset>
    <legend>Adres dostawy</legend>
    <!-- pary oznaczeń i kontrolek -->
</fieldset>
```

Jak widzisz, elementy `legend` obydwu elementów `fieldset` przedstawiam jako szare paski z tekstem, aby oddzielić dwie sekcje formularza.

KONTROLKI I OZNACZENIA

Formularz zawiera jedną lub więcej kontrolek. Jak wspomniałem, kontrolka jest terminem odnoszącym się do elementu formularza, który użytkownik zaznacza, z którego wybiera lub w którym coś wpisuje. Każdą kontrolkę, w której można pisać, potocznie nazywamy polem. Każda kontrolka formularza, z wyjątkiem przycisku `Zatwierdź`, musi być oznaczona elementem tekstowym `label`, który opisuje dane obsługiwane przez kontrolkę. Ilustracją tej struktury jest poniższy kod i [rysunek 6.14](#).

```
<label for="user_name">Login</label>
<input type="text" id="user_name" name="user_name" />
<p>Podaj nazwę użytkownika</p>
```

tworzy pole tekstowe —————

RYSUNEK 6.14. Kontrolka formularza i jej oznaczenie. W tym przypadku kontrolka jest polem tekstowym, w którym można i zwykle powinno się umieścić wskazówkę



To powiązanie atrybutów pozwala również użytkownikowi na zaznaczenie kratki i kliknięcie opcji przełącznika poprzez kliknięcie ich oznaczeń.

Element `label` może obejmować kontrolkę, poprzedzać ją lub znajdować się po niej. Jeżeli oznaczenie, tak jak w powyższym przykładzie, nie obejmuje kontrolki, jego atrybut `for` musi mieć taką samą wartość, jak atrybut `id` kontrolki, by je ze sobą powiązać. Jeżeli jednak kontrolka osadzona jest w oznaczeniu, tak jak poniżej, to są one domyślnie powiązane i nie trzeba podawać atrybutu `for`.

```
<label>Login<input type="text" name="user_name" /></label>
```

Powiązanie oznaczenia z kontrolką na któryś z tych sposobów jest istotne, aby Twoja strona była poprawnie interpretowana przez technologie pomocnicze w rodzaju czytników ekranowych. Sam wolę umieszczać oznaczenie przed kontrolką (choć zamieszczam je za przełącznikami i kratkami), gdyż daje mi to więcej możliwości stylizacji.



Choć nie pokazuję tego w tym przykładzie, możesz utworzyć drugi akapit, przydzielony do klasy `.error`, jeśli chcesz, by informacja o błędzie miała inną treść niż wskazówka.

Poza oznaczeniem i kontrolką często warto dodać wskazówkę dla użytkownika, zwłaszcza jeśli dane wymagane od użytkownika muszą być zapisane w określonym formacie, tak jak w przypadku daty. Wskazówki formularza nie są definiowane określonym elementem HTML, więc sam korzystam z elementu `p`. Staram się także pisać

wskazówkami tak, by tekst można było podświetlić, zwykle na czerwono, by oznaczyć go jako informację o błędzie, gdy użytkownik poda niewłaściwe dane. Dzięki temu nie muszę tworzyć osobnej informacji o błędzie. Dodawanie podświetlenia opiszę w dalszym ciągu omówienia tego przykładu.

RODZAJE KONTROLEK



Pełną listę atrybutów `input` znajdziesz na stronie <https://developer.mozilla.org/en-US/docs/HTML/Element/Input>.

Najczęściej spotykanym elementem HTML formularza jest `input`. Element `input` jest nietypowy pod takim względem, że może różnie wyglądać i zachowywać się na ekranie. Może przyjąć formę pola tekstuowego czy kratki, w zależności od tego, jaką wartość ma jego atrybut `type`. Wartości są następujące:

- `text` — podstawowe pole tekstowe;
- `password` — wpisywane znaki wyświetlane są jako punktory;
- `checkbox` — kratki do zaznaczania (wielokrotny wybór);
- `radio` — przełącznik (jeden wybór);
- `submit` — przycisk, który zatwierdza formularz;
- `time, date, search` — wariacje pola tekstuowego HTML5.

Zwróć uwagę na różne wartości atrybutu `type`, których użyłem w przykładowym kodzie. To one właśnie określają wygląd i zachowanie elementów.

Widać również jedną kontrolkę tekstową, która nie jest oparta na elemencie `input`, a mianowicie wielowierszowe pole `textarea`. Zwróć uwagę, że wskazówka, która wyświetlana jest w tym polu, zanim użytkownik zaczyna coś w nim pisać, dodana została atrybutem `placeholder`.



Pełną listę elementów formularza znajdziesz na stronie <http://reference.sitepoint.com/html-elements-form>.

KRATKI, PRZEŁĄCZNIKI I MENU WYBORU

Te trzy elementy formularzy są bardziej skomplikowane od standar-dowych zestawień kontrolek i oznaczeń.

- Kratki zaznaczania umożliwiają użytkownikowi dokonanie wyboru jednej lub więcej opcji. Wybór ten nie jest wyłączny — wybranie jednej opcji z zestawienia nie uniemożliwia wybrania innych.
- Przełączniki pozwalają użytkownikowi na wybranie jednej opcji ze zbioru. Jest to wybór wyłączny — dokonanie go sprawia, że poprzedni wybór jest anulowany.

Zauważ, że kratki i przełączniki włączane są do zbioru poprzez nadanie jednakowej wartości ich atrybutom `name`; w przypadku przełączników z tego przykładu tą wartością jest `radioset`. Każdy przycisk w tym zbiorze jest określony unikalną wartością atrybutu `value` — w tym przypadku `Choice_1`, `Choice_2` itd. Jeśli więc użytkownik wybierze pierwszy przełącznik, serwerowi przekazana zostanie para nazwy i wartości `radioset=Choice_1`.

- Kontrolka `select` tworzy menu, które rozwija się po kliknięciu. W obrębie tego elementu każda z opcji oznaczona jest elementem tekstowym `option`.

Sposoby kodowania formularzy

Ponieważ nie ma określonego elementu HTML, w którym miałyby być zawierane kontrolki wraz z powiązanymi oznaczeniami, używam w tym celu elementu blokowego `section`, co pozwala mi na sprawniejsze zorganizowanie i wystylizowanie liniowych kontrolek formularza i ich oznaczeń `label`.

KODOWANIE PODSTAWOWYCH PAR KONTROLEK I OZNACZEŃ

Jak widać w przykładowym kodzie, każdą parę oznaczenia i kontrołki zawarłem w elemencie `section`, poprzedzając kontrolkę oznaczeniem:

```
<section>
  <label>...</label>
  <input />
  <p>...</p> <!-- wskazówka dla użytkownika -->
</section>
```

Zawierając każdą parę w elemencie blokowym, schludnie rozmieszcza ją pionowo, a dzięki ich rodzicom mogę je pozycjonować na stronie.

KODOWANIE KRATEK I PRZEŁĄCZNIKÓW

Ponieważ grupy przełączników i kratek składają się z par oznaczeń i kontrolek, całą grupę umieściłem w znaczniku `section` (co sprawia, że wszystkim parom nadany jest ten sam styl CSS), a poszczególne pary we własnych znacznikach `section`.

pierwsza kratka lub przełącznik —|

```
<section>
  <h4>Nagłówek zbioru</h4>
  <section>
    <input />
    <label>...</label>
  </section>
  <section>
    <input />
    <label>...</label>
  </section>
  <p>...</p>
</section>
```

druga kratka lub przełącznik —|

```
<section>
  <input />
  <label>...</label>
</section>
```

wskazówka dla użytkownika —|

```
<p>...</p>
</section>
```

Tym razem oznaczenia zamieściłem po elementach `input`, aby znajdowały się po prawej stronie kontrolek. O ile poszczególne kontrolki (zarówno kratki, jak i przełączniki) mają własne elementy `label`, to elementem `label` nie mogę oznaczyć nagłówka grupy, ponieważ służy on jedynie do oznaczania kontrolek. Do tego celu skorzystałem zatem z elementu `h4`. Jak już wspomniałem, do oznaczenia wskazówki dla użytkownika użyłem znacznika akapitu. Być może w przyszłych wersjach HTML pojawią się elementy służące konkretnie do tych celów, ale na razie będziesz się stykał z różnymi sposobami, w jakie różni programiści tworzą elementy potrzebne do stylizacji formularzy. Przyjrzyjmy się teraz kodowi CSS, którym stylizuję podany wcześniej kod formularza.

Stylizacja formularza

Zacznę od zorganizowania ogólnego układu formularza poprzez określenie stylów elementu `form` i obydwu elementów `fieldset`.

```
form.stylin_form1 {  
ogólna szerokość formularza | width:14em;  
wyśrodkowuje formularz | margin:20px auto;  
względem kontenera | border:1px solid #bbb7ae;  
nagłówek formularza | padding:.5em .5em .15em;  
| }  
| .stylin_form1 h3 {  
nagłówek formularza | margin:0;  
| padding:0 0 .2em .2em;  
| font-weight:600;  
| color:#bbb7ae;  
| }  
obejmuje pary kontrollek | .stylin_form1 fieldset {  
i oznaćzeń | margin:0;  
| padding:0 0 .2em 0;  
| width:100%;  
| border:0;  
| }  
pozycjonowanie znaczników | .stylin_form1 legend {  
legend jest dziwne, | width:100%;  
więc umieszczam tekst | padding:.3em 0;  
w blokowym elemencie span, |  
który następnie stylizuję |  
szary pasek | background:#bbb7ae;  
| }  
określa styl nagłówków | .stylin_form1 legend span {  
oznaczeń legend | display:block;  
| font-size:1em;  
| line-height:1.1em;  
| padding:0 0 0 .4em;
```

```
font-weight:700;
biały tekst na szarym pasku —————| color:#fff;
} 
```

Layout wygląda teraz tak jak na **rysunku 6.15.**

RYSUNEK 6.15. Wystylizowane elementy `form` i `fieldset`

Wystylizowany formularz

Sekcja 1 • Podstawowe kontrolki

* wymagane pole

Login *

Podaj nazwę użytkownika

Hasło *

Hasło musi mieć minimum 8 znaków

Opis

Opis tutaj opis.

Data

rrrr-mm-dd

Sekcja 2 • Kontrolki wielokrotnego wyboru

Zaznacz dowolną liczbę kratek

Jak widać na **rysunku 6.15.** wokół elementu `form` umieściłem dopełnienie, aby odsunąć treść od krawędzi. Elementowi `legend` nadałem wygląd jednobarwnego paska z wydzielonym tekstem. Ponieważ domyślne położenie elementu `legend` określa nieznany nam mechanizm przeglądarki, a nie jej arkusz stylów, nie można go dokładnie kontrolować przy użyciu kodu CSS. To zachowanie można jednak obejść tak, jak tu pokazałem — zamieszczając tekst elementu `legend` w znaczniku `span` z właściwością `display:block` i określając jego położenie. Następnie zdefiniuję style kontrolek i oznaczeń pierwszego elementu `fieldset`.

```
.stylin_form1 section { 
```

sprawia, że element `section` —————| `overflow:hidden;`
obejmuje kontrolki
i oznaczenia formularza

```
padding:.2em 0 .4em 0;  
oddziela graficznie kolejne sekcje — border-bottom:8px solid #e7e5df;  
}  
  
brak obramowania ostatniej — .stylin_form1 section:last-child {  
sekcji każdej z grup  
border-bottom:0px;  
}  
  
oznaczenia formularzy — .stylin_form1 section label,  
element h4 zawiera nazwy — .stylin_form1 section h4 {  
zbiorów kratek i przełączników  
display:block;  
clear:both;  
  
prawy margines sprawia, — margin:.3em .3em 0 0;  
że tekst oznaczenia zostaje  
zawinięty, zanim zetknie się  
z obszarem elementu input  
padding-bottom:.1em;  
font-size:.8em;  
font-family:"Droid Sans";  
font-weight:400;  
line-height:1.1;  
}  
  
gwiazdka, która oznacza — .stylin_form1 section label span,  
wymagane pola  
.stylin_form1 section h4 span {  
font-size:.75em;  
vertical-align:text-top;  
color:#f00;  
}  
  
wymagane pola tekstowe — .stylin_form1 section p.note {  
font-size:.7em;  
color:#f00;  
margin:0;  
padding:0 0 .3em 0;
```

```

        }

.stylin_form1 section input,
.stylin_form1 section textarea,
.stylin_form1 section select {
    margin:.2em .5em .2em 0;
    padding:.2em .4em;
    color:#000;
    box-shadow:1px 1px 3px #ccc;
    font-size:.8em;
}

font-family:inherit;
outline:none;
}

.stylin_form1 section input,
.stylin_form1 section textarea {
    width:12em;
    border:1px solid #bbb7ae;
    border-radius:3px;
}

height:5em;
margin-top:.3em;
line-height:1.1;
}

.stylin_form1 section p {
    margin:.3em .75em 0;
    clear:both;
    font-size:.7em;
    line-height:1.1;
}

```

tworzy przestrzeń wokół tekstu kontrolki

bez tej deklaracji Firefox używały fonta Courier

usuwa domyślną, niebieską obwódkę, wskazującą sfokusowany obszar

określa styl pól tekstowych (typu text, password, date, textarea itd.)

określa szerokość pól

tworzy zaokrąglone narożniki

wysokość pola textarea

tworzy przestrzeń pod oznaczeniem

tekst wskazówki

```

        color:#000;
    }
    .stylin_form1 section p.error {
        color:#f00;
    }

```

podświetla tekst wskazówki na czerwono, gdy dodana jest klasa error

RYSUNEK 6.16. Kontrolki pierwszego elementu są w pełni wystylizowane

The screenshot shows a 'Wystylizowany formularz' (Stylized form) with the following structure:

- Sekcja 1 • Podstawowe kontrolki**
 - wymagane pole**
 - Login ***: An input field with placeholder "Podaj nazwę użytkownika".
 - Hasełko ***: An input field with placeholder "Hasło musi mieć minimum 8 znaków".
 - Opis**: A text area with placeholder "Wprowadź tutaj opis.".
- Data**: A date input field with placeholder "rrrr-mm-dd" and a calendar icon.
- Sekcja 2 • Kontrolki wielokrotnego wyboru**
 - Zaznacz dowolną liczbę kratek**: A checkbox labeled "Opcja 1".

Jak widać na rysunku 6.16, wszystkim kontrolkom, które są polami (czyli `text`, `date`, `textarea` i `select`) nadałem jednakową szerokość, obstylowałem tekst wskazówki oraz dodałem grube dolne obramowanie, aby oddzielić graficznie wszystkie elementy `section`.

Małe czerwone gwiazdki znajdujące się na końcu tekstu oznaczają niektórych pól wskazują, że dane pole trzeba wypełnić i użytkownik nie może go pozostawić pustego. Gwiazdki (które uzyskuje się, naciskając `Shift+8`) są umieszczone w elementach `span`, aby można było określić ich położenie względem tekstu i nadać im inny kolor. Zwrót jeszcze uwagę na klasę `p.error`, odnoszącą się do akapitów ze wskazówkami. Ta klasa będzie nadawana powiązanym akapitom ze wskazówkami, jeśli umieszczone w kontrolce dane nie będą poprawne. Tekst wskazówki wyświetlany jest wtedy na czerwono, wskazując użytkownikowi, które dane trzeba zmienić przed ponownym zatwierdzeniem formularza.



Klasa `error` jest dodawana przez kod odpowiadający za walidację formularza.

Zajmę się teraz kratkami, przełącznikami i elementem `select`, które znajdują się w drugim znaczniku `fieldset`. Jak widać na rysunku 6.16, ich oznaczenia znajdują się pod kontrolkami, a wolałbym, żeby były położone obok siebie. Zanim przyjrzesz się kolejnemu fragmentowi kodu CSS, możesz rzucić okiem na punkt „Sposoby kodowania formularzy”, aby przypomnieć sobie wygląd kodu HTML kratek i przełączników.

wewnętrzny kontener
par oznaczeń i kontrolek

sprawia, że element obejmuje
pływające oznaczenia

przełącznik lub kratka

wyzerowuje odziedziczoną
szerokość

góra krawędź zostaje
wyrównana do oznaczenia,
lewa sprawia, że kontener
nie przycina elementu input

wyzerowuje odziedziczoną
wartość

tworzy odstępy między
sąsiadującymi kratkami
oraz między każdą kratką
a jej oznaczeniem

wyzerowuje odziedziczoną
wartość

```
.stylin_form1 section section {  
    overflow:hidden;  
    margin:.2em 0 .3em .4em;  
    padding:0 0 .1em 0;  
    border-bottom:none;  
}  
  
.stylin_form1 section section input {  
    float:left;  
    clear:both;  
    width:auto;  
    margin:.1em 0 0 .3em;  
}  
  
.stylin_form1 section section label {  
    float:left;  
    clear:none;  
    width:15em;  
    margin:.15em 0 0 .6em;  
    font-size:.7em;  
    font-weight:normal;  
    line-height:1.2;  
}
```

```
.stylin_form1 section select {  
    margin-left:.4em;  
    font-size:.85em;  
}  
  
przycisk zatwierdzania ————— .stylin_form1 section input[type="submit"] {  
wyzerowuje ustawienie ————— width:auto;  
szerokości odziedziczone  
po innych elementach input  
    margin:.4em .3em 0 0;  
    font-size:1em;  
    font-weight:800;  
    color:#fff;  
    background-color:#bbb7ae;  
    cursor:pointer;  
}  
  
zmienia się w rączce, gdy cursor ————— .stylin_form1 > section:last-child {  
znajduje się nad przyciskiem  
zatwierdzania  
    text-align:center;  
}  
  
wyśrodkowuje przycisk ————— .stylin_form1 > section:last-child {  
zatwierdzania
```



Zauważ, że przełączniki i kratki zachowują się jak grupa, gdy przypisze się im wspólną wartość atrybutu `name`. Wartość atrybutu `name` przekazywana jest w ramach pary nazwy i wartości.

Nagłówek każdego zbioru przełączników i kratek wygląda jak oznaczenia z pierwszego elementu `fieldset` formularza, ale w rzeczywistości jest elementem `h4`. Elementów `label` używam tylko tam, gdzie występuje jakaś powiązana kontrolka, więc znacznik `h4` obstylowałem, dodając selektor `h4` do stylu elementów `label` z pierwszej części formularza. Następnie wprowadzam zmodyfikowany styl oznaczenia dla pomniejszych oznaczeń, które towarzyszą wszystkim kratkom i przełącznikom. Dodałem także kilka stylów elementowi `select`, by go lepiej wyrównać i nieco go pomniejszyć, redukując wielkość jego tekstu. Przycisk *Zatwierdź* nie dziedziczy już szerokości wcześniejszych reguł związanych z elementem `input` i wyśrodkowany jest przy dolnej krawędzi formularza, jasno wskazując użytkownikowi, że naciśnięcie go jest ostatnim krokiem całego procesu (**rysunek 6.17**).

RYSUNEK 6.17. Kratki i przełączniki są teraz umieszczone w rzędach, a przycisk zatwierdzania został obstylowany

Wersja formularza z oznaczeniami nad kontrolkami jest już gotowa. Pokażę Ci następnie dodatkowe style, dzięki którym oznaczenia `form` będą się znajdować po lewej stronie pól. Aby skorzystać z tego za-stępczego stylu, nadaj klasę `labels_left` elementowi `form`.

poszerza formularz, żeby ————— starczyło miejsca dla oznaczeń po lewej

```
form.stylin_form1.labels_left {
    width:22em;
}
```

zmienia oznaczenia w płynne elementy, które znajdują się po lewej stronie kontrolek —————

```
form.stylin_form1.labels_left label,
form.stylin_form1.labels_left h4 {
    float:left;
    width:8em;
}
```

wcina wskazówki, by znajdowały się pod kontrolkami —————

```
form.stylin_form1.labels_left p {
    margin:0 0 0 9.35em;
    padding:.3em 0 0 0;
```

sprawia, że wskazówki nie przemieszczają się do miejsca obok płynących oznaczeń lub kontrolek —————

```
clear:both;
```

przestrzeń znajdująca się pod —| `form.stylin_form1.labels_left p.note {`
informacją o wymaganym polu
`margin:0 0 .2em 0;`
`}`

kontener każdego przełącznika —| `form.stylin_form1.labels_left section section {`
i kratki wraz z oznaczeniem
`width:10em;`
`margin-left:6.5em;`
`padding-top:0;`
`}`

szerokość przełącznika lub kratki —| `form.stylin_form1.labels_left section input {`
`width:1.25em;`
`margin-left:0;`
`}`

tworzy prawą kolumnę kontrolek —| `.stylin_form1.labels_left section input,`
`.stylin_form1.labels_left section textarea,`
`.stylin_form1.labels_left section select {`
`float:left;`
`width:12em;`
`}`

tworzy wcięcie elementów select —| `.stylin_form1.labels_left section select {`
`margin-left:.2em;`
`}`

sprawia, że „pływanie” nie jest —| `.stylin_form1.labels_left > section input[type=submit] {`
dziedziczone od pozostałych
elementów input
`float:none;`
`}`

Na **rysunkach 6.18a i b** widać obydwie wersje w ukończonej postaci. W wersji z oznaczeniami po lewej klasa `error` została nadana wszystkim elementom ze wskazówkami.

RYSUNEK 6.18A i B. Dwie wersje formularzy, z oznaczeniami ponad kontrolkami i obok nich. Komunikaty o błędach wyświetlane są na rysunku 6.18b

Wystylizowany formularz

Sekcja 1 • Podstawowe kontrolki

* wymagane pole
Login *
Podaj nazwę użytkownika

Haseł *
Haseł musi mieć minimum 8 znaków

Opis
Wprowadź tutaj opis.

Data
mm-mm-yy

Sekcja 2 • Kontrolki wielokrotnego wyboru

Zaznacz dowolną liczbę kratek
 Opcja 1
 Opcja 2 jest wstępnie zaznaczona
 Opcja 3 — dodaj ich, ile chcesz
Musisz wybrać przynajmniej jedną

Wybierz tylko jedną opcję
 Opcja 1 jest wstępnie zaznaczona, a jej tekst jest zwiędły, jeśli rozrasta się na kilka wierszy.
 Opcja 2
 Opcja 3

Wybierz
Brak

Wystylizowany formularz

Sekcja 1 • Podstawowe kontrolki

* wymagane pole
Login *
Podaj nazwę użytkownika

Haseł *
Haseł musi mieć minimum 8 znaków

Opis
Wprowadź tutaj opis.

Data
mm-mm-yy

Sekcja 2 • Kontrolki wielokrotnego wyboru

Zaznacz dowolną liczbę kratek
 Opcja 1
 Opcja 2 jest wstępnie zaznaczona
 Opcja 3 — dodaj ich, ile chcesz
Musisz wybrać przynajmniej jedną

Wybierz tylko jedną opcję
 Opcja 1 jest wstępnie zaznaczona, a jej tekst jest zwiędły, jeśli rozrasta się na kilka wierszy.
 Opcja 2
 Opcja 3

Wybierz
Brak

Zatwierdź

Jak widać po dużej ilości kodu HTML i CSS w tym przykładzie, stylizacja formularza jest zajęciem skomplikowanym i czasochłonnym. Z tego powodu opracowałem i skomentowałem kod i arkusz stylów tego podstawowego przykładu formularza tak, byś mógł bez problemu korzystać z niego jako podstawy własnych formularzy. Skopiuj bloki kodu HTML lub wykorzystaj jego strukturę, aby stworzyć pożądane kontrolki, a następnie dołącz do strony kod CSS [form](#). Następnie nadaj klasę `stylin_form1` elementowi `form`, aby w mgnieniu oka nadać mu wygląd przedstawiony powyżej, który będziesz mógł dostosować do własnych potrzeb.

Formularz wyszukiwania

Każda witryna oferuje jakąś możliwość wyszukiwania w jej obrębie. Pole wyszukiwania może nie kojarzyć Ci się z formularzem, ale tak właśnie jest — to formularz składający się z jednego pola. Oto prosty komponent wyszukiwania, który idealnie nadaje się do umieszczenia np. w prawym górnym rogu nagłówka, czyli tam, gdzie zwykle można go zobaczyć. Pole wyszukiwania na stronie Apple'a jest drobnym, dyskretnym komponentem, który powiększa się po kliknięciu, tworząc więcej miejsca dla tekstu kwerendy. Nie ma on nawet przycisku, który trzeba by kliknąć — zakłada się, że użytkownik wie, iż powinien kliknąć pole, co wpisać i nacisnąć *Enter*. Do utworzenia takiego komponentu wystarczy użyć elementu `form`, zawierającego oznaczenie `label` i kontrolkę `input`. W tym przykładzie umieszczę formularz w znaczniku `header`, aby uzyskać nagłówek, w którym ten komponent będzie się znajdował.

