Helion

SMASHING MAGAZINE

Eric A. Meyer

TEN FANTASTYCZNY CSS

ERIC MEYER O TWORZENIU NOWOCZESNYCH UKŁADÓW STRON WWW

Wykorzystaj potencjał technologii CSS 3 i HTML 5 pod okiem najsłynniejszego eksperta w tej dziedzinie!

- Przegląd najbardziej przydatnych narzędzi, selektorów oraz nowości w języku CSS 3
- Ponad piętnaście skutecznych technik rozmieszczania elementów na stronie
- Sposoby na tworzenie efektownych układów oraz formatowanie i stylizowanie tabel

helion.pl

Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

Katalog książek

- Katalog online
- Zamów drukowany katalog

Twój koszyk

- Dodaj do koszyka

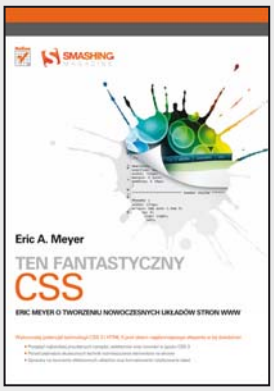
Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

Czytelnia

- Fragmenty książek online

Ten fantastyczny CSS! Eric Meyer o tworzeniu nowoczesnych układów stron WWW



Autor: Eric A. Meyer
Tłumaczenie: Łukasz Piwko
ISBN: 978-83-246-3242-8
Tytuł oryginału: Smashing CSS: Professional Techniques for Modern Layout
Format: 168×237, stron: 280

Wykorzystaj potencjał technologii CSS 3 i HTML 5 pod okiem najsłynniejszego eksperta w tej dziedzinie!

Czym byłby dzisiaj Internet, gdyby nie fantastyczne możliwości CSS? Przestrzenią pełną nudnych, podobnych do siebie i zapewne średnio atrakcyjnych dla współczesnych użytkowników stron WWW… Choć jeszcze dziesięć lat temu technologii tej wróżono odejście do lamusa, dziś swoją popularnością dorównała językowi HTML i stała się już niemal tak samo powszechna. Kaskadowe arkusze stylów spotkamy wszędzie, od przeglądarek internetowych, przez zaawansowane sklepy internetowe, po aplikacje do czatowania. Na tym nie koniec! CSS nadal rozkwita – jego ogromne możliwości oraz zakres użycia coraz bardziej się rozszerzają! A w branży twórców stron internetowych niewiele jest osób, które potrafią tak dobrze objaśnić wszystkie aspekty korzystania z języka CSS, jak słynny Eric Meyer – autor tej fantastycznej książki!

To właśnie z nim wyruszysz w podróż do najeżonego nowościami i zmianami świata najświeższych specyfikacji HTML 5 i CSS 3. To dla Ciebie stworzył ten nowoczesny przewodnik z setką porad, skutecznych metod i praktycznych sztuczek w zakresie tworzenia najwyższej jakości witryn internetowych przy użyciu CSS. W pierwszej części znajdziesz krótki przegląd przydatnych narzędzi i podstawowych technik, wliczając w to mało znane selektory CSS. Następnie zobaczysz, co można zrobić przy użyciu CSS – poznasz ciekawe efekty oraz różne sposoby osiągania tego samego celu i tworzenia wydajnych układów. Dowiesz się, jak CSS 3 współdziała z biblioteką JQuery.
W ostatniej części znajdziesz opis technik zaawansowanych. Co ważne, każdy opis jest niezależny od pozostałych, możesz więc otworzyć książkę na dowolnej stronie i wykorzystać w swojej pracy to, co się na niej znajduje – bez obaw, że straciłeś coś ważnego.

W tej niezwykłej książce znajdziesz między innymi:

- opis więcej niż piętnastu technik rozmieszczania elementów na stronie (clearfix, układy dwu- i trzykolumnowe, układy z fałszywymi kolumnami, One True Layout, Holy Grail, układy oparte na jednostce em, płynne siatki, kleiste stopki)
- sposoby ukrywania elementów i wyrzucania ich poza ekran
- metody definiowania tła elementów body i html w języku XHTML
- opis wielu efektów CSS (wyskakujące okienka CSS, tworzenie nieregularnych kształtów na stronie, zaokrąglanie rogów, sprajty CSS, Sliding Doors, Liquid Bleach)
- techniki formatowania tabel za pomocą CSS, w tym elementów thead, tbody, tfoot i nagłówków wierszy
- sposoby formatowania wybranych kolumn, stylizowania tabel przy użyciu JQuery, zmieniania tabel w wykresy i mapy
- przegląd niektórych nowości w języku CSS 3 (definiowanie wielu obrazów w tle elementów, model kolorów RGBA)

Opanuj najlepsze techniki tworzenia nowoczesnych układów stron WWW!

Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2011

Spis treści

CZĘŚĆ I	**PODSTAWY**	**15**
Rozdział 1	**Narzędzia**	**17**
	Firebug	18
	Pasek narzędzi Web Developer	24
	Pasek narzędzi Internet Explorer Developer Toolbar	29
	Dodatek Dragonfly dla Opery	33
	Dodatek WebInspector do przeglądarki Safari	35
	XRAY	37
	Narzędzie Selectoracle	38
	Diagnostyczny arkusz stylów	40
	Zerowanie stylów	42
	Skrypt IE9.JS	45
Rozdział 2	**Selektory**	**47**
	Pseudocoś tam	48
	Precyzyjne lądowanie	50
	Precyzja selektorów	51
	Reguła ważności	52
	Co się dzieje, gdy zostanie pominięta jakaś wartość w zapisie skróconym?	53
	Wybiórcze przesłanianie własności w zapisie skróconym	55
	Selektor uniwersalny	57
	Identyfikatory a klasy	58
	Łączenie identyfikatorów z klasami	61
	Przypisywanie jednego elementu do wielu klas	61
	Selektory atrybutów	62
	Selektor atrybutu class	64
	Identyfikatory a selektory atrybutów	65
	Selektor podłańcuchów wartości atrybutów	66
	Selektor podłańcuchów wartości atrybutów — ciąg dalszy	68
	Selektory dzieci	70
	Częściowa imitacja selektora elementów dzieci	72
	Selektory elementów siostrzanych	73
	Generowanie treści	75