```
<header>
  <form class="stylin_form_search1" action="#" method="post">
    <label for="search">search</label>
    <input type="search" id="search" name="search" placeholder="szukaj" />
  </form>
</header>
```

Ten przykład daje też sposobność do wprowadzenia przejść CSS3, które pozwalają na animowanie pewnych właściwości CSS. W tym przypadku utworzę przejście będące zmianą szerokości pola tekstowego. Oto kod CSS:

```
* {margin:0; padding:0;}
header {
  font-family:helvetica, arial, sans-serif;
  display:block;
  overflow:hidden;
  width:500px;
  margin:30px;
  border-radius:6px;
  background-color:#ddd;
}
```

określa styl nagłówka  `header {`

Przejścia CSS3

Przejścia CSS3 pozwalają na animowanie właściwości CSS. Zmiana dokonująca się natychmiast po wyzwoleniu jakiegoś zdarzenia, np. zmiana koloru odnośnika po najechaniu na niego kursorem, może odbywać się stopniowo przez określony czas. Pierwsza reguła CSS określa początkowy stan właściwości i przejścia. Druga reguła określa nowy stan, w który właściwość ma przejść po wyzwoleniu zdarzenia.

Poniższe dwie reguły sprawiają, że obramowanie kontrolki formularza zmienia kolor z czarnego na zielony przez dwie sekundy po kliknięciu pola.

```
input {border-color:black; transition:border-color 2s;}
input:focus {border-color:green;}
```

Miej na uwadze, że stosowanie przejść wymaga podania prefiksów dla wszystkich przeglądarek.

Przejścia są standardowo wyzwalane przy użyciu reguły z pseudoklasą `:hover`, kiedy użytkownik przesuwa kurSOR nad jakiś element, bądź pseudoklasą `:focus`, kiedy element formularza zostaje sfokusowany. Ponadto, przejście można wyzwolić poprzez zdefiniowanie nowego stanu regułą z selektorem klasy i dodanie nazwy tej klasy elementowi przy użyciu JavaScript (lub frameworka JavaScript w rodzaju jQuery czy MooTools), kiedy następuje kliknięcie lub inne zdarzenie.

Istnieje pięć właściwości przejścia:

- `transition-property` — nazwa właściwości CSS, która ma ulec zmianie, np. `color` lub `width`.
- `transition-duration` — określony sekundami lub milisekundami czas, przez jaki przejście ma się odbywać, np. `2s, 500ms`.
- `transition-timing-function` — określa, czy przyspieszenie przejścia ma być zmienne na początku lub na końcu, np. `ease-in, ease-out, ease-in-out` lub `linear` (które jest ustawieniem domyślnym).
- `transition-delay` — określone sekundami lub milisekundami opóźnienie, z jakim przejście zaczyna się po wyzwoleniu, np. `1s, 200ms`.
- `transition` — właściwość zbiorcza, np. `transition:color 2s ease-in 1ms;`.

Wiele właściwości, choć nie wszystkie, można animować przy użyciu właściwości `transition`. Pełną listę „animowalnych” właściwości znajdziesz na stronie <http://www.w3.org/TR/css3-transitions/#animatable-properties>. Znajduje się na niej pełna specyfikacja CSS3 Transitions Module w stylu W3C. Dobre omówienie przejść CSS3 znajduje się też na stronie <http://www.css3.info/preview/css3-transitions>.

```
form.stylin_form_search1 {
    float:right;
    width:200px;
    margin:5px; padding:5px;
}
form.stylin_form_search1 input {
```

kontener oznaczenia  i kontrolki

```

float:right;
width:70px;
padding:2px 0 3px 5px;
outline:none;
font-size:.8em; border-color:#eee #ccc #ccc #eee;
border-radius: 10px; -webkit-transition:2s width;
}

form.stylin_form_search1 input:focus {width:200px;}
form.stylin_form_search1 label {display:none;}

```

usuwa domyślne podświetlenie obramowania —————|
dodaje prefiks dla innych przeglądark —————|
to oznaczenie jest wymagane, —————| ale nie widać go na ekranie

RYSUNEK 6.19A i B. Małe pole wyszukiwania poszerza się stopniowo po kliknięciu, a tekst zastępczy zostaje zastąpiony tekstem podanym przez użytkownika



Element `form` ma określoną szerokość i spływa w prawo, a zawarty w nim element `input` również spływa w prawo (**rysunek 6.19a**). Element `label` nie jest widoczny na ekranie, ale musi być obecny w kodzie. Tekst widoczny w polu utworzyłem przy użyciu atrybutu `placeholder`; zostaje on automatycznie ukryty, kiedy użytkownik zaczyna pisać (**rysunek 6.19b**).

DODANIE PRZEJŚCIA CSS3

W powyższym kodzie CSS reguła `input` określa szerokość pola na 70 pikseli, a reguła `input:focus` określa szerokość pola jako 200 pikseli. Kiedy pole zostaje sfokusowane (tzn. uaktywnione na potrzeby klawiatury; po kliknięciu go pojawiają się w nim znaki wpisywane na klawiaturze), zmienia ono szerokość. Ze względu jednak na regułę `transition:2s width;`, zamiast od razu zmieniać szerokość, pole rozciąga się płynnie przez dwie sekundy. Zwróć uwagę, że przejścia CSS3 umieszczone są w regule z wejściowym stanem, a nie stanem końcowym. Przejścia wymagają zastosowania prefiksów, ale w tym przykładzie korzystam jedynie z prefiksu `webkit`, obsługiwanyego przez Safari i Chrome. Ten komponent w kontekście w pełni wystylizowanej strony przedstawię w następnym rozdziale. Więcej na ten temat przejść CSS przeczytasz w ramce „Przejścia CSS3”.

Chmurka

Chmurka jest komponentem wyświetlany po najechaniu kursorem na element. Jest to skuteczny sposób na podawanie dodatkowych informacji, gdy brakuje miejsca, ponieważ użytkownicy intuicyjnie przesuwają cursor nad interesujące ich elementy. Utworzenie chmurki może się wydawać proste, ale wkrótce przekonasz się, dlaczego pozostawiłem ją sobie na koniec. Na jej podstawie zaprezentuję Ci dwa bardzo potężne, ale słabo rozumiane funkcje CSS: właściwość `z-index` i dynamicznie generowane elementy HTML. Użyję w tym przykładzie trzech obrazów z podpisami. Oto kod HTML, w którym korzystam z nowych elementów HTML5 `figure` i `figcaption`.

```
<figure>
  
  <figcaption>
    <h3>Różowe szpilki</h3>
    <a href="#">Więcej</a>
  </figcaption>
</figure>
<figure>
  
  <figcaption>
    <h3>Szpilki w panterkę</h3>
    <a href="#">Więcej</a>
  </figcaption>
</figure>
<figure>
  
  <figcaption>
    <h3>Czerwone szpilki na koturnie</h3>
    <a href="#">Więcej</a>
  </figcaption>
</figure>
```

Zauważ, że element `figcaption` może wystąpić w elemencie `figure` raz, przy czym musi być w nim osadzony jako pierwsze lub ostatnie dziecko. Zacznę od wystylizowania elementu `figure` i utworzenia z niego pola wokół obrazu.

```
figure {  
    wielkość pola z obrazem | width:144px;  
    wielkość pola z obrazem | height:153px;  
    odstęp między polami | margin:20px 20px;  
    obramowanie obrazu | border:1px solid #666;  
    kontekst pozycjonowania chmurek | position:relative;  
    sprawia, że obrazy rozmieszczone | float:left;  
    są obok siebie  
}  
  
usuwa odstęp pod obrazem | img {display:block;}
```

RYSUNEK 6.20. Obrazy mają teraz obramowania i są rozmieszczone obok siebie



Jak widać na rysunku 6.20, obramowania elementów `figure` dopasowane są wielkością tak, by ściśle obejmowały obrazy; są to elementy płynące, dzięki czemu mogą się znajdować obok siebie. Elementy `figcaption` na razie wyświetlane są w domyślnych miejscach.

W następnym kroku elementy `figcaption` przekształcimy w chmurki. Zauważ, że nadal obrazom właściwość `display:block`. Zrobiłem to, ponieważ obrazy są domyślnie elementami liniowymi i są dorównywane do linii bazowej tekstu, a nie dolnej krawędzi swoich kontenerów. W wyniku tego obrazy umieszczone w elementach blokowych mają pod sobą drobne odstępy. Przekształcenie obrazu w element blokowy pozwala temu zaradzić. Elementom `figure` nadal położenie względne, aby móc względem nich określić położenie `figcaption`.

```
figcaption {  
    ukrywa chmurki → display:none;  
    określa położenie → position:absolute;  
    umieszcza chmurkę po prawej → left:74%; top:14px;  
    stronie obrazu  
    szerokość chmurki → width:130px;  
    obszar wokół zawartości chmurki → padding:10px;  
    background:#f2eaea;  
    border:3px solid red;  
    border-radius:6px;  
}  
  
wyświetla chmurkę, kiedy kurSOR → figure:hover figcaption {display:block;}  
znajduje się nad obrazem  
treść chmurki → figcaption h3 {  
    font-size:14px;  
    color:#666;  
    margin-bottom:6px;  
}  
  
treść chmurki → figcaption a {  
    display:block;  
    text-decoration:none;  
    font-size:12px;  
    color:#000;  
}
```

W powyższym kodzie CSS zastosowałem tę samą technikę, polegającą na użyciu właściwości `display` do ukrycia tekstu i wyświetlenia go po najechaniu kursorem na element, co wcześniej przy tworzeniu rozwijanych menu. Aby umieścić lewą krawędź chmurki wewnątrz pola `figure`, nadałem jego właściwości `left` wartość 74%. Mogłeś się domyślać, że użyłbym raczej właściwości `right` w tym celu, ale wtedy określiłbym relację między prawymi krawędziami obrazu i chmurki, a nie lewymi.

RYSUNEK 6.21A, B i C. Chmurka pojawia się po najechaniu kursorem na powiązany z nią obraz. Na rysunkach 6.36a i b chmurki pierwszego i drugiego obrazu są jednak zakryte przez obrazy po ich prawej



Stosy i z-index

Na rysunkach 6.21a i b widać pewien problem — chmurki pierwszego i drugiego obrazu zakryte są obrazami po ich prawej. Wynika to z kolejności rozmieszczenia elementów `figure` w stosie. Kiedy umieszczasz braci w kontenerze, tak jak tutaj, w przypadku trzech elementów `figure` w elemencie `body`, każdy z nich tworzy własny stos; ich dzieci umieszczane są w stosie ponad nimi. Gdybyś nałożył pierwsze dwa elementy `figure` na siebie, pierwszy element `figure i wszystkie jego dzieci w jego stosie` znalazłyby się pod drugim elementem. Chmurka znajduje się w stosie pierwszego elementu `figure`, więc domyślnie wyświetlna jest pod wszystkimi elementami stosu drugiego elementu.

Właściwość CSS `z-index` określa kolejność elementów w stosie i tym samym pozwala na zmianę ich domyślnej kolejności. Elementy o wyższej wartości `z-index` znajdują się nad elementami o niższej wartości. Właściwość `z-index` może mieć dowolną wartość od zera wzwyż. Wartości ujemne też są dopuszczalne, ale w niektórych

przeglądarkach mogą nie działać poprawnie. Właściwość **z-index** wszystkich stosów ma domyślnie wartość **auto**, czyli tożsamą z zerem.

Właściwość **z-index** działa jednak tylko z elementami, których właściwość **position** ma wartość inną niż **static**. Innymi słowy obydwa elementy muszą mieć wartość **absolute**, **relative** lub **fixed**. W tym przypadku chmurka jest już pozycjonowana bezwzględnie wobec względnie pozycjonowanych elementów **figure**, więc muszę już tylko nadać chmurce **z-index** o wartości większej niż 0, np. 2.

umieszcza na pierwszym planie chmurkę elementu, nad którym znajduje się kursor

figure:hover figcaption {display:block; z-index:2;}

RYSUNEK 6.22. Po nadaniu wartości **z-index** pływającemu elementowi **figure** jego dziecko wyświetlane jest nad innymi obrazami



Nauczyłem się, by w takich sytuacjach nadawać wartości **z-index** elementom znajdującym się w stanie **hover**, zamiast wszystkim elementom, żeby zapobiec przycinaniu chmurki. Używając techniki „**z-index** po najechaniu kursem”, możesz swobodnie rozmieszczać obrazy ze świadomością, że uaktywniona chmurka będzie widoczna na pierwszym planie strony (rysunek 6.22).

Tworzenie trójkąta w CSS

Chciałbym teraz wzmacnić połączenie między chmurką a obrazem, co zrobię poprzez dodanie do lewej krawędzi chmurki małej trójkątnej strzałki, wskazującej obraz. Technikę polegającą na wykorzystaniu krawędzi obramowania pola do utworzenia trójkąta omówię teraz na przykładzie elementu **div**.

RYSUNEK 6.23. Trójkąt można utworzyć z szerokiego obramowania pola



Oto kod, który pozwala na uzyskanie ostatniej figury z **rysunku 6.23**.

```
div {  
    border:12px solid;  
    border-color:transparent red transparent transparent;  
    height:0px;  
    width:0px;  
}
```

Jak widać na **rysunku 6.23**, trójkąt można utworzyć przy użyciu samego kodu CSS, poprzez stworzenie pola z szerokim obramowaniem, którego szerokości i wysokości nadaje się wartość **0**, a trzem obramowaniom wartość **transparent**. Do powyższego dodam teraz pseudoelement **::before**. Pseudoelementami **::before** i **::after** zwykle dodaje się drobne elementy w rodzaju tekstu lub ikonek, ale utworzone nimi pola elementów można w pełni wystylizować, tak jak wszelkie inne elementy kodu. Pole pseudoelementu przedstawię tutaj jako trójkąt i zamieszcze je przy lewej krawędzi chmurki.

```
pole czerwonego trójkąta ————— figcaption:::after {  
konieczna jest jakaś treść ————— content:"";  
— tu akurat pusty ciąg tekstowy ————— position:absolute;  
———— border:12px solid;  
———— border-color:transparent red transparent transparent;  
umieszcza trójkąt na obramowaniu pola ————— right:100%; top:17px;  
zmniejsza pole, by stworzyć ————— height:0px; width:0px;  
trójkąt }  
}
```

RYSUNEK 6.24. Trójkąt łączy się graficznie z chmurką danego obrazu



Wygenerowany kodem trójkąt jest pozycjonowany bezwzględnie wobec chmurki (**rysunek 6.24**), który jest z kolei pozycjonowany bezwzględnie wobec obrazu. Te dwadzieścia linijek kodu CSS pozwoli Ci wyświetlać dowolną liczbę elementów z chmurkami. Kto powiedział, że programowanie nie może być kreatywne? Zauważ, że właściwości `content` pseudoelementu `::before` trzeba nadać jakąś wartość, żeby wygenerowany element w ogóle się wyświetlał. Ponieważ nie potrzebuję żadnej konkretnie treści, zwyczajnie podałem pusty ciąg tekstowy poprzez wstawienie otwierającego i zamknięjącego cudzysłowu. Jak wrażenia? Ciekawsze to od formularzy, nieprawdaż?

Podsumowanie

W tym rozdziale nauczyłeś się stylizowania niektórych z najpopularniejszych komponentów interfejsów internetowych: menu, formularzy i chmurek. Wszystkie przykłady z tego rozdziału są dostępne do pobrania i napisane tak, byś mógł je wykorzystać we własnych projektach. Tak czy inaczej, najlepszym sposobem na rozwinięcie umiejętności z zakresu kodowania w CSS jest odtworzenie tych komponentów samodzielnie, gdyż w ten sposób ugruntowuje się najważniejsze założenia i techniki związane z wyświetlaniem, pozycjonowaniem, tłami i innymi powszechnie stosowanymi funkcjami CSS. Połączmy teraz techniki tworzenia layoutów z poprzedniego rozdziału z przedstawionymi w tym rozdziale komponentami interfejsu i stworzmy pełnoprawną witrynę.

ROZDZIAŁ 7

Strona internetowa z CSS3

W tym rozdziale wykorzystam techniki tworzenia layoutów z rozdziału 5. oraz techniki stylizacji komponentów z rozdziału 6., by stworzyć pełnoprawną witrynę. Zapoznasz się z wieloma funkcjami stylistycznymi CSS3, m.in. zaokrąglonymi rogami, cieniami tekstu i pól, przejściami oraz przekształceniami, które nadają oprawie graficznej bardziej nowoczesny i profesjonalny wygląd.

Witryna, którą stworzę w tym rozdziale, to ta, którą tworzę równolegle z tą książką — jej nowa strona internetowa. Tak jak zawsze, zaprezentuję na tym przykładzie nowe techniki, z których będziesz mógł korzystać we własnych projektach, oraz zwróci uwagę na utrudnienia, z którymi możesz się zetknąć.

Pokażę Ci najpierw, jak rozplanować szkielet strony, a następnie przeprowadzę krok po kroku przez proces nadawania stylów CSS każdemu obszarowi strony. Po przeczytaniu tego rozdziału będziesz wiedział, czego potrzeba do stworzenia pełnej witryny, i będziesz gotów tworzyć własne.

Struktura strony

Kiedy tworzysz jakąkolwiek bardziej skomplikowaną stronę, musisz napisać setki linijek kodu HTML i CSS. Warto zatem, żeby był uporządkowany. Ważne jest, aby nadać kodowi logiczną strukturę i myśleć hierarchicznie, żeby porządek kodu CSS był zgodny z kodem HTML. Tak właśnie sformatowany jest kod kolejnego przykładu i zdecydowanie polecam przyjęcie takiego podejścia. Wymaga to pewnej dyscypliny, ale zdecydowanie się opłaca, ponieważ łatwo dzięki temu znaleźć kod CSS odnoszący się do dowolnego fragmentu kodu HTML. Ponadto możesz dzięki temu uniknąć dezorientującej sytu-

acji, w której kilka rozrzuconych po kodzie reguł nadaje style jednemu elementowi. To podejście przede wszystkim jednak ułatwia zrozumienie kodu i jego edycję, zarówno przeze mnie, jak i przez innych.

Rzućmy najpierw okiem na ukończoną witrynę (rysunek 7.1).

The screenshot shows a blog post titled "Klasy CSS w jQuery" from September 7, 2012. The post features a photo of Charles Wyke-Smith and a large, bold, stylized letter 'S' at the beginning of the text. The text discusses the use of CSS classes with jQuery. On the right side of the page, there's a sidebar with a newsletter sign-up form, recent posts, and book recommendations for CSS, JavaScript, and PHP.

Strona o CSS
Blog i książki Charlesa Wyke-Smitha

Artykuły Książki Materiały Biblioteczka Kontakt

szukaj

/ września 2012

Klasy CSS w jQuery

 **S**INTUS AT NEQUE IN MAGNA LACINIA BIBENDUM SIT AMET SIT AMET dolor. Phasellus pretium gravida interdum. Nam interdum posuere tempus. Ut commodo laoreet dolor, non hendrerit mi dictum vitae. Nam nec egestas libero. Quisque sodales tortor ut tortor egestas eu adipiscing enim pellentesque. Cras condimentum tellus in nisl sodales tortor ut tortor egestas eu adipiscing enim tincidunt et ornare augue dignissim. Nam laoreet elit vitae lorem tincidunt adipiscing. Maeccenas pharetra mattis sodales tortor ut tortor egestas eu adipiscing enim urna, eu adipisciing diam pharetra ac.

Pobierz kody i aktualizacje

E-mail:
Hasło:

Zapisz się [Zrób to teraz!](#)

Nie jesteś zarejestrowany? [Zaloguj się](#)

Ostatnie wpisy

- Z-index — problemy się nawarstwiają
- Jak używać box-image
- Cienie w CSS3

 **Stylin' with CSS3**
HTML5 + CSS3

 **Scriptin' with JavaScript and Ajax**
JavaScript

 **Codin' for the Web**
PHP + SQL

 **Visual Stylin' with CSS3**
Prezentacja

Szablon CSS z książki **CSS. Witryny internetowe szyte na miarę, wydanie trzecie** Charlesa Wyke-Smitha

[Polityka prywatności](#) | [Kontakt](#)

RYSUNEK 7.1. Oto ukończona witryna

Planowanie kodu HTML

Kiedy dopiero zaczynasz pracę z HTML, przełożenie projektu graficznego na elementy kodu może być nie lada wyzwaniem. Można jednak przyjąć pewne dobre podejście. Przed rozpoczęciem kodowania strony sam projekt zwykle tworzy się w Photoshopie, Fireworks lub chociaż na kartce papieru. To na tym etapie możesz omówić projekt z klientem, a następnie dostosować go, by mieć pewność, że oprawa graficzna i organizacja treści zgadzają się z wymogami projektu. Następnie należy zabrać się za kodowanie stron. Pierwszym krokiem jest rozrysowanie w layoucie pól przedstawiających główny, strukturalny kod HTML.

Ponieważ kod HTML tworzy prostokątne elementy, powinieneś znaleźć sposób, by podzielić layout na kilka możliwie największych obszarów, określających najwyższe poziomy kodu, a następnie podzielić je prostokątnymi polami na mniejsze sekcje, zawierające zstępne elementy strukturalne. Oto, jak ten podział wygląda na mojej stronie.



RYSUNEK 7.2. Struktura pól na stronie pozwala mi wyłonić trzy poziomy strukturalne

Jak widać na **rysunku 7.2**, strona podzielona jest schludnie na cztery prostokątne (pomarańczowe) pola, zajmujące całą szerokość strony, które odnoszą się do elementów HTML najwyższego poziomu. Zauważ, że zamierzam także zatrzymać cały layout w jednym kontenerze (oznaczonym tu na zielono), aby móc z łatwością ustawać ogólną szerokość layoutu i wyśrodkować treść w oknie przeglądarki. Warto na tym etapie również wybrać klasy i identyfikatory głównych znaczników. Elementy HTML `header` i `footer` nie potrzebują klasy ani identyfikatora, gdyż w dokumencie występują tylko raz, ale dwa środkowe prostokąty — elementy `section` — trzeba będzie odróżnić identyfikatorami: `feature_area` i `book_area`.



Fakt, że obszar nawigacji pokrywa się z obszarem tytułu i pola wyszukiwania, wynika z tego, że już postanowiliem, iż te dwa ostatnie obszary będą pozyjonowane bezwzględnie. Sprawia to, że element `nav` ignoruje je i wypełnia cały nagłówek. Będę mógł następnie wyśrodkować zawarte w nim menu na stronie, co zademonstruję w dalszej części rozdziału.

Kolejnym krokiem jest rozplanowanie struktury drugiego poziomu, która oznaczona tu jest niebieskimi prostokątami. Przyglądam się każdemu elementowi najwyższego poziomu, aby określić największe elementy, w jakich można zorganizować ich zawartość.

W nagłówku `header` znajdują się trzy zbiory treści: obszar tytułu po lewej, menu nawigacyjne pośrodku oraz formularz wyszukiwania po prawej. W obszarze `feature_area` wpis blogowy umieściłem po lewej, a po prawej obszar `aside` z polem logowania i listą odnośników do wpisów. Zauważ, że w jednym elemencie zawarłem *obydwie* te komponenty; obszar ten podzielił dalej w kolejnym kroku.

W obszarze `book_area` znajdują się cztery kontenery na cztery książki. W znaczniku `footer` znajduje się tekst oraz strukturalny element `nav`.

Muszę następnie dalej podzielić obszar `feature_area`, aby uzyskać dwa osobne komponenty: formularz i odnośniki. Wszystkie obrazy książek w obszarze `book_area article` zawarłem we własnych elementach, aby uzyskać kontekst pozycjonowania dla chmurki informacyjnej każdej z książek. Te dodatkowe elementy składają się na trzeci poziom hierarchii strukturalnej i oznaczone są fioletowymi prostokątami.

Tego planowania już wystarczy, by zacząć pisać kod, choć w ramach tworzenia strony konieczne może się okazać dodanie jednego czy dwóch elementów. Utworzę teraz wstępny kod, który odzwierciedla opisaną strukturę.

```
<div id="wrapper">
    poziom pierwszy
    poziom drugi
    poziom drugi
    poziom drugi
        <header>
            <section id="title">
                <!-- nagłówki h1 i h2 -->
            </section>
            <nav class="menu">
                <!-- menu nawigacyjne -->
            </nav>
            <form class="search">
                <!-- pole wyszukiwania -->
            </form>
        </header>
        <feature_area>
            <article>
                <!-- obraz -->
                <!-- opis -->
            </article>
            <article>
                <!-- obraz -->
                <!-- opis -->
            </article>
            <article>
                <!-- obraz -->
                <!-- opis -->
            </article>
            <article>
                <!-- obraz -->
                <!-- opis -->
            </article>
        </feature_area>
        <aside>
            <!-- pole logowania -->
            <!-- lista odnośników -->
        </aside>
    </div>
```

```
</form>
</header>
poziom pierwszy ━━━━━━ <section id="feature_area">
poziom drugi ━━━━━━ <article id="blog_leadoff">
                        <!-- treść bloga -->
                </article>
poziom drugi ━━━━━━ <aside>
poziom trzeci ━━━━━━ <form class="signin">
                        <!-- pole logowania -->
                </form>
poziom trzeci ━━━━━━ <nav>
                        <!-- odnośniki do wpisów -->
                </nav>
                </aside>
            </section>
poziom pierwszy ━━━━━━ <section id="book_area">
poziom drugi ━━━━━━ <article>
                        <div class="inner">
                            <!-- obrazy książek i obrócony tekst -->
                        </div>
                </article>
                <!-- cztery elementy article -->
            </section>
poziom pierwszy ━━━━━━ <footer>
                        <!-- tekst elementów footer i nav -->
                </footer>
poziom trzeci ━━━━━━ </div>
```

Jak widzisz, każdy poziom osadzonych pól layoutu odzwierciedlony jest w kodzie w postaci osadzonych elementów. Najbardziej odpowiednie elementy HTML można dobierać na bieżąco. Lista menu na przykład zdecydowanie powinna znaleźć się w elemencie `nav`.

Mając już ukończony kod strukturalny, określę ogólne ustawienia fonta i koloru tła strony dla elementu `body` oraz definiuję styl kontenera `wrapper`, by podać ogólną szerokość layoutu i wyśrodkować go na stronie.

```
body {  
    font-family:helvetica, arial, sans-serif;  
    background:#efefef;  
    margin:0;  
}  
  
wrapper {width:980px; margin:0 auto 20px;}
```

Mogę teraz zająć się kolejnymi elementami na stronie, nadając im treść i style CSS.

Stylizacja nagłówka

Zacznijmy od utworzenia kodu treści nagłówka `header`.

```
<header>  
  
<section id="title">  
    <h1>Strona o CSS</h1>  
    <h2>Blog i książki Charlesa Wyke-Smitha</h2>  
</section>  
  
<nav class="menu">  
    <ul>  
        <li class="choice1"><a href="#">Artykuły</a></li>  
        <li class="choice2"><a href="#">Książki</a></li>  
        <li class="choice3"><a href="#">Materiały</a></li>  
        <li class="choice4"><a href="#">Biblioteczka</a></li>  
        <li class="choice5"><a href="#">Kontakt</a></li>  
    </ul>  
</nav>
```

```
<form class="search" action="#" method="post">
    <label for="search">search</label>
    <input type="text" id="search" name="search"
        placeholder="szukaj" />
</form>
</header>
```

atribut for (o takiej samej wartości, jak identyfikator kontrolki) łączy oznaczenie z kontrolką

Znacznik `header` dzieli się na trzy części: tytuł, obszar wyszukiwania i wyśrodkowane menu. Zacznę od oznaczonego identyfikatorem `title` obszaru tytułowego.

Obszar tytułowy

Elementy `h1` i `h2` pozycjonuję bezwzględnie w górnym lewym rogu elementu `header`. Oto kod CSS:

```
header {
    kontekst pozycjonowania obszaru tytułowego i wyszukiwania
    ustalona wysokość elementu obejmującego elementy
    pozycjonowane bezwzględnie
    kolejność: lewy górny róg, prawy
    górnny, prawy dolny, lewy dolny
    ujemne oddalenie sprawia, że cień
    nie przekracza szerokości pola
    sprawia, że marginesy elementu
    i jego dzieci nie ulegają scaleniu
}
header section#title {
    odpowiednio duża szerokość
    pozwala zapobiec zawijaniu tekstu
    wystarczająco duża wysokość
    do objęcia dwóch wierszy tekstu
    pozycjonowanie w lewym górnym
    rogu}
```

position: relative;
height: 70px;
margin: 10px 0;
background: #fff;
border-radius: 20px 0px 20px 0px;
box-shadow: 0 12px 8px -9px #555;
padding: 1px;
position: absolute;
width: 300px;
height: 65px;
left: 0px;

```

    top:0;
}

header h1 {
    padding:9px 12px 0;
    font-family:'Lato', helvetica, sans-serif;
    font-weight:900;
    font-size:2.2em;
    line-height:1;
    letter-spacing:-.025em;
    color:#4eb8ea;
}

header h2 {
    padding:0px 12px;
    font-family:"Source Sans Pro", helvetica, sans-serif;
    font-weight:400;
    font-size:.9em; line-height:1;
    letter-spacing:-.025em;
    color:#333;
}

```

grubość, którą trzeba określić ————— dla pobieranych fontów

Strona o CSS

Blog i książki Charlesa Wyke Smitha

RYSUNEK 7.3. Elementy h1 i h2 są już wystylizowane. Tymczasowo włączone obramowania ukazują położenie elementów

Pierwsze, na co warto zwrócić uwagę w tym kodzie, to ustalona wysokość elementu `header`. Jak pokazałem w wielu poprzednich przykładach, zwykle warto pozwolić, by treść określała wysokość elementów strukturalnych, a wysokość strony zmieniała się wraz z dodawaniem treści. W tym przypadku, jako że `header` zawiera pozycjonowane bezwzględnie elementy, które nie rozpychają swoich rodziców, wysokość musiało określić sam. Zawartość elementu `header` nie będzie ulegać większym zmianom, jeśli w ogóle, więc jest mało prawdopodobne, że kiedykolwiek wyjdzie poza obszar zdefiniowany jego wysokością.

Zwróć uwagę na ciekawe zestawienie dwóch zaokrąglonych i dwóch zwykłych rogów, które nadałem elementowi `header` oraz wielu innym znacznikom na stronie. Ten dyskretny a wyrazisty efekt dodaje layoutowi niepowtarzalnego charakteru. W ramce „Zaokrąglone rogi” znajdziesz więcej szczegółów związanych z tworzeniem zaokrąglonych rogów pól elementów HTML.

Pozycję elementu `title` (oznaczonego kolorem czerwonym na **rysunku 7.3**) określiłem bezwzględnie, a następnie zdefiniowałem wielkość i dopełnienie jego elementów tekstowych oraz ustaliłem jego wysokość i szerokość, żeby te elementy obejmował.

Zauważ, że w tych nagłówkach używam fonta internetowego Lato z Google Web Fonts. Grubość wielu fontów internetowych określa się właściwością `font-weight`, inaczej niż w przypadku fontów zainstalowanych w systemie, gdzie dla tej właściwości można określić jedynie styl normalny i pogrubiony. Więcej na temat Google Web Fonts przeczytasz w rozdziale 4.

Cienie pól to kolejny charakterystyczny aspekt tego projektu. Są one zdefiniowane tak, by pojawiały się tylko przy dolnej krawędzi pola i miały od pola mniejszą szerokość. Tworzy to wrażenie, że pole unosi się nad stroną. Przykład znajdziesz w podświetlonym kodzie w powyższym przykładzie, a więcej na temat cieni pól przeczytasz w ramce „Cienie pól”.

Mając już gotowe pole tytułu, w podobny sposób umieścmy pole wyszukiwania po lewej stronie.

Formularz wyszukiwania

Zacznijmy od kodu:

```
<form class="search" action="#" method="post">  
    <label for="search">search</label>  
    <input type="text" id="search" name="search"  
        placeholder="szukaj" />  
</form>
```

Poniżej znajduje się kod CSS formularza, który pojawił się w przykładzie w poprzednim rozdziale. Jedyna istotna różnica dotyczy tego, w jaki sposób określone jest położenie formularza w nagłówku.

Zaokrąglone rogi

Tworzenie zaokrąglonych rogów, które były charakterystyczne dla designu Web 2.0 kilka lat temu, wymagało korzystania z rozbudowanego kodu JavaScript lub starannego rozmieszczania plików graficznych w zagnieżdżonych elementach `div`. Dziś wystarczy napisać jedną linijkę kodu CSS.

Podstawowa składnia wygląda następująco:

`border-radius:10px;`

W tym przypadku zaokrąglenie wszystkich czterech rogów ma wielkość 10 pikseli.

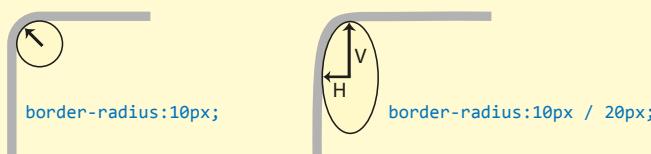
Można tutaj użyć standardowych deklaracji zbiorczych pola, z tym że nie obowiązuje kolejność właściwości `top`, `right`, `bottom`, `left` (które odnoszą się do krawędzi), lecz `top-left`, `top-right`, `bottom-right`, `bottom-left` (które odnoszą się do rogów).

Zauważ, że możesz określić zarówno poziomy, jak i pionowy promień zaokrąglenia

`border-radius:10px / 20px;`

Na rysunku 7.4 widać, że podane wartości określają promień koła lub elipsy, na której oparte jest zaokrąglenie.

`border-radius:10px;`



RYSUNEK 7.4. Ten diagram przedstawia rezultaty zastosowania dwóch powyższych przykładów kodu

Jeżeli chcesz nadać odmienne wartości promieni poziomych i pionowych każdemu rogowi, możesz napisać deklarację zbiorczą:

`border-radius:10px 6px 4px 12px / 20px 12px 8px 24px; /* cztery poziome promienie, cztery pionowe */`

Zauważ, że obramowanie nie musi być widoczne, by korzystać z zaokrąglonych rogów. Jak widać na przykładzie menu z tego rozdziału, kolor tła elementu wyświetla samo zaokrąglenie bez obramowania.

```

szerokość pozwalająca na objęcie poszerzonego pola → form.search {
położenie określone względem prawego górnego rogu nagłówka →   position:absolute; width:150px;
                                                               top:23px; right:20px;
                                                               }

tworzy miejsce, w którym pole → .search input {
może się poszerzyć w lewo →   float:right; width:70px;
                               padding:2px 0 3px 5px;
                               border-radius:10px 0px 10px 0px;
                               font-family:"Source Sans Pro", helvetica, sans-serif;

```

Cienie pól

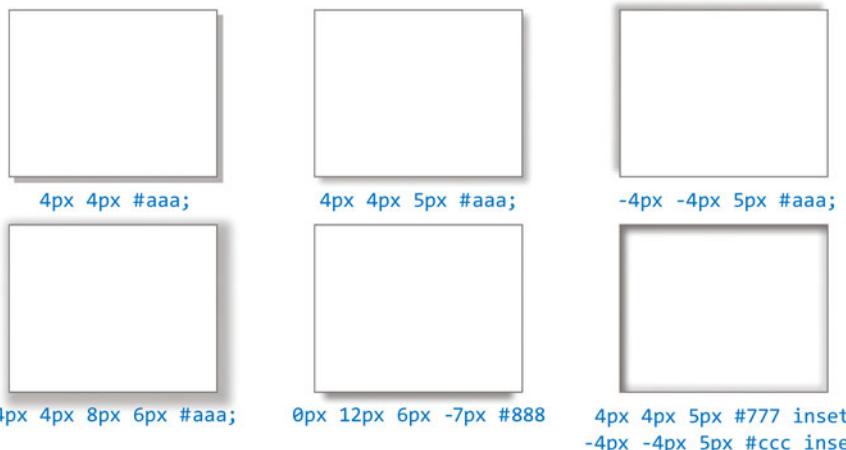
Cienie pól elementów HTML są kolejnym przykładem efektu, który — zanim pojawił się CSS3 — uzyskiwano przy użyciu różnych elementów graficznych, elementów `div`, co wymagało niemałej cierpliwości, a dziś można uzyskać dzięki jednej linijce kodu CSS.

Składnia podstawowej deklaracji wygląda następująco:

```
box-shadow:4px 4px 5px 8px #aaa inset;
```

Kolejność stylów jest następująca: przesunięcie w poziomie, przesunięcie w pionie, rozmycie, rozszerzenie, kolor, wpuszczenie cienia do wewnętrz pola (cień znajduje się domyślnie poza polem).

Musisz przynajmniej podać przesunięcie w poziomie i w pionie oraz kolor — uzyskujesz wtedy cień o wyraźnej krawędzi, o podanej szerokości i barwie. Nadając wartości ujemne ustawienniom przesunięcia w prawo i w dół, przesuwasz cień w lewo i w górę. Słowo kluczowe `inset` wpuszcza cień w pole. Można podać więcej niż jedną deklarację cienia, oddzielając je od siebie przecinkami. Na rysunku 7.5 widnieją różne przykłady tego, co można tą właściwością uzyskać.



RYSUNEK 7.5. Przy użyciu różnych wartości, zarówno dodatnich, jak i ujemnych, można uzyskać różne efekty cienia pola

```
font-weight:400;
font-size:1em;
color:#888;
outline:none;
-webkit-transition:2s width;
}
.search input:focus {width:140px;}
```

usuwa domyślne podświetlenie obramowania

tworzy animację zmiany wielkości pola (wymagane są tu także inne prefiksy)

element rozciąga się na taką szerokość po sfokusowaniu

```
.search label {display:none;}
form.search input {background-color:#fff;}
form.search input::-webkit-input-placeholder {color:#ccc;}
```

Strona o CSS

Blog i książki Charlesa Wyke-Smitha

poz. bezwzględne

RYSUNEK 7.6. Przejście zwiększa szerokość pola. Tymczasowo widać obramowanie formularza

Jak widać w wyróżnionym kodzie, określам szerokość elementu `form` i określam położenie jego prawej krawędzi względem prawej krawędzi elementu `header`. Jak wskazuje tymczasowo widoczne obramowanie na **rysunku 7.6**, kontener formularza jest wystarczająco duży, by obejmować rozszerzone pole. Nie będę się rozwodzić nad samym obszarem wyszukiwania, ponieważ omówiłem go już w poprzednim rozdziale, ale wspomnę, że tekst zastępczy — przynajmniej w przeglądarkach opartych na Webkit — można wystylizować inaczej niż tekst, który użytkownik podaje w polu, co widać w ostatniej linijce powyższego kodu.

Menu

 W celu zaoszczędzenia miejsca nie pokazuję osadzonych poziomów listy, które składają się na rozwijane menu. O tworzeniu takich menu przeczytasz w punkcie „Rozwijane menu” w rozdziale 6.

Rozmieściłem już elementy bezwzględnie przy obydwu krawędziach nagłówka, usuwając je ze struktury dokumentu. Mogę teraz umieścić między nimi wyśrodkowane menu. Oparte jest ono na kodzie CSS menu z rozdziału 6., z wyjątkiem kilku drobnych zmian.

```
<nav class="menu">
<ul>
  <li class="choice1"><a href="#">Artykuły</a></li>
  <li class="choice2"><a href="#">Książki</a></li>
  <!-- kolejne elementy menu -->
</ul>
</nav>
```

Kod składa się ze standardowej listy odnośników w elemencie `nav`, ale poszczególne odnośniki przypisałem do osobnych klas, aby nadać im różne kolory.

Oto kod CSS:

```
nav.menu {  
    margin:19px auto;  
    padding:0;  
    text-align:center;  
    font-size:.8em;  
}  
  
wyrównuje menu względem kontenera  
kontener ściśle obejmuje elementy listy  
rozkłada menu w poziomie  
usuwa domyślne punktory listy  
kontekst pozycjonowania osadzonej listy  
sprawia, że odnośnik wypełnia element li  
usuwa podkreślenie odnośnika  
zapobiega migotaniu tekstu pod koniec zmiany przezroczystości w przeglądarkach opartych na Webkit  
nav.menu > ul {display:inline-block;}  
nav.menu li {  
    float:left;  
    list-style-type:none;  
    position:relative;  
}  
  
nav.menu li a {  
    display:block;  
    padding:.25em .8em;  
    font-family:"Source Sans Pro", helvetica, sans-serif;  
    font-weight:600;  
    font-size:1.2em;  
    text-align:left;  
    color:#fff;  
}  
  
nav.menu li a {text-decoration:none;  
-webkit-font-smoothing:antialiased;  
}  
nav.menu li.choice1 a {background:#f58c21;}  
nav.menu li.choice2 a {background:#4eb8ea;}
```

```
nav.menu li.choice3 a {background:#d6e636;}  
nav.menu li.choice4 a {background:#ee4c98;}  
nav.menu li.choice5 a {background:#f58c21;}  
nav.menu li:hover > a {  
    color:#555;  
    border-color:#fff;  
    border:0;  
}  
nav.menu li:last-child a {border-bottom-right-radius:10px;}  
nav.menu li:first-child a {border-top-left-radius:10px;}
```



RYSUNEK 7.7. Menu jest teraz wyśrodkowane na stronie. Pozycjonowane bezwzględnie elementy tytułu i pola wyszukiwania są wyjęte poza strukturalny ciąg dokumentu, wobec czego element nav może być szeroki na całą stronę (co wskazuje tymczasowo widoczne obramowanie)

Jako że obszary tytułu i wyszukiwania są pozycjonowane bezwzględnie, są one wyjęte z ciągu strukturalnego dokumentu.

Element blokowy **nav** zachowuje się przez to tak, jakby nie były one obecne, i rozciąga się na całą szerokość swojego rodzica, **header** (rysunek 7.7). Mogę dzięki temu wyśrodkować menu na stronie. Posłużę się tym przykładem, aby bardziej szczegółowo omówić wyśrodkowywanie przy użyciu kodu CSS.

WYŚRODKOWYWANIE ELEMENTÓW O NIEOKREŚLONEJ SZEROKOŚCI

Wyśrodkowanie elementu zawartego w drugim elemencie może być trudne. Kiedy masz do czynienia z elementami pozycjonowanymi standardowo wartością **static**, możesz sprawić, by element spływał w lewo lub w prawo, albo użyć właściwości **text-align** z rodzicem, aby wyrównać element do lewej (**left**), prawej (**right**) bądź wyśrodkować go (**center**). Element można także wyśrodkować przy użyciu marginesów o wartości **auto**. Minusem tych technik jest konieczność ustalenia szerokości elementu, który chce się tak rozmieścić. Kiedy lista tworząca takie menu generowana jest dynamicznie na

podstawie bazy danych, czy nawet jeśli jej elementy będą modyfikowane ręcznie, to nie sposób przewidzieć szerokości wyśrodkowanego elementu i ją ustalić. Często otrzymuję e-maile z pytaniami, jak wyśrodkowywać menu w kontenerze, wobec czego omówię wyśrodkowywanie elementów o nieokreślonej szerokości.

Wartość `inline-block` jest dziwnym, hybrydowym ustawieniem właściwości `display`, które łączy w sobie zachowanie elementów liniowych i blokowych. Zachowanie blokowe elementu o takim ustawieniu polega na obsłudze marginesów i dopełnień oraz możliwości obejmowania innych elementów blokowych. Zachowanie liniowe z kolei polega na ścisłym okalaniu zawartości zamiast rozszerzania się na całą szerokość rodzica. Oznacza to, że szerokość elementu zawsze jest równa szerokości jego treści. Kolejną użyteczną cechą wartości `inline-block` jest to, że element obejmuje pływające elementy. Problem w tym, że taki element *nie obsługuje* jednej wartości marginesu — `auto` — która jest najprostszym sposobem na wyśrodkowanie elementu w kontenerze.

Rozwiązaniem jest nadanie właściwości `text-align:center` rodzicowi elementu, czyli `nav`, a następnie nadanie wyśrodkowywanemu elementowi właściwości `display:inline-block` — w tym przypadku elementu `ul` z elementami menu. To zestawienie pozwala na uzyskanie pożądanego rezultatu, czyli wyśrodkowanego w rodzinie elementu o nieokreślonej szerokości. Jak widać w pierwszych dwóch wyróżnionych wierszach podanego wcześniej kodu, tak właśnie zrobiłem. Menu jest teraz idealnie wyśrodkowane, gdyż rodzic `nav` ignoruje bezwzględnie pozycjonowane elementy znajdujące się po jego bokach, oraz rozciąga się na całą szerokość elementu `header`.

Aby zademonstrować, jak to się świetnie sprawdza, usunę jeden z elementów listy menu.



RYSUNEK 7.8. Menu pozostaje wyśrodkowane nawet po zmianie liczby zawartych w nim elementów

Na **rysunku 7.8** usunąłem ostatni element menu, które pozostaje idealnie wyśrodkowane. Jest to idealne rozwiązanie dla witryny z dynamicznie generowaną treścią, na której różni użytkownicy (np. za-rejestrowani i niezarejestrowani) otrzymują różne menu. Zauważ też, że zaokrąglonych rogów nie nadałem pierwszemu i piątemu elementowi menu, gdyż uniemożliwiłoby to wprowadzanie zmian w menu. W celu uzyskania bardziej praktycznego kodu CSS, jak widać w wyróżnionym kodzie, zaokrąglone rogi nadałem elementom **:first-child** i **:last-child**. Po usunięciu piątej pozycji menu czwarty element staje się ostatnim i otrzymuje zaokrąglony róg określony selektorem **:last-child**.

DODANIE ROZWIJANEGO MENU

W ramach dodania rozwijanego menu zaprezentuję kolejny przykład przejść w CSS.