CZĘŚĆ II NIEZBĘDNIK 79

Rozdział 3 Porady 81
Sprawdzaj poprawność kodu! 82
Kolejność własności pisma 83
Wysokość linii 83
Bezjednostkowe wartości własności line-height 84
Określaj styl obramowania 86
Ustawianie koloru obramowania 86
Wyłączanie wyświetlania elementów 88
Wyłączanie widoczności elementu 89
Wyrzucanie elementów poza ekran 90
Obrazy zamiast tekstu 92
Style dla druku 94
Pisanie arkuszy stylów dla druku 95
Łącza blokowe 96
Margines czy dopełnienie? 97
Wycinanie list 99
Definiowanie punktorów list 100
Punktory w tle 102
Generowanie markerów 105
Masz do dyspozycji więcej kontenerów, niż myślisz 107
Tła dokumentu 110
Arkusze serwerowe 111

Rozdział 4 Układy 115
Stosowanie obrysu zamiast obramowania 116
Ustawianie bloków na środku 118
Kontenery elementów pływających — przepełnienie 121
Pływające kontenery elementów pływających 123
Clearfix 125
Kliring elementów przylegających 126
Prosty układ dwukolumnowy 128
Prosty układ trzykolumnowy 129
Fałszywe kolumny 132
Technika Liquid Bleach 135
One True Layout 138
Święty Graal 142
Płynne siatki 146
Układ oparty na jednostkach em 150
Ujemne marginesy w układzie normalnym 154
Pozycjonowanie w kontekście 156
Wyjście poza kontener 158
Pozycjonowanie na sztywno nagłówków i stopek 161

Rozdział 5 **Efekty** **165**
Complexspiral 166
Menu podręczne w CSS 170
Menu CSS 172
Nieregularne kształty 174
Zaokrąglanie rogów przed nastaniem CSS 3 177
Zaokrąglanie rogów przy użyciu CSS 3 181
Sprajty CSS 183
Sliding Doors 185
Przesuwane drzwi z przycinaniem 189
Paralaksa CSS 191
Nieregularne kształty pływające 193
Lepsze nieregularne kształty pływające 197
Pola obrazów 201
Ograniczanie rozmiaru obrazów 202

CZĘŚĆ III NAJNOWSZE TECHNIKI **205**

Rozdział 6 **Tabele** **207**
Nagłówek, treść główna i stopka tabeli 208
Nagłówki wierszy 211
Formatowanie według kolumn 213
Tworzenie map z danych tabelarycznych 217
Wykresy z tabel 224

Rozdział 7 **Pieśń przyszłości** **233**
Formatowanie elementów HTML 5 234
Imitacja elementów HTML 5 za pomocą nazw klas 236
Zapytania o media 237
Formatowanie wybranych elementów-dzieci 242
Formatowanie wybranych kolumn 246
Kolory RGB z kanałem alfa 249
Kolory HSL i HSL z kanałem alfa 250
Cienie 252
Definiowanie wielu obrazów w tle elementów 254
Przekształcenia dwuwymiarowe 258

Skorowidz **269**

3 PORADY

KAŻDEMU CZASEM W ŻYCIU przydają się dobre rady. Do moich ulubionych należą „lepszy mały dom na uboczu w miłej okolicy niż wielki dom w niebezpiecznym miejscu" i „nie jedz ołowiu". To samo dotyczy także CSS — krótka sentencja zawarta w kilku słowach może szybko naprowadzić Cię na właściwy tor.

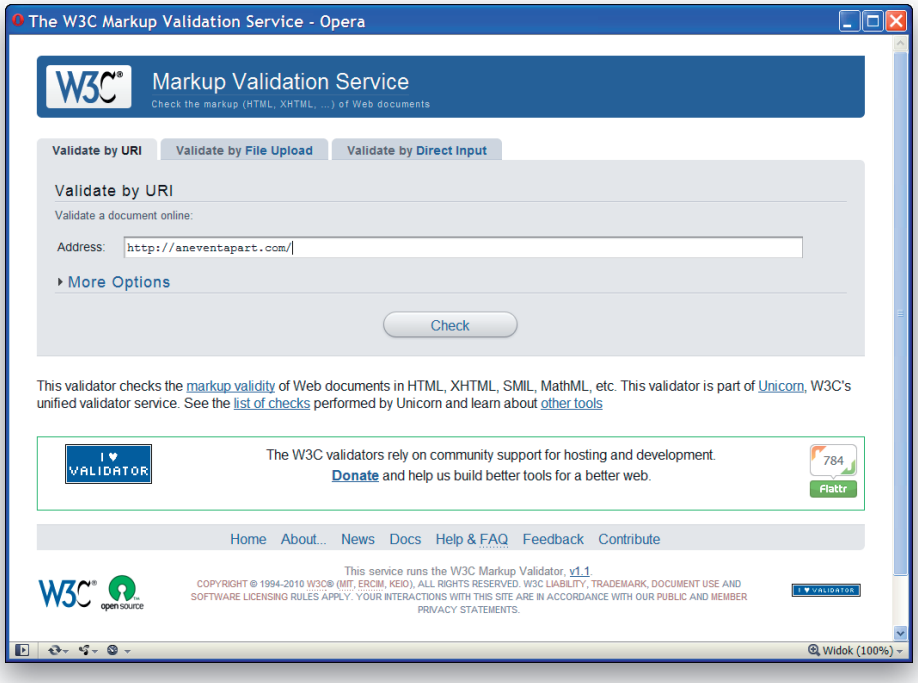
Z tego rozdziału dowiesz się, jakie znaczenie ma ustawienie odpowiedniej kolejności wartości, jak poprawnie stosować wartości bezjednostkowe, jak sprawić, aby elementy znikały, nauczysz się kontrolować wygląd obramowania elementów, sztuczek związanych z formatowaniem list, tworzyć arkusze stylów przeznaczone dla druku i wiele więcej.