```
nav.menu li ul {  
    ukrywa menu → opacity:0; visibility:hidden;  
    określa położenie względem → position:absolute;  
    szerokość rozwijanego menu → width:12em;  
    wyrównuje lewą krawędź → left:0;  
    podmenu w stosunku do rodzica  
    wyrównuje element w stosunku → top:100%;  
    do dolnej krawędzi rodzica  
    tworzy przejście → -webkit-transition:1s all;  
    → -moz-transition:1s all;  
    → transition:1s all;  
    → }  
  
nav.menu li:hover > ul {  
    obydwie właściwości podlegają → opacity:1; visibility:visible;  
    przejściu → }  
  
nav.menu li li {  
    niweluje odziedziczone płynwanie → float:none;  
    elementów — sprawia, że odnośniki  
    rozmieszczane są w pionie → }  
  
nav.menu li li:first-child a {border-radius:0;}  
nav.menu li li:last-child a {border-bottom-left-radius:10px;}  
/* kod dla przeglądarek, które nie obsługują przejść CSS */
```

```
.no-csstransitions nav.menu li ul {
    przesłania wersję z przejściem   visibility:visible;
    przesłania wersję z przejściem   opacity:1;
    ukrywa menu, jeśli przeglądarka
    nie obsługuje przejść CSS       display:none;
                                    }

wyświetla menu, kiedy kurSOR
znajduje się nad rodzicem      .no-csstransitions nav.menu li:hover > ul {display:block;}
```



RYSUNEK 7.9. Rozwijane menu dodane do menu głównego

Rozwijane menu tworzę takim samym kodem, jak w rozdziale 6. Żeby nie tłumaczyć wszystkiego od nowa, odsyłam Cię niniejszym do tego rozdziału i komentarzy do powyższego listingu. Chciałbym jednak zwrócić uwagę na trzy rzeczy obecne w tej wersji kodu.

Po pierwsze, zaokrąglone rogi są odziedziczone przez elementy rozwijanego menu. Zamiast zaokrąglić przeciwległe rogi, tak jak w menu głównym, chcę, by zaokrąglone były jedynie dolne rogi menu. Usuwam zatem odziedziczone zaokrąglenie w lewym górnym rogu i tworzę zaokrąglenie lewego dolnego rogu; zaokrąglenie prawa dolnego rogu pozostaje odziedziczone (**rysunek 7.9**).

Jak wskazują komentarze, przezroczystość menu ulega przejściu, żeby stawało się widoczne stopniowo. Za pierwszym podejściem nadałem jedynie właściwość `opacity` o początkowej wartości `0` (przezroczystość) i końcowej wartości `1` (pełna widoczność) elementowi, na którym znajduje się kurSOR. Dzięki temu menu rzeczywiście pojawiało się i znikało stopniowo, ale zawsze znajdowało się na swoim miejscu, nawet gdy nie było widoczne. Gdybym przesunął kurSOR w miejsce pod menu, rozwijane menu i tak się pojawiało, nawet jeśli kurSOR nie znajdował się nad odpowiednią pozycją w menu głównym. Spróbowałem następnie dodać właściwości `display:none` i `display:block`, aby całkowicie usunąć rozwijane menu, kiedy kurSOR nie znajdował się nad elementem menu. Wprawdzie rozwią-

załem w ten sposób problem, ale przez to rozwijane menu włączały się i wyłączały bez przejścia. Postanowiłem usunąć właściwość `display` i zamiast tego wyłączyć po najechaniu na element kursorem właściwość `visibility` podczas zmiany przezroczystości w ramach przejścia. W rezultacie rozwijane menu pojawiało się stopniowo, ale znikało bez przejścia. Wreszcie, po utworzeniu przejścia obejmującego zarówno `opacity`, jak i `visibility`, menu całkowicie znikało, gdy było przezroczyste, a zaczęło się pojawiać i znikać zgodnie z zamierzeniem, czyli stopniowo. Chcę zwyczajnie powiedzieć, że czasami trzeba eksperymentować, żeby osiągnąć pożądany efekt. Mam nadzieję, że kiedyś oszczędzisz sobie dzięki temu kilku godzin pracy. Zawsze do usług.

Po trzecie, skorzystałem z okazji, by pokazać skrypt Modernizr w działaniu i podać zapasowy kod CSS odpowiedzialny za działanie menu w przeglądarkach, które nie obsługują przejść CSS3. W takim przypadku Modernizr nadaje klasę `no-css-transitions` elementowi głównemu `html` podczas wczytywania strony. Używam tej klasy w selektorach reguł, których mają używać jedynie te przeglądarki, które nie obsługują przejść CSS. W takich regułach anuluję ustawienia `visibility` i `opacity`, które określają działanie menu w przeglądarkach obsługujących przejścia, podając do wyświetlania i ukrywania menu podstawowe reguły `display:none` i `display:block`, których użyłem przy menu w rozdziale 6.

Tym samym ukończyłem stylizację elementu `header`. Przejdźmy teraz do stylizacji obszaru obejmującego wpis blogowy, pole logowania i odnośniki do wpisów.

Wyśrodkowywanie w pionie

Poziome wyśrodkowywanie w CSS nie jest łatwe. Kiedy wyśrodkowujesz pojedynczy wiersz tekstu w obrębie elementu o stałej wysokości, np. 300 pikseli, właściwości `line-height` tekstu nadaj wartość równą wysokości kontenera:

```
text-align:center; /* wyśrodkowywanie w poziomie */  
line-height:300px; /* wyśrodkowywanie w pionie = wysokość kontenera */
```

Aby wyśrodkować pionowo pozostałe elementy, takie jak obrazy, właściwości `display` kontenera nadaj wartość `table-row`, a następnie nadaj mu działającą tylko z komórkami tabel właściwość `vertical-align` o wartości `middle`.

```
display:table-cell; /* uaktywnia zachowanie elementu jako tabeli */  
vertical-align:middle; /* wyśrodkowywanie w pionie */  
text-align:center; /* wyśrodkowywanie w poziomie */
```

Żadne z tych rozwiązań nie jest szczególnie eleganckie, ale w CSS nie ma konkretnych właściwości, które pozwalałyby na pozycjonowanie elementów w pionie.

Obszar treści

W głównym obszarze treści znajdzie się wprowadzenie do najnowszego wpisu, a na mniejszym obszarze po prawej umieszczone pole logowania i spis odnośników do najnowszych artykułów na blogu. Oto kod HTML obszaru `feature_area`.

```
<section id="feature_area">  
  <article id="blog_leadoff">  
    <div class="inner">  
      <h4>7 września 2012</h4>  
      <a href="#"><h3>Klasy CSS w jQuery</h3></a>  
        
      <p class="css_cols3">Sintus at neque in magna...</p>  
    </div>  
  </article>  
  <aside>  
    wymagany znacznik form |  
    kontener zbioru kontrollek |  
    oznaczenie tekstowe zbioru |  
    kontrollek  
    kontener pomagający przy stylizacji  
    kontrolki, oznaczenia i wskazówek |  
    atrybut for (o takiej samej wartości,  
    jak identyfikator kontrolki) łączy |  
    oznaczenie z kontrolką  
    wartość atrybutu text sprawia, |  
    że kontrolka wyświetlana jest  
    jako pole tekstowe  
    <form autocomplete="off" class="signin" action="process_form.php" method="post">  
      <fieldset>  
        <legend><span>Pobierz kody i aktualizacje</span>  
        </legend>  
        <section>  
          <label for="email">E-mail</label>  
          <input type="text" id="email" name="email" />  
        </section>  
        <section>  
          <label for="password">Hasło</label>
```

wpisywane znaki wyświetlane są —
jako punktory

przycisk zatwierdzający ——|

```
<input type="password" id="password"
       name="password" maxlength="20" />
</section>
<section>
  <input type="submit" value="Zapisz się" />
  <p class="signup">Nie jesteś zarejestrowany?
    <a href="#">
      Zrób to teraz!</a></p>
</section>
</fieldset>
</form>
<nav>
  <!-- odnośniki do wpisów -->
</nav>
</aside>
</section>
```

Element oznaczający tę sekcję jest kontenerem rozciągniętym na całą szerokość strony. Zawarte w nim elementy `article` i `aside` będą spływać na boki, zajmując miejsca obok siebie.

sprawia, że element obejmuje
pływające elementy dzieci —|

odstęp między nagłówkiem —| a obszarem treści

```
section#feature_area {
  overflow:hidden;
  margin:16px 0 0;
  padding:0 0 10px;
}
section#feature_area article {float:left; width:66%;}
section#feature_area aside {float:right; width:34%;}
```

W ten sposób tworzę dwie kolumny w kontenerze. Zauważ, że określiłem ich szerokość wartościami procentowymi, gdyż chcę, by stronę można było oglądać na różnych urządzeniach, m.in. tabletach i smartfonach. Wrócę do tego w kolejnym rozdziale. Utworzone kolumny mają zatem szerokość będącą określonym odsetkiem szerokości kontenera.

Wiem, że na tym etapie muszę osadzić `div` (który wyróżniłem w wyższym kodzie) w elemencie `article`, żeby utworzyć obramowanie wokół treści. Przyjrzyjmy się teraz stylom obszaru `article`.

kontener z zaokrąglonymi rogami i cieniem → `section#feature_area article .inner { padding:12px; background:#fff; border-radius:20px 0; box-shadow:0 12px 8px -9px #555; }`

odnośnik z nagłówka → `section#feature_area article a {text-decoration:none;}`

zdjęcie → `section#feature_area article img { float:left; padding:0 10px 10px 0; }`

data → `section#feature_area article h4 { font-family:"Source Sans Pro", helvetica, sans-serif; font-weight:400; font-size:1em; color:#f58c21; letter-spacing:-.025em; }`

nagłówek bloga → `section#feature_area article h3 { font-family:'Lato', helvetica, sans-serif; font-weight:700; font-size:1.75em; color:#555; margin:0 0 12px 0; }`

```
letter-spacing:-.05em;
}

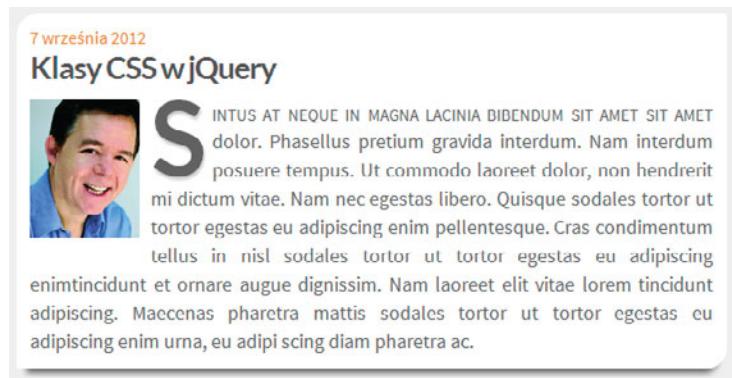
tekst główny —————| section#feature_area article#blog_leadoff p {
    font-family:"Source Sans Pro", helvetica, sans-serif;
    font-weight:400;
    font-size:1.1em;
    line-height:1.5em;
    color:#616161;
    text-align:justify;
}

inicjał —————| section#feature_area article#blog_leadoff p::first-letter {
    font-family:'Lato', helvetica, sans-serif;
    font-weight:700;
    font-size:4.5em;
    float:left;
    margin:.05em .05em 0 0;
    line-height:0.6;
    text-shadow:1px 3px 3px #ccc;
}

cień wyświetlany jest w IE10
i nowszych wersjach przeglądarki —————| kapitaliki w pierwszym wierszu —————| section#feature_area article#blog_leadoff p::first-line {
    font-variant:small-caps;
    font-size:1.2em;
}

prawa kolumna —————| section#feature_area aside {
    width:34%;
    float:right;
}
```

RYSUNEK 7.10. Wystylizowany element `article` w obszarze `feature_area`



The screenshot shows a web page section titled "Klasy CSS w jQuery" with a date "7 września 2012". It features a portrait of a smiling man in a blue shirt on the left. The main content starts with a large, bold letter "S". The text is justified and includes several paragraphs of placeholder Latin text.

Powyzsze style odnoszą się do kilku elementów, które już wcześniej omówiłem. Warto zwrócić uwagę na odnośnik `a`, obejmujący nagłówek `h3`. Umieszczanie elementów blokowych w elementach liniowych było kiedyś absolutnie niedopuszczalne, ale w HTML5 odnośnikiem można objąć dowolny element, dzięki czemu oczywiście łatwiej określać klikalność elementów.

Jak widać na rysunku 7.10, obraz spływa w lewo i jest oblany tekstem. Użyłem zestawienia inicjału z kapitalikami, które zaprezentowałem w rozdziale 4., aby urozmaicić oprawę i gładko przejść od nagłówka do tekstu. Inicjałowi nadałem także cień, aby wybijał się nieco ze strony. Zajmę się teraz stylizacją formularza.

Stylizacja pola logowania

Czytelników, którzy chcą pobrać kody z moich książek, proszę o rejestrację, abym mógł im wysyłać aktualności i utrzymać z nimi kontakt. Na stronie głównej znajduje się pole logowania z odnośnikiem do formularza rejestracyjnego dla nowych użytkowników. Formularz na stronie głównej składa się z takiego samego kodu, jak formularz omówiony w rozdziale 6. Oto jego kod HTML:

```
<form autocomplete="off" class="signin" action="process_form.php" method="post">
```

`<fieldset>`

oznaczenie tekstowe zbioru kontrolek

`<legend>Zapisz się, by pobrać kody i otrzymywać aktualizacje</legend>`

pole na adres e-mailowy

`<section>`

atribut `for` (o takiej samej wartości, jak identyfikator kontrolki) łączy oznaczenie z kontrolką

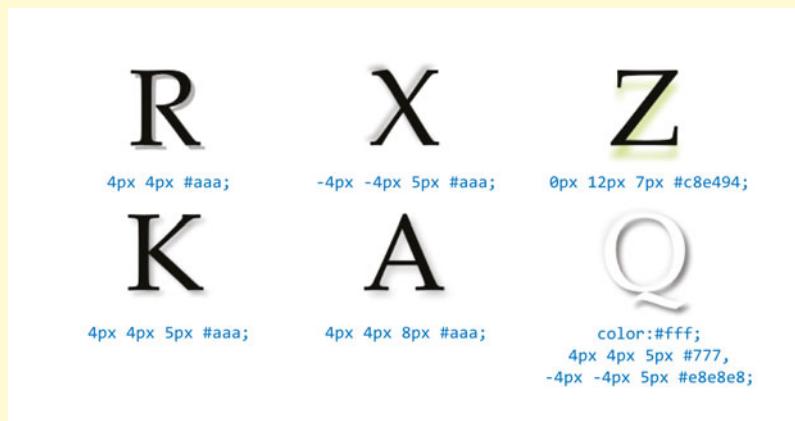
`<label for="email">E-mail</label>`

Cienie tekstu

Cienie tekstu są bardzo podobne do cieni pól, które omówiłem wcześniej w ramce „Cienie pól”. Oto składnia kodu, którym się je tworzy:

```
text-shadow:4px 4px 5px #aaa;
```

Kolejność stylów jest następująca: przesunięcie w poziomie, przesunięcie w pionie, rozmycie i kolor. W odróżnieniu od cieni pól, cienie tekstu nie mają ustawienia rozszerzenia. Musisz podać przynajmniej przesunięcie w poziomie i pionie oraz kolor, co tworzy cień o wyrazistej krawędzi, określonej szerokości i kolorze. Określając ujemne wartości przesunięcia w prawo i w dół, przesuwasz cień w lewo i w górę. Można podać więcej niż jedną deklarację cienia, oddzielając je od siebie przecinkami. Na **rysunku 7.11** widnieją różne przykłady tego, co można tą właściwością uzyskać. O bardziej zaawansowanym zastosowaniu cieni tekstu przeczytasz w moim e-booku *Visual Styling with CSS3*.



RYSUNEK 7.11. Przy użyciu różnych wartości, zarówno dodatnich, jak i ujemnych, można uzyskać różne efekty cienia tekstu

wartość atrybutu `text` sprawia, że kontrolka wyświetlna jest jako pole tekstowe

pole hasła

tekst ukryty, kiedy element nie przynależy do klasy `error`

przycisk zatwierdzający

```
<input type="text" id="email" name="email" />
</section>
```

```
<section>
```

```
    <label for="password">Hasło</label>
```

```
    <input type="password" id="password"
           name="password" maxlength="20" />
```

```
    <p class="direction">Błędna nazwa użytkownika lub
       hasło</p>
```

```
</section>
```

```
<section>
```

```
<input type="submit" value="Zapisz się" />
<p class="signup">Nie jesteś zarejestrowany?
<a href="#">
    Zrób to teraz!</a></p>
</section>
</fieldset>
</form>
```

Z przykładu formularza w rozdziale 6. pobrałem jedynie nieodzowne fragmenty kodu CSS, które dostosowałem do potrzeb tego formularza.

```
form.signin {
    width:19em;
    float:right;
    background:#fff;
    border-radius:10px 0 10px 0;
    box-shadow:0 12px 8px -9px #555;
}

usuwa domyślne obramowanie elementu fieldset
.signin fieldset {border:0; margin:10px 14px;}
.signin legend span {
    font-family:'Lato', helvetica, sans-serif;
    font-weight:700;
    font-size:1.3em; line-height:1.1em;
    color:#4eb8ea;
    letter-spacing:-.05em;
}
.signin section {
    overflow:hidden;
    padding:.25em 0;
}
.signin section label {
    font-family:"Source Sans Pro", helvetica, sans-serif;
```

ogólna szerokość formularza



width:19em;

usuwa domyślne obramowanie elementu fieldset



.signin fieldset {border:0; margin:10px 14px;}

sprawia, że element obejmuje kontrolkę i oznaczenie formularza



overflow:hidden;

odstęp między elementami



formularza

padding:.25em 0;

}

.signin section label {

font-family:"Source Sans Pro", helvetica, sans-serif;

```
    font-weight:400;
    float:left;
    width:5em;
    margin:.5em .3em 0 0;
    font-size:1em; line-height:1.1;
    color:#555;
}

.signin section input {
    float:right;
    width:10.5em;
    margin:.2em 0 0 .5em;
    padding:3px 10px 2px;
    color:#555;
    font-size:.8em;
    outline:none;
    border-radius:10px 0 10px 0;
}

input:-webkit-autofill {color:#fff !important;}
.signin section input[type=submit] {
    float:right;
    width:auto;
    margin:0 2px 3px 0;
    padding:0px 8px 3px;
    font-size:1em;
    font-weight:800;
    color:#fff;
    border:none;
    background-color:#d6e636;
```

```

        box-shadow:1px 1px 2px #888;
    }

tekst „Nie jesteś zarejestrowany?” — .signin section p {
    float:right;
    clear:both;
    margin:.2em 0 0;
    text-align:right;
    font-size:.8em;
    line-height:1;
    color:#555;
}

odnośnik do formularza rejestracyjnego — .signin section p a {color:#333;}
.signin section p a:hover {
    color:#777;
    text-decoration:none;
}

komunikat o błędzie — .signin section p.direction.error {
    display:block;
    color:#f00;
}

podświetla tekst wskazówk na czerwono, gdy dodana jest klasa error — .signin section p.direction {display:none;}
```

RYSUNEK 7.12. Wystylizowany formularz logowania z komunikatem o błędzie



Niezależnie od stopnia złożoności formularza, utworzenie go zawsze wymaga wielu linijek kodu. Kod w tym przykładzie w większości jest jednak dość prosty, a komentarze objaśniają jego najważniejsze fragmenty. Chciałbym jeszcze tylko zwrócić uwagę na komunikat o błędzie, który pozostaje ukryty, dopóki nie jest potrzebny (**rysунek 7.12**). Wystarczy dodać klasę `error` do elementu `p` (który już jest przypisany do klasy `direction`), aby komunikat się pojawił. Tym niemniej założenie jest takie, że ta klasa ma być w razie potrzeby dodawana automatycznie przez skrypt służący do walidacji danych formularza. Jako osoba odpowiedzialna za stworzenie interfejsu użytkownika, odpowiadasz za umieszczenie na stronie ukrytego zwykle elementu HTML z informacją o błędzie i opracowanie kodu CSS służącego do jego wyświetlenia — zadaniem programistów jest rozpracowanie, jak i kiedy nadać klasę przywołującą kod CSS odpowiedzialny za wyświetlenie komunikatu.

Odbośniki do wpisów

Pod formularzem znajdują się odnośniki do wpisów. Jak zwykle zamieściłem je w nieuporządkowanej liście.

```
<nav>
  <h3>Ostatnie wpisy</h3>
  <ul>
    <li><a href="#">Z-index — problemy się nawarstwiają</a></li>
    <li><a href="#">Jak używać box-image</a></li>
    <li><a href="#">Cienie w CSS3</a></li>
  </ul>
</nav>
```

A oto CSS:

```
section#feature_area nav {  
    width:19em;  
    float:right;  
    margin:15px 0 0;  
    padding:.6em 0em .75em;  
    background:#fff;  
    border-radius:10px 0 10px 0;
```

```
    box-shadow:0 12px 8px -9px #555;
}

#feature_area nav h3 {
    pozioma przestrzeń na tytuł ——— padding:0 14px;
    kontekst pozycjonowania punktorów ——— font-family:'Lato', helvetica, sans-serif;
    autorskie punktory ——— font-weight:700;
    pusty ciąg tekstowy jako treść ——— font-size:1.3em;
    położenie punktora ——— text-align:left;
    kształt punktora ——— color:#aaa;
    wielkość punktora ——— letter-spacing:-.05em;
}

#feature_area nav ul {margin:0 0 0 20px;}
#feature_area nav li {
    kontekst pozycjonowania punktorów ——— position:relative;
    odnośnik zajmuje całą szerokość kontenera ——— list-style-type:none;
}

#feature_area nav li::before {
    pozycjonowanie względem elementów listy ——— content: "";
    położenie punktora ——— position:absolute;
    kolor punktora ——— height:10px; width:10px;
    ksztalt punktora ——— left:12px; top:12px;
    wielkość punktora ——— border-radius:5px 0 5px 0;
    pusty ciąg tekstowy jako treść ——— background-color:#d6e636;
    autorskie punktory ——— box-shadow:1px 1px 2px #888;
}

#feature_area nav li a {
    odnośnik zajmuje całą szerokość kontenera ——— display:block;
```

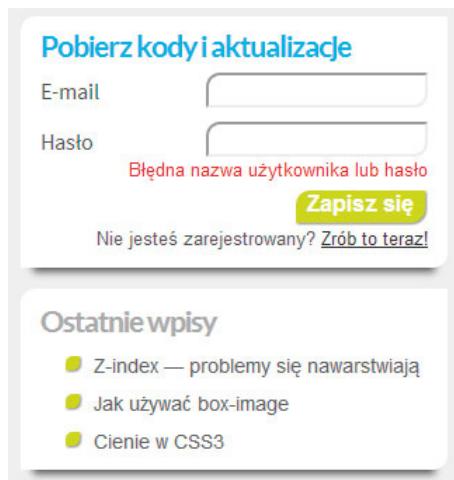
usuwa domyślne podkreślenie —→

```
text-decoration:none;
font-size:.9em;
color:#616161;
}
```

```
#feature_area nav li a:hover {color:#000;}
```

Wystylizowany obszar odnośników znajduje się bezpośrednio pod obszarem pola logowania. Obydwa obszary zawarte są w elemencie **aside**, który pływa obok elementu **article** (**rysunek 7.13**).

RYSUNEK 7.13. Ukończony element aside z odnośnikami do wpisów



Sprawiając, by element **nav** spływał tak samo jak znajdujący się nad nim element **form**, zyskuję możliwość umieszczenia go bezpośrednio pod formularzem. O ile element ten jest wystylizowany jak standartowa lista, to jego punktory są ciekawe. Chciałem, żeby odzwierciedlały motyw „dwóch zaokrąglonych, dwóch zwyczajnych” rogów, którym charakteryzuje się oprawa graficzna strony. Zamiast więc tworzyć osobny obraz punktora, użyłem pseudoelementu **::before** do utworzenia 10-pikselowego, kwadratowego elementu i zaokrąglenia jego rogów. Drobny, jednopikselowy cień wybija nieco punktory ze strony.

Obszar książek

Okładki czterech książek wyświetlane są w rzędzie u dołu strony. Na tym obszarze mamy do czynienia z kilkoma interesującymi elementami, takimi jak chmurki i obrócony tekst. Oto kod HTML:

```
obrócony tekst ————— | <section id="book_area">  
wskazówka ————— | <!-- poniższy element powtórzony jest jeszcze trzykrotnie -->  
                        <article class="left">  
                            <div class="inner">  
                                <h3>HTML5 + CSS3</h3>  
                                  
                                <aside>  
  
                                    <ol>  
                                        <li><a href="#">Pobierz kody</a></li>  
                                        <li><a href="#">Spis treści</a></li>  
                                        <li><a href="#">Kup książkę</a></li>  
                                    </ol>  
                                </aside>  
                            </div>  
                        </article>  
                        <!-- koniec powtórzeń kodu ramek z książkami -->  
                    </section>
```

Powyższy fragment kodu jest bardzo prosty. Jak widać w komenczu, powtórzony jest on jeszcze trzy razy, odpowiednio dla kolejnych książek. Przyjrzymy się teraz powiązanemu kodowi CSS. Zaczynam od utworzenia układu obszaru z książkami i obróconego tekstu, a następnie zajmuję się chmurkami.

```
element o pełnej szerokości ————— | section#book_area {  
odstęp nad i pod elementem ————— |     clear:both;  
                                         border-radius:20px 0px 20px 0px;  
                                         border:1px solid #f58c21;  
                                         margin:8px 0 16px;  
                                         overflow:hidden;  
                                         }
```

cztery kolumny na książki

```
#book_area article {
    float:left;
    width:25%;
    padding:10px 0;
    background:none;
}
```

kontener obejmujący książki

```
#book_area article .inner {
```

kontekst pozycjonowania chmurek

```
position:relative;
```

obejmuje każdą książkę

```
width:140px;
```

wysrodkowuje każdą książkę

```
margin:0 auto;
```

```
}
```

obrócony tekst

```
#book_area .inner h3 {
    position:absolute;
    width:160px;
    left:112%; bottom:5px;
    transform:rotate(-90deg);
    transform-origin:left bottom;
    color:#ccc;
    font-family:'Lato', helvetica, sans-serif;
    font-weight:900;
    font-size:1.4em;
    text-align:left;
}
```

inne przesunięcie

dla węższej okładki

```
#book_area article.right:last-child h3 {left:85%;}
```

cień pod każdą książką

```
#book_area article img {box-shadow:0 12px 8px -9px #555;}
```

W elemencie **section** o pełnej szerokości (#book_area) zamieściłem cztery płyniące elementy **article**, każdy o szerokości równej 25%.

W każdym z nich wysrodkowałem ustalonej szerokości **div** klasy

inner, zawierający obraz książki. Uzyskałem w ten sposób rząd

okładek z ładnymi odstępami (**rysunek 7.14**). Element **aside**, który

przekształcić w chmurkę, jest obecnie ukryty.



RYSUNEK 7.14. Każda książka w tym obszarze oznaczona jest skrótnym opisem, obróconym do pionu i zamieszczonym przy jej prawej krawędzi

Położenie obróconego tekstu określone jest dwiema funkcjami właściwości CSS3 `transform`. Pierwsza z nich, `transform-origin`, wskazuje lewy dolny róg pola elementu `h3` jako punkt początkowy przejścia. Punkt początkowy działa tak, jakbyś w jego miejscu wbił w element szpilkę. Obróciłem następnie element `h3` o dziewięćdziesiąt stopni przy użyciu funkcji `rotate` właściwości `transform`, a na koniec przesunąłem go o pięć pikseli w góre, by dostosować jego położenie. Krótkie omówienie właściwości `transform` znajdziesz w ramce „Przekształcenia w CSS3”, a więcej na ten temat przeczytasz w moim e-booku *Visual Stylin' with CSS*.

Nadszedł czas, by dodać chmurki do książek. Chmurkę przedstawioną w rozdziale 6. nieco rozwinąłem na dwa sposoby. Po pierwsze, chmurki książek znajdujących się w prawej połowie strony będą wyświetlane po lewej stronie okładek, aby nie były ucinane przez okno przeglądarki. Po drugie, strzałkę przylegającą do chmurki wystylizowałem tak, by uzyskać wrażenie, że ona także jest objęta obramowaniem pola, a tym samym jest jego częścią. Jak to często bywa z takimi pozornie drobnymi, lecz istotnymi szczegółami, uzyskanie ich wymaga dużej ilości kodu.

W podanym powyżej kodzie widać, że każdemu elementowi `article` z obrazami książek nadałem klasy `left` i `right`, aby móc im przypisać kod CSS odnoszący się do rozmieszczenia chmurek oraz ich strzałek po lewej lub prawej stronie. A oto kod!

tutaj zaczynają się style

wspólne dla wszystkich chmurek — #book_area article aside {

ukrywa chmurki — display:none;

pozycjonowanie względem
elementu div klasy inner,
który otacza obraz — position:absolute;
z-index:2;

szerokość chmurki — width:200px;

Przekształcenia w CSS3

Jeżeli pracowałeś z edytorami graficznymi takimi jak Adobe Illustrator lub Adobe Fireworks, to prawdopodobnie już obracałeś, skalowałeś i pochylałeś tekst lub innego rodzaju elementy. Teraz możesz uzyskać te same efekty w przeglądarce, używając przekształceń CSS3 (które widać na [rysunku 7.15](#)).

Do tworzenia przekształceń w CSS3 służą dwie właściwości: `transform` i `transform-origin`. Zaczniemy od omówienia właściwości `transform`.

Właściwość `transform` opiera się w działaniu nie na zwykłych wartościach, ale na funkcjach. Funkcje pozwalają Ci na podanie rodzaju przekształcenia oraz wartości, z których ma być obliczona.

Format przekształcenia wygląda następująco: `transform: nazwaFunkcji(wartość liczbową lub x, y)`.

Oto funkcje przekształceń:

- `scale` — powiększa lub zmniejsza element. Wartości większe niż 1 powiększają element, a mniejsze zmniejszają go. Przykład: `transform:scale(1.5)`.
- `rotate` — obraca element o podaną liczbę stopni. Wartości dodatnie obracają go zgodnie z ruchem wskazówek zegara, a ujemne w przeciwną stronę. Przykład: `transform:rotate(-30deg)`.
- `skew` — przechyla element względem jego osi x lub y. Kiedy podana jest jedna wartość, przechylenie względem osi y nie zachodzi. Przykład: `transform:skew(5deg, 50deg)`.
- `translate` — przesuwa obiekt o podaną odległość po jego osi x lub y. Przypomina to pozycjonowanie względne, ponieważ miejsce pierwotnie zajęte przez element nie znika. Przykład: `transform:translate(-50px, 20px)`.

Właściwość `transform-origin` określa punkt, względem którego element ma zostać przekształcony. Domyślnie jest to punkt pośrodku elementu w poziomie i pionie, więc jeśli obróciś element, to będzie się on zachowywał, jakby wbito w jego środek szpilkę, i obracał się wokół niej. Właściwością `transform-origin` możesz wybrać inny punkt w elemencie przy użyciu słów kluczowych (`top`, `right` itd.) albo podać dodatnie lub ujemne wartości liczbowe. W ten sposób punkt przekształcenia można umieścić nawet poza polem elementu.



RYSUNEK 7.15. Oto kilka przykładów przekształceń

```
background:#fff;
przestrzeń wokół treści chmurki → padding:10px 2px 5px;
border:2px solid #f58c21;
border-radius:10px 0px 10px 0px;
box-shadow:4px 4px 16px #555;
color:#555;
font-family:"Source Sans Pro", helvetica, sans-serif;
font-size:.8em;
line-height:1.5em;
}

wyświetla chmurkę, kiedy
kursor znajduje się nad książką → #book_area article:hover aside {display:block; }

#book_area article aside li {
pionowe odstęp między
elementami listy i dopełnienie → padding:.25em 0 .75em 1em;
z lewej
usuwa domyślne punktory → list-style-type:none;
elementów listy
line-height:1.2em;
}

odnośniki tekstowe → #book_area article aside li a {
text-decoration:none;
font-size:1.2em;
color:#616161;
}

podświetla odnośniki, nad którymi → #book_area article aside li a:hover {
znajduje się kurSOR
color:#333;
}

koniec stylów wspólnych
dla wszystkich chmurek → }

dwie chmurki książek
znajdujących się po lewej → #book_area article.left aside {
umieszcza chmurkę po prawej → left:84%;
stronie obrazów znajdujących
po lewej
top:14px;
}
```

dwie chmurki książek znajdujących się po prawej — #book_area article.right aside {
umieszcza chmurkę po lewej — right:84%;
stronie obrazów znajdujących się po prawej — top:14px;
}
}

pole pomarańczowego trójkąta — #book_area article aside::after {
pusty ciąg tekstowy — wymagane — content:"";
jest podanie jakiejkolwiek treści — position:absolute;
położenie określone względem chmurki — top:33px;
border:12px solid;
height:0px; width:0px;
}
}

zmniejsza pole, by stworzyć trójkąt — #book_area article.left aside::after {
określa położenie i kolor trójkątów — right:100%;
chmurek książek, które znajdują się po lewej — border-color:transparent #f58c21 transparent transparent;
}
}
określa położenie i kolor trójkątów — #book_area article.right aside::after {
chmurek książek, które znajdują się po prawej — left:100%;
border-color:transparent transparent transparent #f58c21;
}
}
pole białego trójkąta — #book_area article aside::before {
pusty ciąg tekstowy — wymagane — content:"";
jest podanie jakiejkolwiek treści — position:absolute;
położenie określone względem chmurki — top:37px;
border:8px solid;
height:0px; width:0px;
z-index:100;
}
}

styl, położenie i kolor białego trójkąta chmurek książek, które znajdują się po lewej

```
#book_area article.left aside::before {  
    right:100%;  
    border-color:transparent white transparent transparent;  
}
```

styl, położenie i kolor białego trójkąta chmurek książek, które znajdują się po prawej

```
#book_area article.right aside::before {  
    left:100%;  
    border-color:transparent transparent transparent white;  
}
```

Chmurka jest elementem pozycjonowanym bezwzględnie w stosunku do elementu `div` klasy `inner`, który zawiera książkę. W przykładzie w rozdziale 6. z boku chmurki zwyczajnie umieściłem czerwony trójkąt (**rysunek 6.24**). W tym przykładzie znacznie ulepszyłem ten efekt; trójkąt wygląda, jakby wybijał się z chmurki i był objęty jej obramowaniem (**rysunek 7.16**). Uzyskałem ten rezultat, tworząc nieco mniejszy, biały (czyli takiego samego koloru, jak tło chmurki) trójkąt nad pomarańczowym trójkątem i wyrównując je w pionie. Pomarańczowy trójkąt służy zatem do uzyskania efektu obramowania. Biały trójkąt utworzyłem pseudoelementem `::before`, a pomarańczowy pseudoelementem `::after`. Do precyzyjnego wyrównania ich położenia użyłem pozycjonowania bezwzględnego oraz właściwości `z-index`. Rezultat jest bardzo estetyczny i łatwo uwierzyć, że chmurka i trójkąt stanowią razem jeden element.

Trudne może tu jednak być rozeznanie się w tym, co właściwie znajduje się z lewej, a co z prawej. Chmurki dwóch książek po lewej są na przykład zamieszczone po prawej stronie obrazów, a ich trójkąty z kolei znajdują się po ich lewej stronie. W przypadku dwóch książek z prawej położenie elementów jest odwrócone.



RYSUNEK 7.16. W chmurkach przylegających do każdej książki znajdują się odnośniki do dodatkowych informacji; zastosowanie chmurek pozwala na zachowanie miejsca

Pracę od ogółu do szczegółu, o której wcześniej wspomniałem, widać tutaj w działaniu. Jak widać w komentarzach do powyższego kodu, najpierw podałem kod CSS odnoszący się do wszystkich chmurek, czyli style określające ich wielkość, dopełnienia, kolor obramowań i treść. Dalej podałem kod służący do rozmieszczenia dwóch chmurek po prawej stronie pierwszej i drugiej książki oraz dwóch chmurek po lewej stronie trzeciej i czwartej książki. Dalej znajdują się style tworzące trójkąty, które występują przy każdej chmurce. Na końcu podaję style trójkątów chmurek dwóch książek po lewej oraz dwóch książek po prawej.

Choć trzeba było dużo wytłumaczyć, powyższy kod jest uporządkowany logicznie i nie występuje w nim zjawisko powtarzania się całych zbiorów stylów dla każdej chmurki. Warto poświęcić nieco czasu na przestudiowanie lub zrekonstruowanie tego przykładu, aby zyskać lepsze zrozumienie tej struktury. Ułatwi Ci to kodowanie wielu elementów o podobnych, ale nieco odmiennych stylach i zaznaczeniach. Przede wszystkim, takie podejście pozwala na uzyskanie kodu zrozumiałego i łatwego w obsłudze dla Ciebie i innych.

Zakończmy tworzenie strony, dodając stopkę.

Stopka

Stopka strony jest dobrym miejscem na informację o autorze i odnośniki do materiałów związanych ze sprawami formalnymi, takich jak zrzeczenie się odpowiedzialności, regulamin, informacje kontaktowe, polityka prywatności i informacje o prawach autorskich. Oto kod HTML:

`<div><div>Strona internetowa Szkoły Podstawowej im. Jana III Sobieskiego w Gostyniu
Zadanie 7
Autorka: Anna Kowalska
Data: 15.05.2023</div></div>`

```
<footer>
  <p>Szablon CSS z książki <a href="http://www.
  stylinwithcss.com">
    <em>CSS. Witryny internetowe sztyte na miarę, wydanie
    trzecie</em>
  </a> Charlesa Wyke-Smitha</p>
  <nav>
    <ul>
      <li><a href="#">Polityka prywatności</a></li>
      <li><a href="#">Kontakt</a></li>
    </ul>
  </nav>
</footer>
```

Oto kod CSS:

```
footer {
  odstęp pod i nad elementem ————— padding:.5em 0 .35em 0;
  wyśrodkowuje treść ————— text-align:center;
  border-radius:10px 0px 10px 0px;
  background:#fff;
  box-shadow:0 12px 8px -9px #555;
}

stylizacja tekstu ————— footer p {
  font-family:"Source Sans Pro", helvetica, sans-serif;
  font-weight:400;
  font-size:.85em;
  letter-spacing:-.05em;
  color:#555;
}

odnośnik w tekście ————— footer p a {
  font-family:"Source Sans Pro", helvetica, sans-serif;
  font-style:italic;
  font-weight:700;
  font-size:1em;
  color:#4eb8ea;
```