SPRAWDZAJ POPRAWNOŚĆ KODU!

Może to żadna nowość dla Ciebie. Może zastanawiasz się, dlaczego marnuję cenny atrament i inne wartościowe surowce na opis tak oczywistych kwestii. Jak często w takim razie sprawdzasz poprawność kodu swoich stron? Raz na zakończenie pracy nad projektem, czy wielokrotnie w ciągu całego procesu?

Nie twierdzę, że trzeba sprawdzać stronę po każdym zapisaniu najdrobniejszych zmian, ale warto wyrobić sobie nawyk robienia tego w regularnych odstępach czasu. Dzięki temu można wykryć usterki w zarodku, zanim zdeformują całą strukturę strony.

Istnieje kilka dobrych narzędzi do sprawdzania kodu HTML i CSS. Jeśli chodzi o HTML, to najpopularniejszym narzędziem jest program udostępniany przez organizację W3C pod adresem *validator.w3.org* (rysunek 3.1). Równie popularny jest jego odpowiednik dla CSS zamieszczony pod adresem *jigsaw.w3.org/css-validator*.



Rysunek 3.1. Walidator HTML W3C

Co zrobić, jeśli jesteśmy uwięzieni za zaporą sieciową albo pracujemy na laptopie z zainstalowanym lokalnym serwerem? Wówczas należy skorzystać z funkcji sprawdzania kodu dodatku Firebug i innych tego typu narzędzi. Jeśli masz dostęp do internetu, masz też możliwość sprawdzenia kodu swoich stron, bez względu na to, czy są one publicznie dostępne, czy nie (bardzo często korzystam z możliwości sprawdzania kodu lokalnie tylko po to, by nie wyjść z wprawy w korzystaniu z tego narzędzia).

KOLEJNOŚĆ WŁASNOŚCI PISMA

Kolejność fontów w deklaracjach CSS to jedna z tych rzeczy, o które potyka się wiele osób, nie wiedząc nawet, co je przewróciło.

Większość własności CSS przyjmuje wiele słów kluczowych, które można ustawiać w dowolnej kolejności, i na dodatek nie ma wymogu, aby stosować wszystkie z nich (weźmy np. własność `background`, która przyjmuje od jednej do pięciu wartości — nie ma znaczenia, w jakiej kolejności zostaną one ustawione). Jednym z niewielu wyjątków w tej dziedzinie jest własność `font`, która ma określony minimalny zestaw wymaganych słów kluczowych i muszą one być ustawione w odpowiedniej kolejności.

Oto najprostsza możliwa deklaracja własności `font`:

```
font: <font-size> <font-family>;
```

Oczywiście nawiasy trzeba zastąpić konkretnymi wartościami, np.:

```
font: 100% sans-serif;
```

Chodzi o to, że obie te wartości muszą być podane w takiej, a nie innej kolejności — najpierw rozmiar, potem rodzina. Jeśli zmienisz ich kolejność albo pominiesz którąś z nich, każda nowoczesna przeglądarka z mety taką deklarację zignoruje.

Ponadto, jeśli chcesz użyć jeszcze innych słów kluczowych w deklaracji, musisz je (oprócz jednego, które jest tematem następnego podrozdziału) wpisać przed tymi, które są wymagane:

```
font: bold italic 100% sans-serif;
font: italic small-caps 125% Georgia, serif;
font: italic bold small-caps 200% Helvetica, Arial, sans-serif;
```

Zwróć uwagę, że wartości przed rozmiarem mogą występować w dowolnej kolejności i nie ma to żadnego znaczenia. Ważne jest tylko to, aby znajdowały się przed określeniem rozmiaru. Jeśli wpiszemy je w innym miejscu, to przeglądarki będą ignorować naszą deklarację.

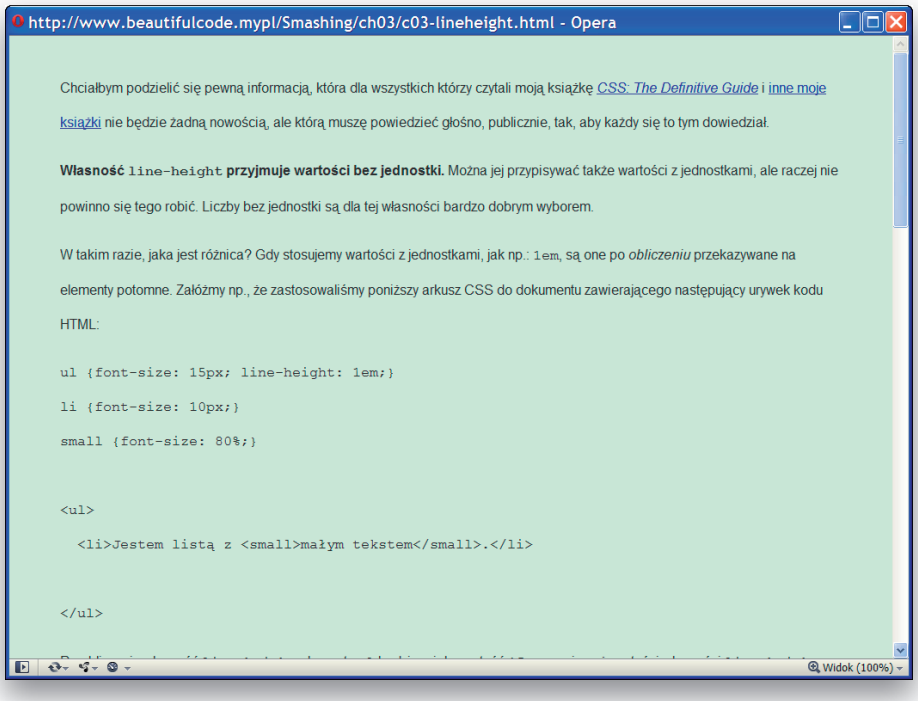
WYSOKOŚĆ LINII

Jeśli opisane wcześniej zasady dotyczące stosowania własności `font` wydawały Ci się dziwne, to to, co zobaczysz teraz, jest czystym wariactwem.

Wcześniej napisałem, że minimalna wartość własności `font` składa się z dwóch wartości w ściśle określonej kolejności — rozmiar i rodzina. To prawda, ale okazuje się, że do wartości rozmiaru można doczepić wartość wysokości linii (własności `line-height` — rysunek 3.2):

```
font: 100%/2.5 Helvetica, sans-serif;
```

Wartości własności `font-size` i `line-height` nie są oddzielone spacją, lecz ukośnikiem (co ciekawe, jest to jedyny przypadek użycia ukośnika w całej technologii CSS).



Rysunek 3.2. Zwiększona wysokość linii

Dodanie wartości własności `line-height` do deklaracji własności `font` jest nieobowiązkowe, ale jeśli już to zrobimy, to miejsce jej wstawienia nie jest dowolne. Wartość tę trzeba wstawić zaraz za wartością rozmiaru pisma i rozdzielić je ukośnikiem.

BEZJEDNOSTKOWE WARTOŚCI WŁASNOŚCI LINE-HEIGHT

Własności `line-height` można przypisywać wartości bez jednostki. Nie jest zabronione również stosowanie jednostek, ale ogólnie rzecz biorąc, jest to niezalecane.

Jaka jest różnica? Jeśli zdefiniujemy wartość z jednostką, np. `1em` lub 100%, zostanie ona po obliczeniu przekazana elementom potomnym. Załóżmy np., że zastosowaliśmy poniższy arkusz CSS do dokumentu zawierającego następujący urywek kodu HTML:

```
ul {font-size: 15px; line-height: 1em;}
li {font-size: 10px;}
small {font-size: 80%;}

<ul>
    <li>Jestem listą z <small>małym tekstem</small>.</li>
</ul>
```

Po obliczeniu własność `line-height` elementu `ul` będzie miała wartość `15px`, ponieważ wartości własności `line-height` wyrażone za pomocą jednostek `em` (i procentów) są obliczane na podstawie obliczonej wartości własności `font-size` elementu. Wartość własności `font-size` zadeklarowałem bezpośrednio, więc wiemy, że jest wyrażona w pikselach.

Teraz najciekawsze — do elementów potomnych zostanie przekazana obliczona wartość `15px`. Innymi słowy, elementy `li` i `small` odziedziczą wartość `15px`. Koniec. Nie zmienią jej, gdy zmieni się ich własny rozmiar pisma. Po prostu biorą te `15px` i używają ich, co jest równoznaczne z sytuacją, w której byśmy napisali:

```
ul {font-size: 15px; line-height: 1em;}
li {font-size: 10px; line-height: 15px;}
small {font-size: 80%; line-height: 15px;}
```

Wyobraź sobie, że usuwamy jednostkę `em` z wartości `line-height`, tak że arkusz stylów wygląda następująco:

```
ul {font-size: 15px; line-height: 1;}
li {font-size: 10px;}
small {font-size: 80%;}
<ul>
<li>Jestem listą z <small>małym tekstem</small>.</li>
</ul>
```

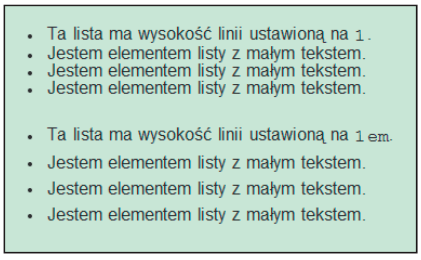
Teraz elementom potomnym (`li` i `small`) przekazywana jest sama liczba, którą wykorzystują one jako współczynnik skali — mnożnik, jeśli wolisz — a nie jako obliczony wynik.

Każdy element, który odziedziczy tę wartość `1`, pomnoży ją przez obliczoną wartość własności `font-size`. Element listy, który ma deklarację `font-size: 10px`, będzie miał obliczoną wartość `line-height: 10px`. Następnie przekaże tę jedynkę do elementu `small`, który pomnoży ją przez swoją obliczoną wartość `font-size`. To da w wyniku osiem pikseli, a więc wysokość linii wyniesie również osiem pikseli.

Ostateczny wynik będzie równoważny z poniższym:

```
ul {font-size: 15px; line-height: 1;}
li {font-size: 10px; line-height: 10px;}
small {font-size: 80%; line-height: 8px;}
```

To duża różnica (rysunek 3.3). Dlatego właśnie zaleca się, aby własności `line-height`, która ma zostać zastosowana na rzecz elementów typu `html` lub `body` (i wszystkich innych, które mogą mieć potomków), nadawać wartości bez jednostek.



Rysunek 3.3. Różnica między wartościami własności line-height z jednostką i bez jednostki

OKREŚLAJ STYL OBRAMOWANIA

Dobrze dobrane obramowanie może zwiększyć walory estetyczne każdego projektu, ale jeśli nie określimy jego stylu, nic nam z tego nie wyjdzie.

Pisząc „bez stylu", nie mam na myśli stylu CSS, lecz własność `border-style`.
Załóżmy, że napisaliśmy poniższą regułę z własnością skrótową `border`:

```
form {border: 2px gray;}
```

Świetnie, tylko że formularze nie zostaną objęte żadnym obramowaniem. Powód jest prosty — pominęliśmy wartość własności `border-style`, przez co została użyta jej wartość domyślna. Co to za wartość? Ta wartość to `none`. W związku z tym powyższa reguła jest równoważna poniższej:

```
form {border: 2px gray none;}
```