```
text-decoration:none;
}
footer p a:hover {
color:#777;
}
lista odnośników ————— footer ul {
sprawia, że pole elementu ścisłe ————— display:inline-block;
obejmuje listę margin:4px 0 0;
}
footer li {
usuwa domyślne punktory ————— list-style-type:none;
rozmiщуca elementy ————— float:left;
listy poziomo font-family:"Source Sans Pro", helvetica, sans-serif;
font-weight:400;
font-size:.85em;
}
footer li + li a {
linie oddzielające odnośniki ————— border-left:1px solid #ccc;
}
footer li a {
usuwa domyślne podkreślenie ————— text-decoration:none;
odnośników color:#aaa;
}
odstępy między odnośnikami ————— padding:0 5px;
}
footer a:hover {
color:#777;
}
```

RYSUNEK 7.17. W wystylizowanej stopce znajduje się element tekstowy i lista

Nie ma tu niczego, czego byś wcześniej nie widział. Treść została wyśrodkowana właściwością `text-align:center` ([rysunek 7.17](#)). Tę właściwość dziedziczy akapit, którego tekst zostaje w nim wyśrodkowany. Zauważ, że właściwość `text-align:center` standardowo nie wyśrodkowuje listy odnośników w ten sposób, ponieważ elementy blokowe standardowo zajmują całą szerokość kontenera. Element `ul` jednak ściśle obejmuje elementy `li`, ponieważ nadałem mu właściwość `display:block-inline`. W ten sposób zyskuje on określona szerokość i jest wyśrodkowywany ustawieniem `text-align:center`. Jak być może pamiętasz z kodu stylizującego menu, wykorzystanie właściwości `display:block-inline` sprawia, że szerokość pozostaje płynna, dzięki czemu lista będzie wyśrodkowywana także po dodaniu do niej lub odjęciu od niej elementów. Zauważ, że zastosowanie marginów o wartości `auto` zamiast właściwości `text-align:center` wyśrodkowałoby listę w równie dobrym stopniu.

Podsumowanie

Niniejszym kończymy rozdział, a tym samym tworzenie layoutu. Mam nadzieję, że przekonałeś się, że rozpoczęcie pracy od utworzenia struktury przy pomocy technik zaprezentowanych w rozdziale 5., a następnie dodawanie do strony komponentów opisanych w rozdziale 6. pozwala na szybkie tworzenie pełnoprawnych witryn.

Nad przykładową stroną będę pracował dalej w kolejnym rozdziale, ponieważ współcześnie witryny internetowe nie są wyświetlane jedynie w formie szerokich layoutów pasujących do dużych monitorów takich jak ten, na którym pracuję. Twoje strony muszą być skalowalne i reagować na wielkość ekranów urządzeń, na jakich się je ogląda. Utworzenie kilku layoutów o różnych wymiarach i staranie się, by właściwy layout trafił na właściwy ekran, nie jest dobrym pomysłem. Zamiast tego lepiej umieć tworzyć strony, które reagują na warunki zewnętrzne i wczytują odpowiedni kod CSS do stylizacji kodu HTML w taki sposób, by layout pasował do urządzenia, na którym się go wyświetla. W ostatnim rozdziale tej książki zademonstruję tworzenie skalowalnych witryn.

ROZDZIAŁ 8

Projektowanie skalowalne

W tym rozdziale przekształczę wersję strony przeznaczoną do wyświetlania na komputerze w stronę odpowiednią dla mniejszych ekranów, ale niemożliwie można powiedzieć o projektowaniu z myślą przede wszystkim o urządzeniach przenośnych. Rzuć okiem na artykuł Luke'a Wroblewskiego na stronie <http://www.lukew.com/ff/entry.asp?933> oraz na jego książkę Mobile First.

WSPÓŁCZESNE LAYOUTY STRON INTERNETOWYCH MUSZĄ

BYĆ ELASTYCZNE i reagować na różne środowiska, w jakich są wyświetlane. Zapewnienie użytkownikowi pozytywnego doświadczenia przy styczności z witryną na dużym monitorze wymaga czegoś zupełnie innego, niż gdy stronę przegląda się na telefonie komórkowym. Wielokolumnowy layout może się sprawdzać na dużym monitorze, ale na telefonie kolumny byłyby tak wąskie, że nie dałoby się ich odczytać. Layout z pojedynczą, „sceryjną” kolumną jest jedynym praktycznym rozwiązaniem, gdyż użytkownik może intuicyjnie i z łatwością przewijać ekran palcem podczas lektury.

Współcześnie, dzięki funkcji zapytań medialnych, umożliwiającej sprawdzenie typu medium, łatwo odczytać wymiary ekranu urządzenia użytkownika, aby podać zastępczy lub dodatkowy kod CSS. Tym samym dostarcza się użytkownikowi stronę zoptymalizowaną pod kątem danego rodzaju wyświetlacza. Tworzenie witryn, które są świadome medium, na jakim są przeglądane, nazywamy projektowaniem **skalowalnym** (ang. *Responsive Web Design – RWD*).

W tym rozdziale wrócę do strony, którą stworzyłem w rozdziale 7. z myślą o oglądaniu w przeglądarce komputerowej, i pokażę Ci kolejne kroki optymalizowania jej pod kątem wyświetlania na coraz mniejszych urządzeniach.

Duże layouty na małych urządzeniach

W poniższych przykładach korzystam z iPada i iPhona, ale opisane przeze mnie koncepcje w takim samym stopniu odnoszą się do wszelkich innych tabletów i smartfonów. Zaczniemy od obejrzenia strony z rozdziału 7. (o ustalonej szerokości 980 pikseli) na mniejszych urządzeniach, takich jak tablety i smartfony.



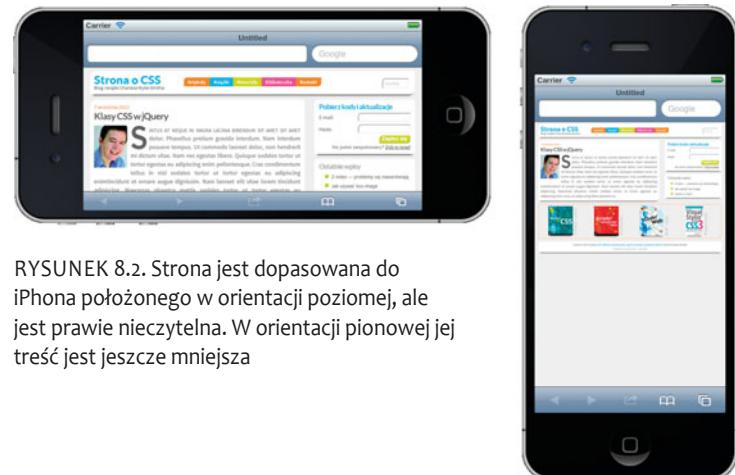
Wymiary ekranów oraz komponentów interfejsów iPada i iPhone'a można znaleźć na stronie <http://upstageapp.com/resources>.

Ekran iPada ma wymiary 1024×768, a ponieważ layout naszej strony ma szerokość 980 pikseli, jest on bardzo dobrze przystosowany do wyświetlania na 1024-pikselowym iPadzie w orientacji poziomej (**rysunek 8.1**). Kiedy jednak obracam iPada do orientacji pionowej, layout nie wypełnia ekranu.



RYSUNEK 8.1. Strona internetowa z rozdziału 7. jest czytelna i dobrze dopasowana do poziomej orientacji iPada, ale w orientacji pionowej jest dość mała i nie pasuje do wymiarów ekranu

Obejrzyjmy teraz stronę na iPhone'u (**rysunek 8.2**).



RYSUNEK 8.2. Strona jest dopasowana do iPhone'a położonego w orientacji poziomej, ale jest prawie nieczytelna. W orientacji pionowej jej treść jest jeszcze mniejsza

Jak widzisz, iPad i iPhone automatycznie dostosowują rozmiar strony do ekranu, ale jej layout zwyczajnie nie sprawdza się na małym ekranie iPhone'a. Konkretnie chodzi o tekst, który jest za mały.

Aspekty projektowania skalowalnego

Istnieją trzy kluczowe aspekty projektowania skalowalnego:

- **Zapytania medialne** — są to elementy CSS, które umożliwiają dostarczanie wybranych reguł CSS przeglądarkom na podstawie ich właściwości, czyli na ogólną szerokość ekranu lub przeglądarki.
- **Płynne layouty** — określając ogólną szerokość strony jednostkami względnymi w rodzaju em i wartości procentowych, pozwalasz layoutowi na zmianę skali w zależności od wielkości ekranu urządzenia.
- **Elastyczne obrazy** — nadając obrazom wielkości określone jednostkami względnymi, zapobiegasz sytuacjom, w których przerastałyby swoje kontenery.

Te koncepcje przedstawił pierwotnie Ethan Marcotte w A List Apart, w swoim fundamentalnym artykule z maja 2010 roku; artykuł znajdziesz na stronie <http://www.alistapart.com/articles/responsive-web-design>.

Użytkownik musiałby użyć gestu rozciągnięcia, aby powiększyć stronę na tyle, by tekst stał się czytelny; wtedy jednak w danej chwili widziałby jedynie mały fragment strony. W orientacji pionowej, która jest najbardziej naturalnym sposobem trzymania smartfona, strona jest jeszcze mniejsza.

Oczywisty wniosek jest taki, że uniwersalny rozmiar layoutu nie istnieje. Potrzebny jest wobec tego sposób na to, by strona wykrywała wielkość urządzenia, na którym jest wyświetlana, i stosownie do tego zmieniała swój layout — innymi słowy, musi być skalowalna. Przyjrzymy się, czego potrzeba do uzyskania takiej funkcjonalności. Pierwszym krokiem jest zapoznanie się z zapytaniami medialnymi.

Zapytania medialne



Urządzenia korzystające z Androida występują w wielu rozmiarach. Na stronie <http://pugetworks.com/blog/2011/04/css-media-queries-for-targeting-different-mobile-devices> znajdziesz zapytania medialne odnoszące się do urządzeń korzystających z iOS, Androida i Windowsa.

Urządzenia korzystające z Androida występują w wielu rozmiarach. Na stronie <http://pugetworks.com/blog/2011/04/css-media-queries-for-targeting-different-mobile-devices>

Zapytania medialne (ang. *media queries*) są kontenerami kodu CSS, który jest wykorzystywany tylko wtedy, kiedy spełnione są pewne warunki, np. takie, że strona jest w danej chwili drukowana albo wyświetlna na ekranie określonego typu lub wielkości. Zapytania medialne występują w dwóch postaciach: jako reguła `@media` lub jako atrybut `media` znacznika `tag`.

Reguła `@media`

Pierwszy format, reguła `@media`, pozwala na umieszczenie zapytania medialnego w kodzie arkusza CSS lub znaczniku `style`:

```
@media print {  
    nav {  
        display:none;  
    }  
}
```

Powyzsza reguła deklaruje, że przy drukowaniu strony element `nav` ma nie być widoczny.

Zauważ, że w tym przykładzie osadziłem regułę CSS w regule `@media`, co na pierwszy rzut oka może wydawać się nieco dziwne. Tym niemniej, o ile w zapytaniu medialnym można zamieścić reguły CSS, to nie można umieścić w nim innego zapytania medialnego. Oto kolejny hipotetyczny przykład (który lepiej odnosi się do problemu widocznego na [rysunkach 8.1 i 8.2](#)), w którym określona jest maksymalna szerokość ekranu.

reguła używana tylko wtedy, kiedy — szerokość okna przeglądarki nie jest większa niż 568px

```
@media screen and (max-width:568px) {  
    .column {float:none; width:96%; margin:0 auto;}  
}
```

W tym przykładzie, jeżeli strona wyświetlana jest na ekranie o szerokości nie większej niż 568 pikseli, CSS sprawia, że elementy o klasie `column` przestają spływać, a układają się w pojedynczą, wyśrodkowaną kolumnę o szerokości 96%.

 Ciekawy artykuł o wyświetlaniu grafiki we właściwej rozdzielczości na Apple Retina i innych ekranach o wysokiej rozdzielczości znajdziesz na stronie <http://coding.smashingmagazine.com/2012/08/20/towards-retina-web>.

Ekran iPiona 4 ma wymiary 320×480 pikseli, ale iPona 5 320×568 pikseli (przynajmniej w kontekście przeglądarki i zapytań medialnych, ponieważ występuje na nich zjawisko podwajania pikseli — piksele fizyczne są dwukrotnie większe od podanych wartości w obydwu kierunkach). Nadając właściwości `max-width` większy wymiar dużego ekranu iPona 5, sprawiam, że kolumny nie będą pływać, lecz układać się jedna pod drugą na wszystkich iPONACH.

Mówiąc krótko, ta reguła twierdzi, że maksymalna szerokość ekranu urządzenia (np. smartfona lub przeglądarki komputerowej) musi być rozpoznana jako 568 pikseli. Ta reguła nie została zatem wykorzystana na iPadzie, którego rozdzielczość to 1024×768.

Przyjrzyjmy się teraz drugiemu sposobowi formatowania zapytań medialnych, czyli atrybutowi `media` znacznika `link`.

Więcej o zapytaniach medialnych

TYPY MEDIUM

Oto najczęściej używane typy medium:

- `all` — odpowiednie dla wszystkich urządzeń;
- `handheld` — przeznaczone dla urządzeń przenośnych (zwykle z małymi, monochromatycznymi ekranami i ograniczonym przeszytem);
- `print` — przeznaczone dla materiałów podzielonych na stronice i treści wyświetlanych w trybie podglądu druku;
- `screen` — przeznaczone głównie dla kolorowych ekranów komputerowych;
- inne typy mediów to `braille`, `embossed`, `projection`, `speech`, `tty` i `tv`.

Więcej o tych typach mediów dowiesz się na stronie <http://www.w3.org/TR/CSS2/media.html>.

Zauważ, że przeglądarka może naraz używać tylko jednego typu medium. Typy mediów były już obsługiwane od czasów IE6, ale związane z nimi funkcje dopiero od IE9 i w nowszych wersjach. Zwykle nie jest to szczególny problem, ponieważ funkcji mediów na ogół używa się do wykrywania nowoczesnych urządzeń w rodzaju tabletów i smartfonów.

FUNKCJE MEDIÓW

Są to opisy właściwości urządzeń do sprawdzenia i często zaczynają się przedrostkiem `min-` lub `max-`.

Najczęściej używa się następujących:

- `min-device-width` i `max-device-width` — odnoszą się do wielkości ekranu urządzenia.
- `min-width` i `max-width` — odnoszą się do szerokości obszaru widoku, tj. okna przeglądarki.
- `orientation (portrait` lub `landscape`) — pozwalają na stosowanie różnych reguł CSS, kiedy wysokość wyświetlacza jest większa od szerokości oraz kiedy jego szerokość jest większa od wysokości.

Jeżeli chcesz użyć zapytań medialnych do utworzenia layoutu, który zmienia wymiary wraz ze zmianą wielkości okna przeglądarki, użyj właściwości `min-width` i `max-width`. Pełną listę funkcji mediów znajdziesz na stronie <http://www.w3.org/TR/css3-mediaqueries/#media>.

Typy i funkcje mediów możesz ze sobą łączyć operatorami logicznymi `and`, `all` i `not`.

Słowo kluczowe `only` ukrywa arkusze stylów przed starszymi przeglądarkami, które nie obsługują zapytań medialnych. O operatorach przeczytasz więcej na stronie https://developer.mozilla.org/en-US/docs/CSS/Media_queries#Operator_precedence.

Dobre wprowadzenie do zapytań medialnych znajdziesz na stronie <http://www.javascriptkit.com/dhtmltutors/cssmediaqueries.shtml>. Zapytań medialnych można używać w IE8 i wcześniejszych wersjach przy użyciu skryptu zastępczego `Respond.js`, o którym piszę w punkcie „Wypełnienia” w „Dodatku”.

Atrybut media znacznika link

Jeżeli przy użyciu zapytań medialnych chcesz nadać wiele reguł CSS, to lepszym rozwiązaniem może być wykorzystanie atrybutu **media** znacznika **link** do warunkowego wczytywania różnych arkuszy stylów. Jak już widziałeś, znacznik **link** służy do łączenia arkuszy stylów z dokumentami HTML. Możesz jednak podać konkretne warunki w atrybucie **media** znacznika **link**, aby wybrany arkusz stylów wczytywany był tylko w przypadku ich spełnienia. Poniżej widnieją te same przykłady, którymi posłużyłem się z regułą **@media**, ale tym razem podane są w elemencie **link** z atrybutem **media**.

```
<link type="text/css" media="print" href="css/print_styles.css" />  
<link type="text/css" media="screen and (max-width:568px)" href="css/iphone_styles.css" />
```



Rzuć okiem na listę reguł **@media**, służących do wybierania różnych urządzeń, którą skompilował Andy Clarke:
http://www.stuffandnonsense.co.uk/blog/about/hardboiled_css3_media_queries.

Wynik jest taki sam — reguły CSS nadawane są, kiedy zostają spełnione warunki określone atrybutem **media**. Drugi arkusz stylów nie zostałby wczytany, gdyby strona była przeglądana na dużym monitorze komputerowym lub iPadzie, ale zostałby użyty w przypadku wyświetlenia jej na smartfonie. Najbardziej bezpośrednim sposobem na wykorzystanie zapytań medialnych jest podzielenie ich zakresów oddziaływaniami wartościami granicznymi.

Wartości graniczne

Wartości graniczne odnoszą się do szerokości ekranu, przy których dane reguły zostają wykorzystane. Zapisuje się je następująco:

```
@media screen and (max-width:640px) { /*reguły css*/ }
```

W tym przykładzie wartością graniczną jest 640-pikselowa szerokość ekranu. Jeżeli szerokość ekranu urządzenia jest równa szerokości określonej jako wartość graniczna bądź od niej mniejsza, wykorzystany zostanie podany kod CSS.

Czasami warto określić wartości graniczne pokrywające się z szerokością ekranów określonych urządzeń, ale sam jednak odkryłem, że najważniejsze jest zapewnienie, by z layoutu zwyczajnie dało się korzystać także po jego pomniejszeniu. Innymi słowy, zamiast dostosowywać wartości graniczne do szerokości urządzeń, po prostu stopniowo zmniejszaj okno przeglądarki i wybieraj wartości graniczne dla kolejnych stylów, kiedy layoutu nie daje się już używać. W takim przypadku nie podajesz wartości w odniesieniu do konkretnych urządzeń, tylko określasz layouty dla różnych zakresów szerokości ekranu, które sprawdzają się na wszelkich urządzeniach, których szerokości mieszczą się w takich zakresach.



Kiedy docieram do szerokości layoutu, którą chcę obrać jako wartość graniczną, sprawdzam liczbę pikseli w informacjach ramki zaznaczenia mojego programu do tworzenia zrzutów ekranowych, SnapzProX Ambrosia Software.

Zmniejszając szerokość okna przeglądarki, możesz szybko zorientować się, jak layout będzie wyglądać na mniejszych urządzeniach; tym niemniej i tak należy go przetestować na takich urządzeniach. Tablety mają wymiary zbliżone do małych monitorów, więc wartość graniczna 1000 pikseli (tj. określająca reguły podawane w przypadku ekranów o szerokości równej 1000 pikseli bądź mniejszej) będzie określała wielkość layoutu na ekranach o wielkości tabletów. Zanim się za to zabierzemy, spójrzmy, jak wyłączyć automatyczne skalowanie stron na iPhonie i iPadzie.

Wartość viewport znacznika meta

Jak widziałeś na [rysunkach 8.1 i 8.2](#), iPad i iPhone zmniejszają stronę zaprojektowaną z myślą o wyświetlaniu na monitorze tak, by zmieściła się na mniejszym ekranie. To całkiem efektowna funkcja, ale zwłaszcza w przypadku iPiona wiąże się z tym, że konieczne jest przybliżenie strony i ciągłe przesuwanie widoku, żeby móc w ogóle przeczytać drobny tekst. Ponieważ mój layout ma być także dostosowany do mniejszych ekranów, chcę wyłączyć automatyczne skalowanie. Zrobię to poprzez zamieszczenie znacznika HTML `meta` w nagłówku strony.

```
<meta name="viewport" content="width=device-width; maximum-scale=1.0" />
```



O wartości `viewport` znacznika `meta` można by się rozpisać, ale zabrakłyby miejsca. Więcej na ten temat przeczytasz na stronie <http://developer.android.com/guide/webapps/targeting.html>.

Powyższy znacznik `meta` każe przeglądarce dostosować treść do szerokości ekranu, zamiast ją skalować. Choć w ten sposób mamy pewność, że layout będzie wyświetlany tak, jak to zamierzyłeś, zastosowanie tej techniki może wywoływać na urządzeniach używających iOS (tj. iPadzie i iPhonie) pewien częsty błąd. Błąd ten i sposoby jego naprawienia omówię w dalszej części rozdziału.

Optymalizacja layoutu na potrzeby tabletów



Szerokość określona wartościami procentowymi odzwierciedla drugi z trzech aspektów projektowania skalowalnego — płynność layoutu.

wartość 98% sprawia, że layout ma drobne marginesy w poziomie → `#wrapper {width:98%;}`

Strona internetowa z rozdziału 7. (czyli layout) ma obecnie stałą szerokość 980 pikseli i jest wyśrodkowana w oknie przeglądarki, kiedy jest ono szersze od layoutu. Gdy okno przeglądarki jest węższe, tak jak na tablecie lub smartfonie, prawy kraniec layoutu zostaje obcięty przez okno przeglądarki. Wybiorę zatem odrobinę wyższą wartość 1000 pikseli jako pierwszą wartość graniczną. Kiedy okno przeglądarki stanie się węższe od tej wartości, layout stanie się płynny i będzie zmieniał wielkość tak, by dostosować się do okna przeglądarki. Ten efekt łatwo osiągnąć, ponieważ wszystkie elementy strukturalne mają szerokość określoną wartościami procentowymi lub wartością `auto`, wobec czego wystarczy określić wartością procentową szerokość kontenera o ustalonej szerokości, by cały layout stał się płynny.

Po uzyskaniu płynnego layoutu trzeba się jednak zmierzyć z kilkoma problemami.

RYSUNEK 8.3. Na szerokim na 768 pikseli ekranie iPada zorientowanego pionowo elementy nachodzą na siebie



Na **rysunku 8.3** widać, że wraz ze zmniejszaniem się szerokości layoutu — tutaj widać 768-pikselową szerokość na zorientowanym pionowo iPadzie — menu nawigacyjne zaczyna nachodzić na znajdujący się po swojej lewej stronie tytuł, a znajdujący się poniżej obszar formularza i odnośników nachodzi na tekst bloga. Widać jak na dloni, że obecne rozmieszczenie menu i dwukolumnowego layoutu pod obszarem nagłówka nie sprawdza się przy mniejszych szerokościach ekranu.

Wprowadzę zatem kilka zmian layoutu właściwych tej 1000-pikselowej wartości granicznej. Przyjrzymy się najpierw kodowi CSS, a następnie wy tłumaczymy sobie zmiany, które wprowadziłem.

```
wartość graniczna 1000px ----- @media only screen and (max-width:1000px) {
    body {
        margin:0 8px 20px;
    }
    #wrapper {width:98%;}
    header {
```

dodaje prawy margines, ----- aby zapobiec przycięciu paska przewijania

layout staje się płynny -----

zwiększa wysokość nagłówka na potrzeby nowego położenia menu zapobiega scaleniu górnego marginesu elementu nav

```
height:100px;
padding:1px 0 0 0;
}
```

zamieszcza menu pod tytułem i polem wyszukiwania

```
margin-top:65px;
}
```

nie jest już potrzebne rozciąga wprowadzenie do artykułu na całą szerokość element zajmujący całą szerokość nie musi już spływać element zajmuje całą szerokość layoutu

```
section#feature_area {padding-bottom:0;}
section#feature_area article {
    float:none;
    width:auto;
}
```

element aside zajmuje całą szerokość layoutu

```
section#feature_area aside {
    float:none;
    width:96%;
    width:auto;
}
```

sprawia, że pole logowania znajduje się po lewej stronie elementu aside wyrównuje do górnego marginesu elementu nav

```
section#feature_area aside form {
    float:left;
    margin:15px 0 0 0;
}
```

zmniejsza szerokość obszaru odnośników do wpisów

```
section#feature_area aside nav {
    width:17em;
}
}
```

kończy zapytanie medialne dla maksymalnej szerokości 1000 pikseli

Na **rysunku 8.4** widać, jak ten nowy layout ulepsza wygląd strony na iPadzie w orientacji pionowej. O ile szerokość ekranu iPada w orientacji poziomej jest większa niż wartość graniczna 1000 pikseli (wobec czego layout nie ulega zmianie, co widzieliśmy na **rysunku 8.1**), strona wyświetlana jest w orientacji pionowej bardziej odpowiednio, opierając się na płynnym layoucie.

RYSUNEK 8.4. Kiedy strona jest węższa niż 1000 pikseli, zastosowany zostaje kod CSS podany w zapytaniu medialnym. Menu przesuwa się w dół, a elementy prawej kolumny przemieszczają się pod dużą lewą kolumnę



Jak widać na **rysunku 8.4**, nowe reguły zastosowane za pośrednictwem zapytania medialnego zwiększą zarówno wysokość nagłówka, jak i górny margines menu. Te zmiany przenoszą menu w nowo utworzony obszar u dołu obszaru nagłówka — znajduje się ono teraz pod obszarami `title` i `search`.

Obszar `feature_area` również ulega zmianie. Jego dwa elementy dzieci, `article` z wprowadzeniem do wpisu oraz `aside` z polem logowania i odnośnikami do wpisów, już nie spływają i mają swoje domyślne szerokości określone wartością `auto`. Znajdują się teraz jedno nad drugim, a nie obok siebie i zajmują całą szerokość layoutu. Zawarty w elemencie `aside` element `form` spływa w lewo, więc elementy `form` i `nav` znajdują się teraz pod artykułem blogowym, odpowiednio po lewej i prawej (element `nav` według kodu już spływał w prawo). Dodatkowo odrobinę zmniejszyłem znajdujący się po prawej element `nav`, żeby nie nachodził z prawej strony na formularz, dopóki szerokość okna nie dotrze do kolejnej wartości granicznej 640 pikseli — ale do tego jeszcze wróćmy.

Tym razem nie musiałem robić niczego z obszarem `book_area`. Jako że pierwotnie kontenerowi każdej książki nadałem szerokość 25%, zamiast określać ją jednostkami bezwzględnymi, książki i powiązany z nimi tekst zwyczajnie przysuwają się do siebie wraz ze zmniejszaniem szerokości layoutu.

Na iPadzie i w zmniejszonym oknie przeglądarki komputerowej taki layout zaprojektowany z myślą o szerokościach mniejszych od 1000 pikseli zapewnia użytkownikowi dużo lepsze doświadczenie. Spójrzmy, co dzieje się przy dalszym zmniejszeniu layoutu.

Optymalizacja layoutu dla smartfonów

Tak dobra sytuacja trwa jednak do pewnego czasu. Gdy szerokość obszaru widoku jest mniejsza niż 640 pikseli (dotkniemy tu do wybranych właściwych smartfonów), elementy `form` i `link` w drugim rzędzie nachodzą na siebie, podobnie jak znajdujące się pod nimi książki.

Utworzę teraz wartość graniczną na podstawie tej szerokości.