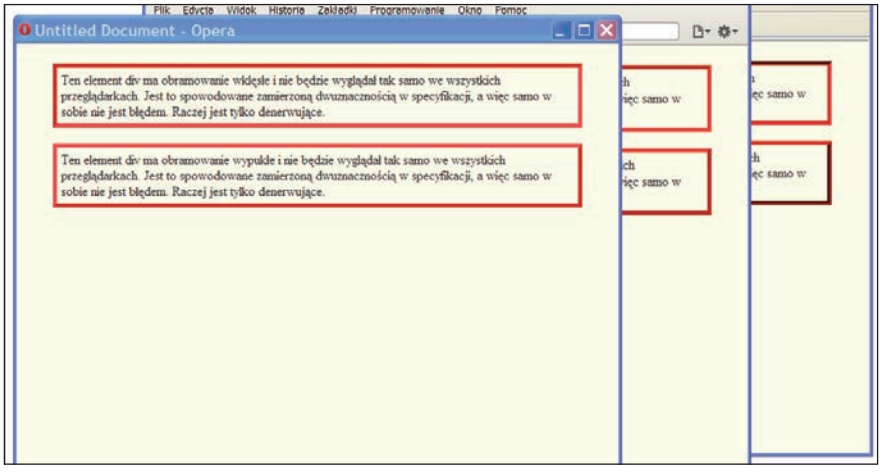
Obramowanie z własnością `border-style` ustawioną na `none` nigdy nie zostanie narysowane, bez względu na wartość własności `border-width` — obramowanie, które nie istnieje, nie może mieć żadnej grubości.

USTAWIANIE KOLORU OBRAMOWANIA

Od czasu do czasu musimy albo po prostu chcemy zastosować na stronie wypukłe (`outset`) albo wklęsłe (`inset`) obramowanie. Nie jestem tu od oceniania gustu, tylko chcę wskazać na pewien potencjalny problem, który jest z tym związany. Weźmy następującą regułę:

```
div {border: 5px red outset;}
```

Nic nadzwyczajnego, prawda? Ale spójrz, jak ta reguła zostanie zinterpretowana w różnych przeglądarkach (rysunek 3.4).



Rysunek 3.4. Różne interpretacje stylów obramowania inset i outset

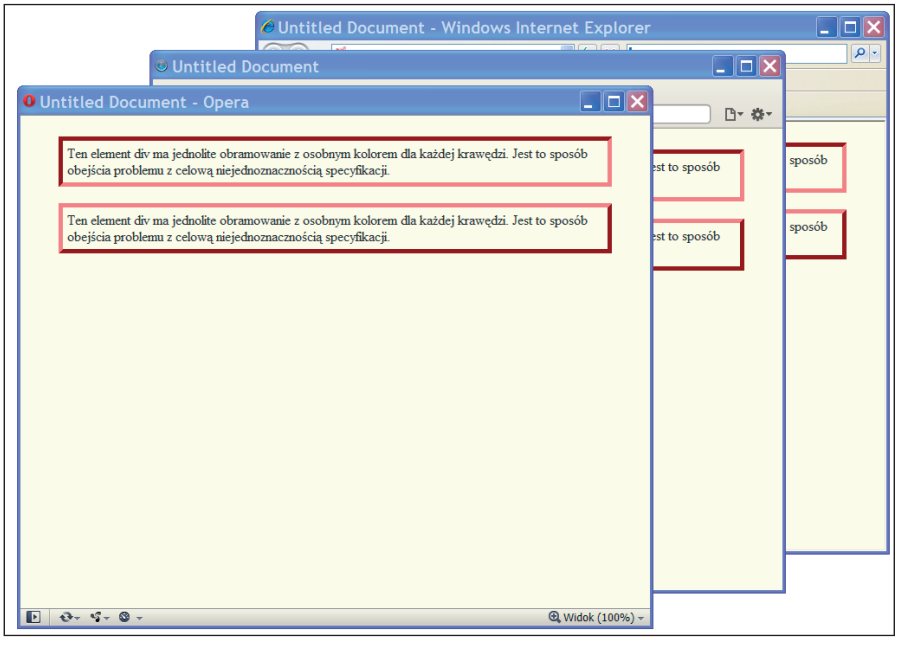
To nie jest błąd i żadna z tych przeglądarek nie interpretuje zdefiniowanych własności źle. Po prostu w specyfikacji CSS nie określono jednoznacznie, jak przeglądarki mają modyfikować kolory w celu uzyskania iluzji wklęsłości bądź wypukłości obramowania. Oto, co jest w niej napisane:

> Kolor obramowania w stylu `groove`, `ridge`, `inset` i `outset` zależy od własności koloru obramowania elementu, ale aplikacje klienckie mogą rzeczywiste kolory obliczać przy użyciu dowolnego algorytmu (*www.w3.org/TR/CSS21/box.html#border-style-properties*).

Zwróć szczególną uwagę na drugą część tej wypowiedzi: „aplikacje klienckie mogą rzeczywiste kolory obliczać przy użyciu dowolnego algorytmu". W świecie przeglądarek od zawsze jest tak, że jeśli zezwoli się na wybór różnych opcji, to każda przeglądarka wybierze inną. I tak jest też w tym przypadku.

Może nie przeszkadzają Ci te różnice w interpretacji stylu obramowania — i w porządku. Moim zadaniem nie jest ferowanie wyroków. Jeśli jednak wolisz, aby obramowanie w każdej przeglądarce wyglądało tak samo (jak na rysunku 3.5), musisz zadeklarować zwykłe obramowanie i własnoręcznie ustawić kolory:

```
#innie {border-color: #800 #F88 #F88 #800;}
#outie {border-color: #F88 #800 #800 #F88;}
```

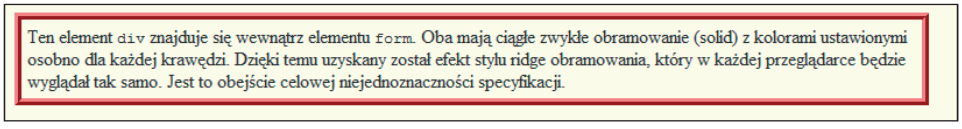



Rysunek 3.5. Takie same kolory obramowania w różnych przeglądarkach

To oczywiście dotyczy tylko stylów `inset` i `outset` obramowania. Aby utworzyć jednolite kolory dla `groove` i `ridge` (rysunek 3.6), trzeba te elementy umieścić w kontenerach (albo wstawić kontenery do elementów), obramowanie każdego z nich ustawić na normalne (`solid`) i zastosować kolory, które utworzą żądany efekt, np.:

```
#innie {border-color: #800 #F88 #F88 #800;}
form {border: 3px solid; border-color: #F88 #800 #800 #F88;}

<form>
  <div class="wrap">
    (treść, elementy formularza itp.)
  </div>
</form>
```



Rysunek 3.6. Spójne obramowania w stylu ridge

WYŁĄCZANIE WYŚWIETLANIA ELEMENTÓW

Czy kiedykolwiek chciałeś ukryć element na stronie przed użytkownikami, ale nie usuwać go z kodu źródłowego dokumentu? Można to zrobić na kilka sposobów, z których każdy ma swoje wady i zalety. Przestudiujemy je w tym podrozdziale i kilku następnych.