```
@media only screen and (max-width:640px) {
```

większy margines poziomy,
określony względową wartością

```
header {height:100px;}
```

wpis blogowy

```
header nav.menu {width:94%;}
```

kontener każdej książki obejmuje
całą szerokość layoutu

```
section#book_area article {
```

```
    width:auto;
```

```
    float:none;
```

```
    margin:0; padding:0;
```

```
}
```

```
section#feature_area aside form,
```

odnośniki blogowe

```
section#feature_area aside nav {
```

tworzy górne i dolne marginesy

```
    margin:10px auto;
```

```
    float:none;
```

```
}
```

kontener obrazu na całą szerokość
layoutu

```
section#book_area article .inner {
```

```
    width:98%;
```

```
    margin:0 0 0 5px;
```

```
}
```

przywraca obrócony tekst  do standardowej postaci

```
#book_area .inner h3 {  
    -webkit-transform:none;  
    -moz-transform:none;  
    -moz-transform-origin:none;  
    transform:none;  
    position:static;  
}
```

określa wielkość obrazu względem  szerokości ekranu urządzenia

```
#book_area article img {width:40%;}
```

```
section#book_area {  
    background:#fff;  
    padding: 0 10px 10px;  
    margin:0 0 10px;  
}
```

wyswietla treść chmurki  obok obrazu

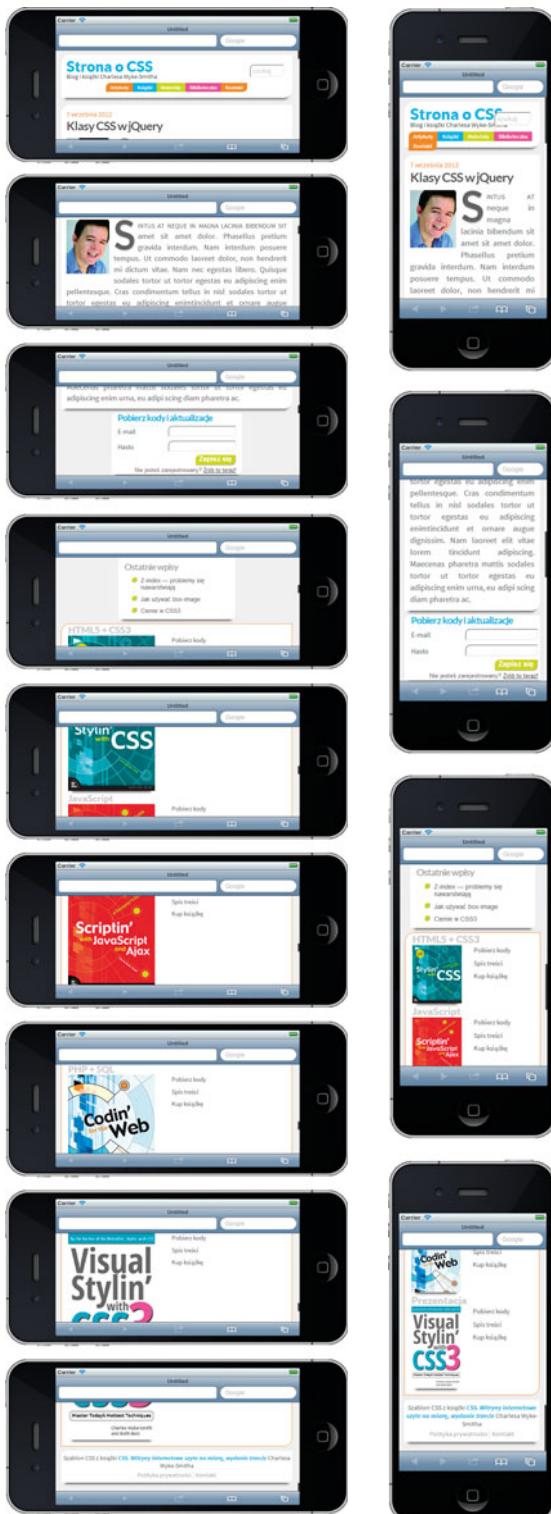
```
#book_area article aside {  
    display:block;  
    position:static;  
    float:right;  
    margin:0; padding:0 0 20px 0;  
    font-size:.8em;  
    border:none;  
    width:55%;  
    box-shadow:none;  
}
```

ukrywa trójkąty przylegające  do chmurek

```
section#book_area article aside::before,  
section#book_area article aside::after {  
    display:none;  
}  
}
```

RYSUNEK 8.5. W orientacji poziomej iPhone i iPada layout z jedną kolumną wygląda bardzo dobrze

RYSUNEK 8.6. W orientacji pionowej layout odpowiednio się mieści, ale elementy nagłówka wciąż są za duże



Ten kod CSS, używany po przekroczeniu 640-pikselowej wartości granicznej, wprowadza jeszcze większe zmiany niż po przekroczeniu 1000-pikselowej wartości granicznej. Po pierwsze, kontenery formularza i odnośników do wpisów, które znajdowały się obok siebie, już nie spływają i mają szerokość całego layoutu, wobec czego znajdują się jeden nad drugim. Nie ma już sensu, by obok okładek znajdował się obrócony tekst, a ponieważ same nagłówki w kodzie poprzedzają obrazy książek, po usunięciu przekształcenia i przywróceniu im domyślnej wartości pozycjonowania `static` przenoszą się one w miejsce nad książkami. Chmurki też już nie są potrzebne, więc wyświetlam ich niewystylizowaną treść i zamieszczam odnośniki obok kolejnych książek. Te zmiany tworzą jednokolumnowy layout, który działa dobrze na smartfonach; wszystkie główne elementy są teraz szerokie na całą stronę i znajdują się jeden nad drugim.

Zauważ, że obrazom nadałem szerokość 40% ([rysunek 8.5 i 8.6](#)), aby nie były większe od layoutu. Użyte tutaj obrazy są dość małe, ale jeśli zdarzy Ci się zamieścić na stronie obrazy, które mogą być szersze od kontenera na małym wyświetlaczu, to w arkuszu stylów umieść kod

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

dzięki któremu ich szerokość nigdy nie przekroczy szerokości ich kontenerów, a jednocześnie nie uniemożliwi im zmniejszenia się. Mamy tu do czynienia z trzecim aspektem projektowania skalowalnego — elastycznymi obrazami.

Ponieważ wszystkie rozmieszczone pionowo elementy mają szerokość okna przeglądarki, layout zmienia swoją postać, kiedy orientacja urządzenia zmienia się z poziomej w pionową i na odwrót.

Dostosowanie layoutu do orientacji pionowej

W orientacji pionowej na iPhone trzeba poradzić sobie jeszcze z dwoma problemami z nagłówkiem, które widać na [rysunku 8.6](#). Po pierwsze, zawartość menu zwija się do kolejnego wiersza, a po drugie, obszar wyszukiwania nachodzi na tytuł strony. Obydwu tym problemom można z łatwością zaradzić, więc dodajmy zapytanie medialne, które włącza się, kiedy iPhone znajduje się w orientacji pionowej. W tym zapytaniu medialnym mógłbym sprawdzić użyć słowa kluczowego `portrait`, ale nie chcę, by takie skalowanie następowało w orientacji pionowej na iPadzie, którego ekran jest większy. Zamiast tego używam ustawienia `max-width` o wartości 320 pikseli, czyli szerokości ekranu iPhone w orientacji pionowej.

```

iPhone w orientacji pionowej | @media only screen and (max-width:320px) {
zmniejsza wysokość nagłówka |   header {height:80px;}
zmniejsza tekst |   header #title h1 {font-size:1.25em;}
zmniejsza tekst |   header #title h2 {font-size:.75em;}
przesuwa pole wyszukiwania w góre |   header .search {top:12px; right:4px;}
zmniejsza menu i przesuwa je |   header nav.menu {font-size:.55em; margin-top:50px;}
w górę |   nav.menu ul li a {
zwiększa rozmiar odnośników, |     padding:5px 4px;
żeby łatwiej było je nacisnąć |     margin:0;
}
}

```

Powyższy kod CSS, jak widać na **rysunku 8.7**, zmniejsza rozmiar nagłówków i przenosi pole wyszukiwania do prawego górnego rogu. Dzięki określeniu mniejszego rozmiaru fonta wartością względnymi menu też ulega pomniejszeniu, jako że jego wielkość jest określona wyłącznie wielkością tekstu odnośników i ich dopełnieniem. Po zmniejszeniu tekstu i menu mogę odzyskać cenne miejsce w pionie poprzez drobne pomniejszenie wysokości nagłówka.

RYSUNEK 8.7. Layout teraz dobrze pasuje do 320-pikselowej szerokości iPhona



Layout jest teraz sformatowany tak, by zapewnić użytkownikowi optymalne doświadczenie na wszystkich urządzeniach, od szerokich monitorów po wąskie smartfony w orientacji pionowej. Czas zająć się drobnymi szczegółami, które pozwolą na ukończenie tego skalowalnego layoutu.

Ostatnie detale

Musimy sobie jeszcze poradzić z dwoma problemami. Wiążą się one z naprawieniem znanego błędu z rozrysowywaniem i skalowaniem na iOS (systemie operacyjnym Apple'a) oraz zapewnieniem, by rozwijane menu działały też na ekranach dotykowych.

Błąd ze skalowaniem w Safari Mobile

W Safari Mobile (przeglądarce iPhone'a) występuje błąd, który może powodować problemy ze skalowaniem i rozrysowywaniem stron podczas przejścia między orientacją pionową a poziomą. Problem z rysowaniem Webkit można naprawić skryptem JavaScript, który zamieściłem w nagłówku `head` ukończonego dokumentu HTML.Więcej na ten temat przeczytasz na stronie <http://webdesignerwall.com/tutorials/iphone-safari-viewport-scaling-bug>, a skrypt znajdziesz na stronie <https://gist.github.com/901295>.

Rozwijane menu na ekranach dotykowych

Został nam ostatni problem: to, że rozwijane menu nie działają na ekranach dotykowych. W ostatnim rozdziale użyłem zestawienia właściwości `opacity` i `visibility`, aby rozwijane menu wyświetlały się i znikaly na ekranie stopniowo. Kod wyglądał następująco:

ukrywa rozwijane menu ————— nav.menu li ul {
stopniowa zmiana przezroczystości i widoczności ————— opacity:0; visibility:hidden;
—webkit-transition:1s all;
-moz-transition:1s all;
transition:1s all;
}
wyświetla rozwijane menu ————— nav.menu li:hover ul {opacity:1; visibility:visible;}
po najechaniu kursem na menu pierwszego poziomu



Oto świetny artykuł o użyciu skryptu Modernizr: <http://webdesignernotebook.com/css/how-to-use-modernizr>.

modernizr musi wykrywać ekran dotykowe, ponieważ przejście właściwości visibility psuje menu dotykowe



Jeżeli nie chcesz używać jQuery, dodaj `ontouchstart=" "` do znacznika `body`. To również uruchamia JavaScript i zamyka menu.

Okazuje się, że urządzenia dotykowe pomijają regułę `:hover`, kiedy chcesz stworzyć przejście właściwości `visibility`. Nic dziwnego, skoro przejście następuje między dwoma stanami boolowskimi — ukrytym i widocznym — a zatem właściwość ta nie jest najlepszym materiałem na przejście. Menu nie będą działać na ekranach dotykowych, jeśli nie usunę właściwości `visibility`, ale wtedy z kolei w przeglądarkach komputerowych rozwijane menu będą wyświetlane po najechaniu kursem na obszar pod menu pierwszego poziomu, jako że będą one całkowicie przezroczyste, ale wciąż „widoczne” dla kurSORA. Rozwiążanie polega na użyciu skryptu Modernizr do sprawdzenia, czy urządzenie obsługuje dotyk, i usunięciu przejścia `visibility` w takim wypadku. Polega to na tym, że jeśli urządzenie obsługuje dotyk, to Modernizr dodaje klasę `touch` do elementu głównego `html`. Dzięki temu mogę podać kod:

```
.touch nav.menu li ul {
    -webkit-transition:1s opacity;
    -moz-transition:1s opacity;
    transition:1s opacity;
}
```

Ta reguła nie jest zatem używana na urządzeniach innych niż przenośne (takich, które nie obsługują dotyku), ale na urządzeniach dotykowych sprawia, że menu pojawia się po naciśnięciu odpowiedniej pozycji w menu pierwszego poziomu. Teraz jednak rozwijane menu zamyka się od razu, zamiast stopniowo znikać, kiedy dотykasz ekranu w dowolnym innym miejscu. Ponieważ jedyną rzeczywistą zaletą stopniowego znikania menu jest to, że możesz szybko z powrotem na nie najechać, jeśli kurSOR przypadkowo Ci się z niego zsunie, efekt natychmiastowego znikania sprawdza się całkowicie dobrze na ekranie dotygowym, na którym kurSOR nie występuje.

Zauważ, że takie menu zamyka się tylko wtedy, kiedy dotkniesz innego miejsca na ekranie, ponieważ dotknięcie ekranu w dowolnym miejscu przywołuje kod naprawiający błąd ze skalowaniem (opisany w poprzednim punkcie), który uruchamia skrypt JavaScript. Samo to wystarczy, by menu zrozumiało, że kurSOR już się nad nim nie znajduje, i zamknęło się. Kolejnym sposobem na uzyskanie tego efektu „dotknij gdzieś, żeby zamknąć” jest wyzwolenie dowolnej funkcji JavaScript. Poniżej znajduje się przykład funkcji jQuery, która nie robi zupełnie nic — jest instrukcją pustą, ale przywołanie jej po- przez dotknięcie ekranu w dowolnym miejscu zamyka menu.

```
(function(){ $(window).on('touchstart',$.noop); })();
```

Rzecz jasna, jest to zupełnie obejście problemu, ale jak to ujął znawca JavaScript Isaac Shapira, który pokazał mi tę sztuczkę: „to eleganckie obejście”. Jeżeli nie potrzebujesz omówionego wcześniej sposobu na naprawienie błędu ze skalowaniem, dodanie tej linijki kodu wystarczy do zamknięcia menu.

W blogosferze mówi się dużo o tym, jak sprawić, by rozwijane menu działały na ekranach dotykowych. Przykład eksperymentów, jakie się w tym zakresie prowadzi, znajdziesz na stronie <http://css-tricks.com/convert-menu-to-dropdown>.

O tym, jak sprawić, by rozwijane menu działały zarówno na urządzeniach wykorzystujących myszy, jak i dotyk, dowiedziałem się wreszcie w trakcie pisania tej książki. Tak czy inaczej, w przypadku urządzeń dotykowych jest jeszcze wiele do zrobienia. Zamierzam stale ulepszać tę technikę i rozwiązywać kolejne problemy, które się trafią, więc możesz śledzić mój blog, jeśli chcesz. Technika ta sprawdza się dobrze na iPadzie i iPhone oraz niewielkiej liczbie urządzeń z Androidem, na których ją testowałem.

Oto mamy stronę internetową, która wyświetlana jest prawidłowo na niemal wszystkich urządzeniach, na których można ją oglądać. Im bardziej urządzenia przenośne zyskują na znaczeniu (a już są jedynym środkiem dostępu do sieci dla coraz większej części ludzi na świecie), tym ważniejsze staje się opracowywanie skalowalnych witryn.

Podsumowanie

Tym samym kończymy przykład, rozdział i książkę. W tym rozdziale zobaczyłeś, jak projektowanie skalowalne wykorzystuje zapytania medialne, płynne layout i elastyczne obrazy, aby sprawić, by witryna była odpowiednio wyświetlana na urządzeniach o różnych wymiarach. Nie mogę zaoferować idealnych rozwiązań problemów związanych z projektowaniem skalowalnym — nikt nie może. Projektowanie skalowalne jest najbardziej aktualnym nurtem współczesnego webdesignu, a związane z nim techniki, przeglądarki i sprzęt ulegają szybkim przemianom.

CSS, a zwłaszcza CSS3, sam w sobie składa się z tysięcy funkcji, znajdujących się na różnych poziomach rozwoju, więc nie ma sensu czekać, aż zostanie ukończony. Musisz wejść w ten świat i dowieźć się, co się dzisiaj sprawdza, co nie, oraz dostosować się do zmian, jakie przyniesie przyszłość. Jasne jest, że internet zaczyna oferować zupełnie nowe możliwości wraz z tym, jak HTML5 i CSS3 stają się coraz sprawniejsze i oferują coraz więcej funkcji właściwych aplikacjom komputerowym. Mam nadzieję, że ta książka zademonstrowała Ci te możliwości i zainspirowała do ziszczenia swoich wizji.

DODATEK

Kwestie techniczne

Pisanie kodu CSS

CSS jest mechanizmem służącym do stylizacji kodu HTML. Niemal wszystkie przykłady w tej książce składają się z kodu HTML, który następnie stylizuję kodem CSS. HTML wygląda tak:

```
<p>Element HTML, który oznacza akapit</p>
```

HTML ignoruje białe znaki, entery i tabulatory; w obrębie tekstu uznawane są pojedyncze spacje, ale wszystko inne jest ignorowane. Daje Ci to różne możliwości formatowania tekstu. Poniższe przykłady są w HTML równoznaczne i wyświetlane są jako „Element HTML, który oznacza akapit”.

```
<p>Element HTML, który oznacza akapit</p>
```

```
<p>Element      HTML,      który      oznacza      akapit </p>
```

```
<p>
```

```
    Element HTML, który oznacza akapit
```

```
</p>
```

Kod CSS wygląda tak:

```
p {color:red;}
```

CSS również ignoruje białe znaki, tabulatory i entery, więc poniższe przykłady są równoznaczne:

```
p {color:red;font-size:20px;line-height:1.2;}
```

```
p {color: red; font-size: 20px; line-height: 1.2; }
```

```
p {
```

```
    color: red;
```

```
    font-size: 20px;
```

```
    line-height: 1.2;
```

```
}
```

Pierwsze dwa przykłady przedstawiają regułę z wieloma deklaracjami w jednym wierszu (w drugim jest odrobinę więcej białych znaków). W trzecim przykładzie znajduje się reguła, której każda deklaracja umieszczona jest we własnym wierszu. Wszystkie trzy przykłady dają taki sam rezultat. W swoim kodzie CSS mieszam ze sobą te style zapisu, tj. niektóre reguły zapisuję z wieloma deklaracjami w jednym wierszu, a inne z deklaracjami w osobnych wierszach. Co więcej, choć deklaracje reguły można zapisywać w dowolnej kolejności, trzymam się następującego schematu:

Reguły związane z wyświetlaniem

Reguły związane z pozycją

Marginsy, dopełnienia i obramowania

Reguły związane z fontami i tekstem

Reguły dekoracyjne

Przykład:

```
.demo {  
    display:block; position:absolute;  
    height:100px; width:300px; left:10px; top:10px;  
    margin:0 5px; padding:10px;  
    font-size:10px; line-height:1.2;  
    background-color:#eee; border:1px solid; border-  
    radius:6px;  
}
```

Sekwencja zaczyna się od najważniejszej moim zdaniem informacji o elemencie — gdzie i jak jest położony na stronie — a kończy się najmniej istotną informacją — jego wyglądem. Czasami zmieniam kolejność i na przykład zamieszczam deklarację o marginsie na początku, jeśli jest to najważniejsza rzecz w regule. Grupowanie powiązanych deklaracji w jednym wierszu uważam za całkiem dobrą praktykę. Umieszczenie wszystkich deklaracji reguły w jednym, długim wierszu sprawia, że CSS staje się mało czytelny, a umieszczenie każdej deklaracji w osobnym wierszu tworzy bardzo długie arkusze, które trzeba ciągle przewijać. To oczywiście kwestia osobistych upodobań — nie ma jednego, prawidłowego sposobu zapisu. W książce korzystam z różnych rodzajów zapisu, czasami w celu zaoszczędzenia miejsca, a czasami aby móc komentować kolejne deklaracje.

Zdecydowanie jednak polecam wypisanie reguł CSS w arkuszu w takiej samej kolejności, w jakiej występują stylizowane przez nie elementy w kodzie HTML, zamiast umieszczać każdą nową regułę na końcu. Przyjrzyj się jednemu z dłuższych arkuszy stylów w plikach, które możesz pobrać, np. kodowi formularza z rozdziału 6. Zobaczysz, że reguły CSS zapisane są w dokładnie takiej samej kolejności, jak elementy przez nie stylizowane. Arkusze stylów bywają bardzo długie i trudno prześledzić, do których elementów dany styl się odnosi, jeśli nie są zapisane w stosownej kolejności.

Testowanie kodu

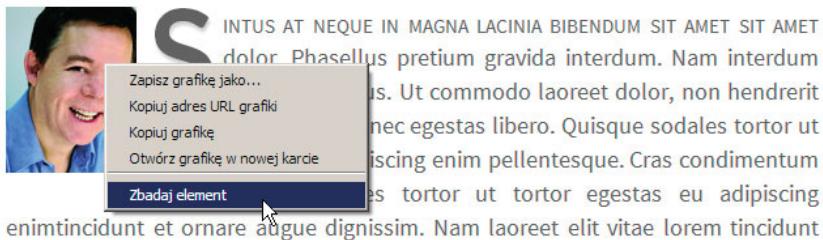
Część czasu spędzasz na pisaniu kodu, CSS lub innego, a część na debugowaniu go, czyli określaniu, dlaczego działa inaczej, niż powinien. Lubię mawiać, że „debugowanie jest systematycznym pozbywaniem się przeświadczeń”. Innymi słowy, chodzi o określenie różnicy między tym, co myślisz, że kod robi, a tym, co rzeczywiście robi.

Pod tym względem bardzo przydatna jest możliwość jasnego określenia, do którego elementu HTML odnosi się dana reguła CSS. Ze względu na kaskadowość CSS wiele różnych reguł CSS może określać jakąś właściwość, ale tylko jedna z nich nadaje jej ostateczną wartość. Styl fonta znacznika `body` w HTML jest dziedziczony przez wszystkie elementy tekstowe strony, ale jeśli danemu elementowi tekstowemu zdefiniowano inny styl fonta, to właśnie on zostanie użyty, przesłaniając wartość odziedziczoną po `body`. Okreście, który styl jest decydujący, często nie jest proste, więc poniżej pokażę Ci, jak przejrzeć wszystkie style odnoszące się do danego elementu i stwierdzić, które z nich rzeczywiście zostaną nadane. Zrobię to na przykładzie mojej strony.

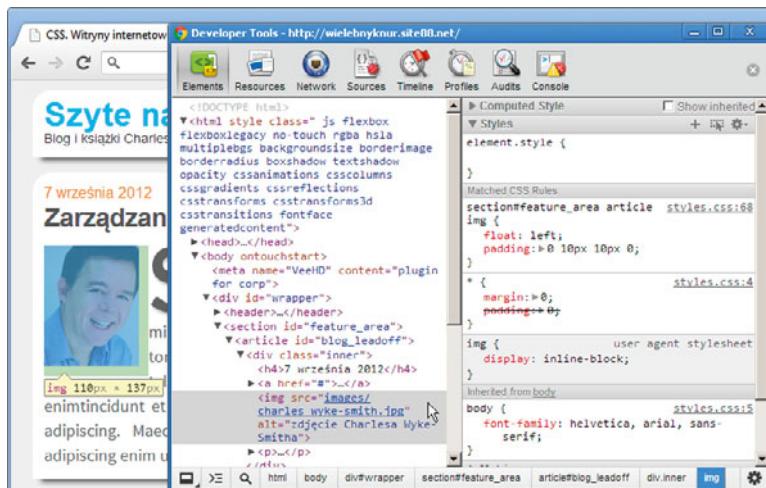
Jak widać na **rysunku A.1**, po kliknięciu elementu strony prawym przyciskiem myszy w przeglądarce (w tym przypadku Chrome) i następnie polecenia *Zbadaj element* na ekranie otwiera się okno *Developer Tools*. W *Developer Tools* widnieje zarówno element w kodzie HTML, jak i spis reguł CSS, które się do niego odnoszą, tak jak widać na **rysunku A.2**.

7 września 2012

Zarządzanie klasami CSS przy użyciu jQuery



RYSUNEK A.1. (Po prawej). Kliknięcie elementu prawym przyciskiem myszy i polecenia Zbadaj element w menu otwiera okno Web Inspector



RYSUNEK A.2. (Poniżej). Okno Web Inspector opisuje wybrany element

 Zdecydowanie polecam Firebuga, który jest dużo bardziej zaawansowanym narzędziem webdeweloperskim. Można w nim przejrzeć strukturę DOM i debugować kod JavaScript, a jest dostępny jako dodatek do Firefoxa. Aby go zainstalować, kliknij Dodatki w menu głównym Firefoxa, wyszukaj w menedżerze dodatków „Firebug”, a następnie postępuj zgodnie z wskazówkami instalacyjnymi.

W listingu CSS widać, że choć w regule określonej selektorem gwiazdkowym podano dopełnienie o wartości `0`, styl ten jest przesłonięty (o czym świadczy przekreślenie) bardziej precyzyjną regułą `section#feature_area article img`. Oznacza to tyle, że zdjęcie ma dopełnienie z prawej i u dołu, co tworzy odstęp między nim a tekstem. W zakładce HTML możesz wskazać dowolny element, aby zobaczyć jego style w zakładce CSS.

Możliwość sprawdzenia, które style rzeczywiście są nadawane wybranemu elementowi HTML, jest oczywiście niezwykle przydatna w toku tworzenia kodu CSS. Jeżeli zmieniłeś styl, a wygląd elementu się nie zmienił, to możesz otworzyć okno Web Inspector, żeby sprawdzić, czy element wykorzystuje zmodyfikowaną regułę, czy jakąś inną. Po skończeniu przeglądania możesz nacisnąć krzyżyk w lewym górnym rogu okna.

Obsługa starszych przeglądarek

HTML5 i CSS3 oferują wiele użytecznych i łatwych w użyciu funkcji, więc trudno się dziwić, że możesz zechcieć z nich korzystać, nie przejmując się starszymi przeglądarkami, które nie obsługują tych wszystkich bajerów. Tymczasem niektóre z nowych funkcji CSS3 zwyczajnie nie będą działać, a niekiedy wręcz powodować problemy z wyświetlaniem stron w przeglądarkach, które ich nie obsługują. Dobrym przykładem są właściwości `display:table` i `box-sizing`, które bardzo ułatwiają tworzenie kolumn (co omówiłem w rozdziale 5.), ale mogą dezorganizować layout w IE6 i IE7. Zdecydowanie warto wobec tego testować strony w starszych przeglądarkach i w razie potrzeby podawać kod zastępczy, z którego będą mogły skorzystać.

Do niedawna sposobem na zapewnienie obsługi w różnych przeglądarkach o różnych możliwościach było ich wykrywanie. Polegało to na wykorzystaniu JavaScript do sprawdzenia ciągu przeglądarki zawierającego nazwę klienta użytkownika i podawaniu kodu dostosowanego do ułomności danej przeglądarki. Ostatecznie jednak nie jest istotne, jakiej przeglądarki ktoś używa, tylko jakie ona obsługuje funkcje. Współcześnie nie przejmujemy się samą przeglądarką, ale tym, jakie funkcje są przez nią obsługiwane bądź nie, oraz zapewnimy kod zapasowy i skrypty, które pozwalają na obejście wszelkich problemów z obsługą.

Kod alternatywny

Kod zapasowy jest kodem podawanym starszym przeglądarkom zamiast kodu CSS3, którego nie obsługują.

Najprostszy kod alternatywny to brak kodu — często wystarczy niczego nie robić, żeby było dobrze. Jeśli na przykład tworzysz zaokrąglone rogi przy użyciu CSS3, to nie będą one widoczne w IE6 i IE7. Użytkownicy tych przeglądarek będą widzieć pola ze zwyczajnymi rogami. To chyba żaden problem — użytkownicy i tak nie wiedzą, co tracą, a zaokrąglone rogi prawdopodobnie nie są nieodzowne do przeglądania treści. W innych przypadkach będziesz jednak musiał stworzyć zastępczy kod dla przeglądarek pozbawionych obsługi pewnych funkcji.

Wersje Internet Explorer starsze niż IE9 nie obsługują złożonych tła, więc można utworzyć prosty kod zastępczy, który składa się z pojedynczej deklaracji tła, poprzedzającej regułę określającą więcej niż jedno tło.

```
.someElement {background-image:url(images/basic_image.jpg);}
```

```
.someElement {background-image:  
    url(images/cool_image1.jpg),  
    url(images/cool_image2.jpg),  
    url(images/cool_image3.jpg);  
}
```

Wszystkie przeglądarki najpierw odczytują pierwszą regułę tła, ale tylko przeglądarki, które mogą wyświetlać więcej niż jedno tło, postępują ostatecznie zgodnie z drugą regułą. Jeżeli przeglądarka nie rozumie deklaracji CSS, w której znajduje się nieobsługiwany kod lub jakiś błąd, to zostaje ona pominięta, a przeglądarka przechodzi do odczytywania kolejnej deklaracji. Przeglądarka IE8 i starsze zignorowałaby zatem drugą regułę i wyświetliłyby tło *basic_image.jpg*.

KOMENTARZE WARUNKOWE

Zauważ, że jeżeli rzeczywiście tworzysz kod wyłącznie na potrzeby IE, to kod zastępczy możesz podawać w komentarzach warunkowych takich jak ten:

```
<!--[if lte IE 8]> <!-- komentarz warunkowy dla IE -->  
    <link src="tylko_dla_ie.css" rel="stylesheet" />  
<![endif]-->
```

Ten szczególny rodzaj komentarza HTML jest ignorowany przez przeglądarki inne niż IE, wobec czego zawarty w nim kod będzie wykonywany tylko przez przeglądarki IE. Wczytuję tutaj arkusz stylów z dodatkowymi stylami, jeżeli przeglądarka to IE8 lub starsza wersja. Możesz użyć kodu **lte** („mniejsze bądź równe”), **lt** („mniejsze”), **gte** („większe bądź równe”), **gt** („większe”) lub wskazać konkretną wersję przeglądarki w rodzaju **IE 6**, aby konkretnie odnieść się do różnych wersji IE.

Czasami, jeśli przeglądarka w ogóle nie obsługuje jakiejś funkcji, a nie wystarczy podanie kodu alternatywnego, konieczne jest skorzystanie ze skryptu.



Wyczerpujący listę wypełnień zebrańych przez Paula Irisha znajdziesz na stronie <https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills>.

Wypełnienia

Wypełnienia to kody JavaScript, zapewniające przeglądarkom funkcje, których w innym wypadku nie mają. Istnieje wiele skryptów zastępujących praktycznie wszystkie funkcje CSS3 i HTML5 — od odtwarzania filmów po cienie — które pozwolą Ci nauczyć stare przeglądarki nowych sztuczek.

Aby skorzystać ze skryptu, pobierz go i zamieść w katalogu z Twoją stroną (sam tworzę w tym celu folder *pomocnicy*), a następnie podaj w nagłówku strony znacznik `script`.

```
<script type="text/javascript" src="pomocnicy/selectivizr.js"></script>
```

Praktycznym narzędziem pozwalającym na sprawdzenie, czy należy skorzystać z wypełnienia, jest Modernizr. Modernizr (<http://modernizr.com>) to plik JavaScript, który pozwala na sprawdzenie obsługiwanych przez przeglądarkę użytkownika funkcji HTML5 i CSS3. Robi się to poprzez dodanie listy klas do elementu głównego `html`, wskazującej, które funkcje są obsługiwane, i ustawienie właściwości obiektu JavaScript `modernizr` tak, by móc owe funkcje przetestować przy użyciu JavaScript. Dodane klasy są najbardziej przydatne w pracy z CSS.

Oto kilka przydatnych wypełnień:

- **html5shiv.js** (<http://code.google.com/p/html5shiv>) umożliwia IE8 i wcześniejszym wersjom rozpoznawanie nowych elementów HTML5 w rodzaju `section`, `article`, `nav` itp.
- **selectivizr** (<http://www.selectivizr.com>) pozwala, by zaawansowane selektory CSS w rodzaju `::first-child` działały w tych wersjach IE (6, 7 i 8), które ich standardowo nie obsługują.
- **IE9.js** (<http://code.google.com/p/ie7-js>) naprawia wiele błędów IE i brakujących funkcji w wersjach od IE6 do IE9.
- **CSS3Pie** (<http://css3pie.com>) daje wersjom od IE6 do IE9 zdolność obsługi graficznych funkcji CSS3 w rodzaju zaokrąglonych rogów, gradientów tła, obrazów obramowań, cieni tła oraz kolorów RGBa.

- **Respond.js** (<https://github.com/scottjehl/respond>) sprawia, że pytania medialne działają w starszych przeglądarkach.
- **-prefix-free** (<http://lea.verou.me/projects>) dodaje prefiksy do deklaracji, w których są potrzebne; więcej na ten temat przeczytasz w ramce „Prefiksy” w rozdziale 3.
- **borderBoxModel.js** (<https://github.com/albertogasparin/borderBox-Model>) pozwala IE6 i IE7 obsługiwać właściwość **box-sizing** w CSS3.

Z powyższych wypełnień korzystam najczęściej. Są niezwykle przydatne przy obsłudze starszych przeglądarek, zwłaszcza Internet Explorer. Dodatkowe informacje techniczne znajdziesz na mojej stronie www.stylinwithcss.com.

Skorowidz

960 Grid, 152

A

Adobe Dreamweaver, 7, 18
 Adobe Fireworks, 264
 Adobe Illustrator, 264
 adres URL, 9, 44
 akapit, 2, 5, 10, 19, 20, 71
 Apple Retina, 275
 arkusz stylów, 25, 50, 274, 291, 293
 autorski, 51
 przeglądarki, 50
 użytkownika, 50

B

biblioteka fontów, 127, 129
 brat, 32, 33, 46

C

chmurka, 224, 267, 268
 Chrome, 128, 223
 cień, 241, 254
 Clarke Andy, 277
 cudzysłów, 19
 cytat, 18

D

debugowanie, 293
 deklaracja, 26, 55
 waga, 52
 Document Object Model, *Patrz: DOM*
 dokument HTML
 dodawanie stylów, 25
 struktura, 7, 20, 21

DOM, 20, 21, 22, 29
 dopełnienie, 63, 66, 68, 71, 161, 163
 dziecko, 6, 13, 21, 32, 33, 81
 dziedziczenie, 49, 110, 119

E

ekran, 273, 275, 277, 279, 282
 dotykowy, 287, 289
 element, 55, 50, *Patrz też: znacznik*
 blockquote, 18, 19, 146
 blokowy, 10, 15, 19, 86, 93, 125
 szerokość, 15, 70, 153
 body, 16
 div, *Patrz: znacznik div*
 dopełnienie, *Patrz: dopełnienie*
 fieldset, 206, 210
 figcaption, 225
 figure, 225
 form, 205, 221
 generowany dynamicznie, 224
 liniowy, 10, 13, 16, 93, 125
 margins, *Patrz: margins*
 oblewanie tekstem, 76
 obramowanie, *Patrz: obramowanie*
 oczyszczający, 75
 pływający, 75, 76, 77, 78, 80, 194
 pozycjonowanie, 62, 87, 90
 bezwzględne, 88, 91, 234, 267
 stałe, 89
 statyczne, 86
 względne, 87
 przerośnięty, 168
 span, 147
 szerokość, 70, 73, 77, 244
 tabelowy, 10
 table, 178
 tekstowy, 2, 5
 label, 207

- element
tr, 178
warstwa, *Patrz:* warstwa
wyśrodkowanie, 244, 248
z cieniem, *Patrz:* cień
z zaokrąglonymi rogami, *Patrz:* zaokrąglone
rogi
zagnieźdzony, 16, 18
złożony, 5
encja, 19, 143
&, 143
&bdquo, 19
>, 143
&rdquo, 19
- F**
- Firebug, 294
Firefox, 128
 Web Developer, 14, 16
font, 108, 110, 111, 112, *Patrz też:* tekst
 bezszerfowy, 110
 Embedded OpenType, 129
 internetowy, 126, 128, 129
 na serwerze, 108, 128
 na zewnętrznych serwisach, 108
 OpenType, 128
 Scalable Vector Graphics, 129
 szeryfowy, 110
 TrueType, 128
 zainstalowane w systemie
 użytkownika, 108, 110
Font Squirrel, 129, 130
Fontspring, 130
formatowanie graficzne, 62
formularz, 5, 45, 201
kodowanie, 209
kontrolka, 207, 208
oznaczenie, 207
- stylizacja, 210, 220
wysyłanie, 206
wyszukiwania, 221, 239
- G**
- Google Web Fonts, 127, 145
gradient, 104
- I**
- identyfikator, 35, 38, 39
 związany z JavaScript, 39
inicjał, 47, 147
interfejs użytkownika, 186
interlinia, 123
Internet Explorer, 129, 130, 170, 171, 177, 295,
 296, 297
iPad, 273, 289
iPhone, 273, 275, 285, 287, 289
Irish Paul, 171, 297
- K**
- kanał alfa, 61
kaskadowość, 50, 293
 zasady, 52, 54
klasa, 35, 36, 39, 40, 41, 53, 182
 error, 215
kod alternatywny, 295
kod prezentacyjny, 164
kolor, 27, 55
 HSL, 59, 61
 jasność, 60
 krycie, 61
 nasycenie, 60
 nazwa, 57
 tła, 95
 wartość
 numeryczna, 58
 procentowa, 59
 szesnastkowa, 58

koło barw, 60
 komentarz warunkowy, 296
 kompatybilność wstępna, 7
 kontener, 77, 78, 80, 81, 163, 233, 274
 kratka, 208, 210

L

layout, 151, 152, 231
 płynny, 172, 176, 274, 279
 skalowanie, 272, 274, 278, 282, 287
 szerokość, 152, 153
 wielokolumnowy, 151, 152, 153, 154, 179
 elastyczny, 151, 152
 o stałej szerokości, 151, 152
 płynny, 151, 152
 wielorzędowy, 179, 180
 wysokość, 152, 153
 leading, 123
 lista, 5, 187, 188

M

margins, 63, 67, 68, 71, 161, 245
 jednostki miary, 69
 scalenie, 68
 media query, *Patrz: zapytanie medialne*
 medium, 276, *Patrz też: zapytanie medialne*
 menu, 186, 242
 poziom
 kolejny, 197
 najwyższy, 192
 poziome, 189
 rozwijane, 191, 195, 196, 246
 ścieżka wyboru, 200
 wyboru, 208
 model
 formatowania graficznego, *Patrz: formatowanie*
 graficzne
 obiektowy dokumentu, *Patrz: DOM*

polowy, 62, 70
 RGB, *Patrz: RGB*
 Modernizr, 101, 248, 288, 297

N

nagłówek, 2, 5, 10, 12
 h1, 2, 5, 21, 237

O

obramowanie, 63, 65, 71, 161, 163
 obraz, 9, 10, 13
 elastyczny, 274
 tła, 95, 96, 102
 odniesienie, 3
 odnośnik, 9, 10, 38, 44, 186
 do wpisów, 258

P

pole, *Patrz: element*
 logowania, 253
 potomek, 49
 pozycjonowanie, *Patrz: element pozycjonowanie*
 prefiks, 103
 projektowanie skalwalne, 272, 274, 278,
 282, 289
 przejście CSS, 222
 dodawanie, 223
 przełącznik, 208, 210
 przezroczystość, 61, 229, 247
 przodek, 32, 49
 pseudoelement, 44, 47, 147
 pseudoklasa, 43
 focus, 45
 interfejsu, 43
 precyzja, 44, 53
 strukturalna, 43, 46
 target, 45

R

- reguła CSS, 24, 26, 27, 55
 - !important, 52
 - @font-face, 128, 130
 - @import, 25
 - @media, 274, 275, 277
 - deklaracja, *Patrz:* deklaracja selektor, *Patrz:* selektor rem, 114
- Responsive Web Design, *Patrz:* projektowanie skalowalne
- RGB, 27, 55, 61
- rodzic, 6, 13, 46, 78, 79, 80
- RWD, *Patrz:* projektowanie skalowalne

S

- Safari, 128, 223
 - Mobile, 287
- selektor, 26, 35, 44, 53, 147
 - atrybutu, 28, 41, 42
 - nazwy, 41
 - gwiazdkowy, 166
 - identyfikatora, 28, 41
 - klasy, 28, 36, 41, 53
 - kontekstowy, 28, 29, 30, 32, 35, 44, 297
 - niepierwodorny, 188
 - potomka, *Patrz:* selektor kontekstowy
 - precyzja, 53
 - pseudoklasy, 44
 - uniwersalny, 34
 - wieloklasowy, 37
- Shapira Isaac, 289
- smartfon, 282
- stopka, 268
- stos, 227
- strona HTML
 - skalowanie, *Patrz:* projektowanie skalowalne
 - szablon, 7, 232

strzałka, 228

- styl
 - lokalny, 25, 51
 - osadzony, 25, 51
 - przesłanianie, 25
 - zewnętrzny, 25, 51
- źródło, 50

T

- tabela, 5
 - CSS3, 177
- tekst, 108, *Patrz* też: font
 - cień, 254
 - jednostki
 - bezwzględne, 113
 - względne, 114
 - kwerendy, 221
 - obrócony, 263
 - stylizacja, 130, 131, 143, 144, 146, 147, 148
 - w siatce, 130, 131, 146
 - wielkość, 113
 - właściwości, 117
 - tło, 93, 95, 96, 97, 99, 100, 101, 102, 104, 295
 - trójkąt, 228, 267
 - Typekit Adobe, 127
 - typografia, 130, 131, 145
 - klasyczna, 141
- W
 - validacja, 3
 - warstwa, 93, 94
 - wartość
 - koloru, 55, 57
 - liczbowa, 55, 56
 - słowna, 55
 - Web Inspector, 294
 - właściwość
 - animacja, 222

background, 94, 101
background-attachment, 94, 100
background-break, 94, 102
background-clip, 94, 102, 194
background-color, 94, 95
background-image, 94, 95
background-origin, 94, 102
background-position, 94, 97
background-repeat, 94, 96
background-size, 94, 99
border, 63
bottom, 87
box-sizing, 168, 180
clear, 75, 76, 81, 82
clear:both, 160
clear:left, 160
display, 10, 93, 225, 245, 295
float, 75, 76, 78
font, 109, 117
font-family, 109, 145
font-size, 109, 112
font-style, 109, 115
font-variant, 109, 116, 145
font-weight, 109, 116
left, 87
letter-spacing, 118, 119, 145
line-height, 118, 123, 145, 148
opacity, 247, 248, 287
overflow, 79
position, 86, 87, 88, 89, 90, 194
right, 87
table, 172, 177
table-cell, 178
text-align, 118, 122, 244, 245, 271
text-decoration, 118, 122
text-indent, 118, 119
text-transform, 118, 124
tła, 94
top, 87

transform, 103, 263
transform, 264
transform-origin, 264
vertical-align, 118, 125
visibility, 248, 287, 288
word-spacing, 118, 121, 145
word-wrap, 171
zbiorcza, 64
z-index, 224, 227
Wroblewski Luke, 201, 272
wypełnienie, 297

Y

ySlow, 166

Z

zaokrąglone rogi, 240
zapytanie medialne, 274, 276, 281, 285
znacznik, 1, 2, *Patrz też: element*
 a, 9, 93
 abbr, 20
 alt, 4
 article, 1, 2, 11, 44, 182, 263
 aside, 1
 atrybut, 4
 alt, 4
 charset, 8
 class, 4, 28, 35
 href, 9
 id, 4, 28, 35, 38
 placeholder, 201
 src, 4, 9
 blockquote, 18, 19
 blokowy, 4
 body, 7, 8, 16, 293
 cite, 18
 div, 2, 57, 81, 90, 91, 92, 119, 163, 164, 165,
 166, 168, 171, 178, 179, 180, 182, 184,

- znacznik
 - div main_wrapper, 176
 - div threecolwrap, 176
 - DOCTYPE, 7
 - em, 6, 20
 - footer, 1, 44, 159, 167, 168, 233, 268
 - główny, *Patrz:* znacznik html
 - h1, *Patrz:* nagłówek h1
 - head, 7, 8
 - header, 1, 233, 236, 238
 - html, 7
 - img, 4, 9, 13, 93
 - input, 201
 - li, 5, 6, 187, 194, 271
 - liniowy, 4, 19, 20
 - link, 25, 277
 - listy, 5
 - meta, 8, 278
 - nav, 1, 39, 44, 49, 176, 180, 182, 187, 234, 244
- nieokalający, 2, 3
- okalający, 2
- ol, 5, 6
- otwierający, 2, 6
- pozbawiony znaczenia semantycznego, 2
- section, 1, 21, 182, 233
- sidebar, 44
- span, 2, 36, 37, 47, 93, 147, 164, 212
- strong, 20
- strukturalny, 1, 21
- style, 26, 274
- tag, 274
- title, 5, 8, 239
- treści, 1
- ul, 5, 39, 40, 186, 189, 190, 191, 193, 194, 197, 200, 236, 242, 245, 246, 258, 271
- zagnieżdżanie, 6, 16, 18
- zamykający, 2, 3, 6
- znak biały, 13, 291