Najprostszym sposobem na ukrycie elementu jest wyłączenie jego wyświetlania za pomocą odpowiedniej własności CSS.

```
.hide {display: none;}
```

Ta reguła sprawi, że na stronie nie zostanie wyświetlony żaden element przypisany do klasy `hide`. Oznacza to, że elementy te nie wygenerują w układzie strony żadnego pola, które by zajmowały, a więc nie będą miały żadnego wpływu na układ. To tak, jakby elementy te w ogóle nie istniały, są jak nieuchwytny ninja.

Z używaniem własności `display: none` wiążą się jednak potencjalne problemy. Jeden z nich jest tylko potencjalny, ale drugi występuje zawsze. Potencjalny problem polega na tym, że jeśli za pomocą JavaScriptu ustawimy wartość własności `display` elementu na `none`, to nie wiadomo, jak ją później przywrócić. To jest trudniejsze, niż się wydaje. Załóżmy, że napisaliśmy poniższy kod:

```
var obj = document.getElementById('linker');
obj.style.display = 'none';
```

Jeśli później zechcemy ten element przywrócić do widoku za pomocą JS, to jakiej wartości użyjemy? To oczywiście zależy od elementu, o który nam chodzi. Jeśli będzie to `span`, to nadalibyśmy mu wartość `inline`, gdyby to był `p` — to wartość `block`. A może nie, przecież elementy `span` można zamieniać na blokowe, a elementy `div` na śródliniowe.

Istnieje jedno bardzo dobre rozwiązanie tego problemu — nie przypisywać żadnej wartości:

```
obj.style.display = '';
```

Dzięki temu element zostanie przywrócony do odpowiedniej wartości `display`, jaka będzie potrzebna w dalszej części arkusza lub w arkuszach wbudowanych przeglądarki.

Innym często stosowanym rozwiązaniem jest nieustawianie własności `display` bezpośrednio, lecz przypisanie elementu do jakiejś klasy, np. `hide`. Aby go wyświetlić, wystarczy później tylko usunąć tę klasę. To rozwiązanie jest o tyle bardziej skomplikowane, że wymaga napisania (albo znalezienia w wyszukiwarce Google) skryptu JavaScript usuwającego i dodającego wartości atrybutu `class`, niemniej jednak jest ono bardzo dobre.

Drugi problem, który występuje zawsze, jest związany z tym, że jak na razie elementy z własnością `display` ustawioną na `none` są „niewidoczne" dla technologii pomocniczych, takich jak np. czytniki ekranu. Ponieważ elementu nie widać na ekranie, czytnik go nie zobaczy — i o to zazwyczaj chodzi, ale nie zawsze.

Wyobraź sobie, że wstawiasz na stronę specjalne odnośniki pozwalające przejść bezpośrednio do głównej treści strony. Chcesz, aby łącza te służyły osobom korzystającym z czytników ekranu, pozwalając im przeskoczyć szybko do odpowiedniego miejsca w dokumencie, ale nie chcesz, aby były wyświetlane na ekranie, bo osobom przeglądającym stronę wzrokowo są niepotrzebne. Jeśli dla kontenera tych odnośników zdefiniujemy własność `display: none`, to znajdą się one poza zasięgiem wszystkich użytkowników, zarówno widomych, jak i niewidomych. Osoby, które ich potrzebują, nie usłyszą nic o ich istnieniu.

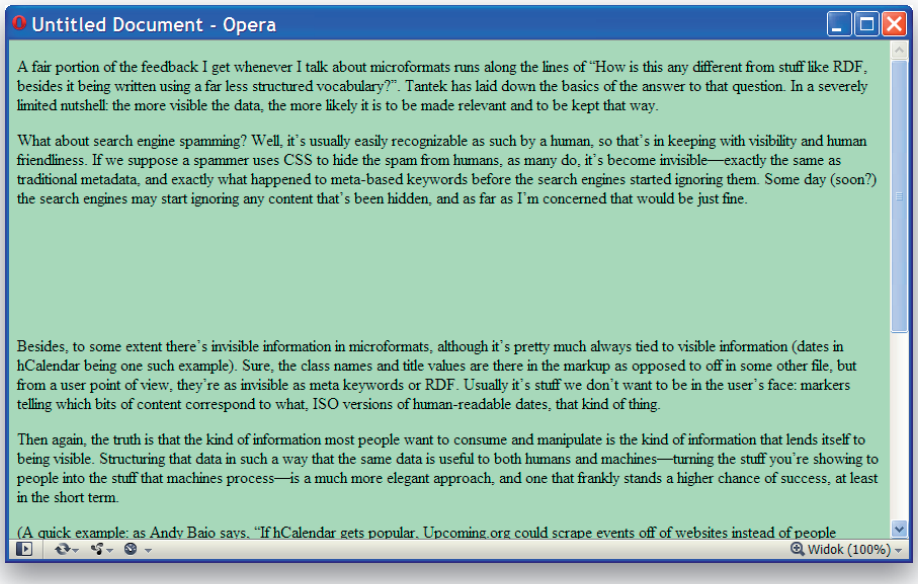
Analogicznie jest z menu rozwijanymi (bez obsługi zdarzeń myszy) ukrytymi przed widomymi użytkownikami — jeśli ustawimy im własność `display: none`, to także czytniki ekranu ich nie znajdą.

WYŁĄCZANIE WIDOCZNOŚCI ELEMENTU

Techniką bardzo podobną do własności `display` jest własność `visibility`, za której pomocą również można znieść element z widoku na ekranie:

```
.hide {visibility: hidden;}
```

Powyższa reguła sprawi, że wskazywany przez nią element będzie niewidoczny, co może się wydawać identyczne z działaniem własności `display: none`. Jest jednak między tymi własnościami ważna różnica — element z własnością `visibility: hidden` nie zostaje usunięty z układu strony, co widać na rysunku 3.7.



Rysunek 3.7. Niewidoczny element

Co dobrego oprócz zajmowania wolnego miejsca może nam przyjść z użycia niewidocznego elementu? Nie ma z nim kontaktu za pomocą myszy, nie da się do niego dostać przy użyciu klawiatury i nie widać go. Czemu w takim razie ma to służyć?

Rozwiązanie to świetnie nadaje się do formatowania elementów pozycjonowanych bezwzględnie, które i tak nie biorą udziału w normalnym układzie elementów na stronie (leżą na wierzchu i nie są uwzględniane przy rozmieszczaniu pozostałych elementów). Można więc włączać i wyłączać ich widoczność bez zmieniania układu strony. Dodatkowo można je ukrywać i wyświetlać z powrotem bez konieczności zabawiania się z własnością `display`, co pozwala uniknąć problemów opisanych w poprzednim podrozdziale.

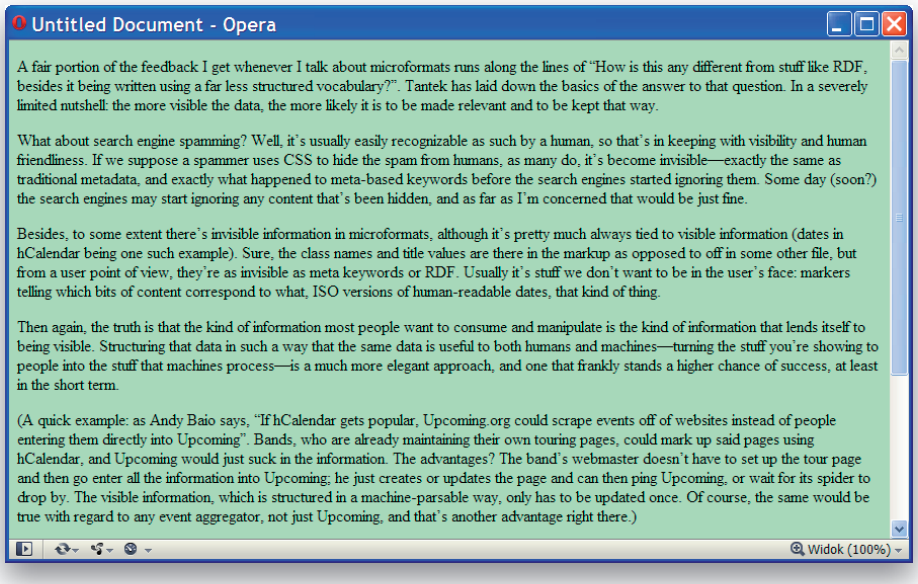
Niestety, problemy z dostępnością pozostają te same — elementy z własnością `visibility: hidden` są całkowicie ignorowane przez prawie wszystkie czytniki ekranu. Także menu rozwijane ukryte tą techniką pozostają dla nich niewidoczne.

WYRZUCANIE ELEMENTÓW POZA EKRAN

Mamy więc problem — chcemy, aby niektóre elementy były niewidoczne wizualnie, ale były dostępne dla czytników ekranu. Oto jedno z możliwych rozwiązań:

```
.hide {position: absolute; top: -10000em; left: -10000em;}
```

Dzięki tej regule trzeci akapit (ten, który powodował wystąpienie pustego miejsca na rysunku 3.7) zostanie jakby usunięty ze strony, co widać na rysunku 3.8.



Rysunek 3.8. Wyrzucanie elementów poza ekran przy użyciu pozycjonowania

Oto, jak to działa. Nasza reguła wzięła trzeci akapit, zastosowała do niego pozycjonowanie bezwzględne i wyrzuciła go daleko poza krawędź ekranu. Dzięki temu, mimo iż element zniknął z widoku wizualnie, nadal jest dostępny przynajmniej dla niektórych czytników ekranu. Dlatego właśnie technikę tę ogólnie uważa się za najlepszą do ukrywania elementów.

Z technicznego punktu widzenia przesunęliśmy lewy górny róg elementu o 10 000 em (tzn. wielkości pisma w tym elemencie) nad lewy górny róg bloku go zawierającego i 10 000 na lewo od niego. W wielu przypadkach ten blok zawierający to element główny, czyli np. `html`. Bez względu na wszystko jest praktycznie niemożliwe, żeby tak daleko wyrzucony poza ekran element był kiedykolwiek widoczny na stronie.

Możliwości przywrócenia elementu jest kilka. Jeśli chcesz, aby po uwidocznieniu był nadal bezwzględnie pozycjonowany, wystarczy, że odpowiednio zmienisz wartości jego własności `top` i `left`, np.:

```
.show {top: 0; left: 0;}
```

Jeśli jednak wolisz wdrożyć go do normalnego układu elementów na stronie, możesz przywrócić mu domyślny typ pozycjonowania.

```
.show {position: static;}
```

Przy zastosowaniu tej metody nie trzeba przywracać wartości własności `top` i `left`, ponieważ w pozycjonowaniu statycznym są one i tak ignorowane. Czy je zmienisz, czy nie, to i tak nie będzie miało żadnego znaczenia.

Jest jeszcze trzecia możliwość, która wchodzi w grę wówczas, gdy chcemy przywrócić element do układu i musi on odgrywać rolę blokowego kontenera dla elementów, które zawiera. Sytuacja taka ma miejsce np. wówczas, gdy chcemy pozycjonować bezwzględnie elementy znajdujące się w przywracanym elemencie. W takim przypadku element ten można pozycjonować względnie, ale trzeba zadeklarować wartości przesunięcia.

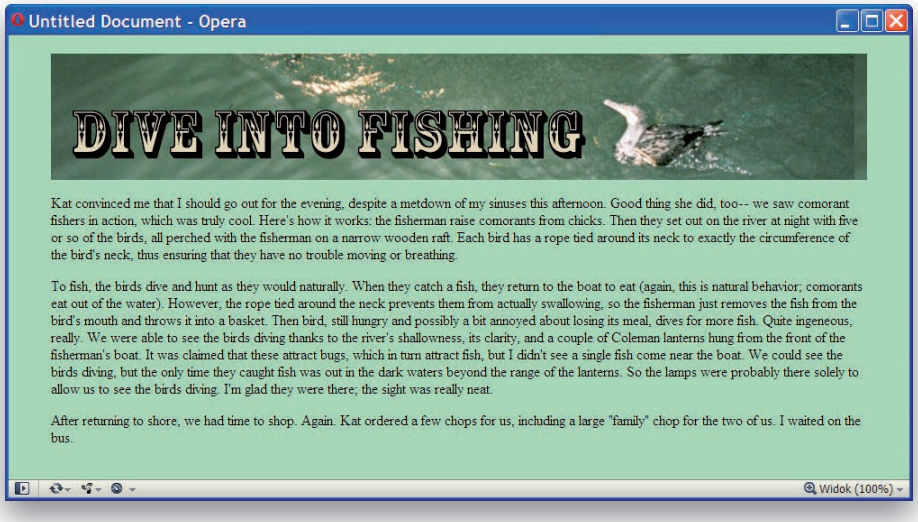
```
.show {position: relative; top: 0; left: 0;}
```

Gdybyśmy nie podali ustawień `top: 0;` i `left: 0;`, to element zostałby przesunięty względem swojego normalnego położenia w układzie normalnym. To spowodowałoby powstanie luki w miejscu, w którym by był, gdyby nie został przesunięty o te 10 000 em.

Oczywiście przesunięcie nie musi wynosić 10 000 em. Może to być dowolna wartość, nie większa od 65 535 w nielicznych starszych przeglądarkach lub 16 777 271 w Safari 3 albo 2 147 483 647 w pozostałych. Ponadto można użyć dowolnej jednostki dostępnej w CSS, a więc np. pikseli, cycerów czy cali. Ogólnie rzecz biorąc, chodzi o to, aby użyć bardzo dużej wartości, by mieć pewność, że element nie pojawi się na stronie, jeśli go celowo nie przywołamy z powrotem.

OBRAZY ZAMIAST TEKSTU

Jedną z najstarszych technik projektowych stosowanych przy użyciu CSS jest tzw. zastępowanie tekstu (ang. *image replacement* — IR). Jest to w istocie zbiór technik polegających na użyciu obrazów w miejscu, w którym powinien znajdować się tekst, tak aby napisany na grafice tekst był widoczny, dostępny w druku itd. Techniki te stosuje się najczęściej do zastępowania krótkich porcji tekstu, takich jak np. logo firmowe, nagłówki stron (rysunek 3.9) itp. Nie nadają się one natomiast do zastępowania całych akapitów.



Rysunek 3.9. Nagłówek z tekstem zastąpionym grafiką

Najpopularniejsza (tak popularna, że niektórzy robili sobie koszulki ze związanymi z nią napisami) technika IR znana jest pod dwiema nazwami: Phark i Rundle Method. Jej zasada jest prosta i polega na zwyczajnym przesunięciu tekstu w lewo za pomocą ujemnego wcięcia, jak najdalej się da.

```
h1 {height: 140px; text-indent: -9999px;
    background: url(page-header.gif);}
```

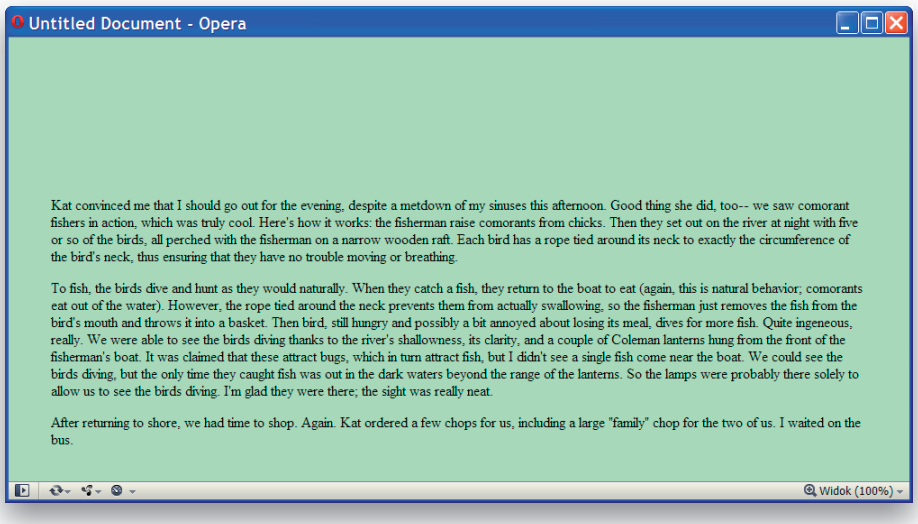
Jest to więc metoda bardzo podobna do sztuczki z ukrywaniem poza ekranem całych elementów przy użyciu pozycjonowania bezwzględnego. W tym przypadku jednak poza ekran usuwamy tylko zawartość tekstową elementu, pola samego elementu nie ruszając.

W druku tło elementów jest domyślnie nieuwzględniane. Mimo iż istnieje możliwość zmiany tego, to jednak bardzo rzadko się z niej korzysta. W związku z tym w arkuszu stylów do druku (szczegóły na ten temat znajdziesz w kolejnym podrozdziale) można po prostu napisać tak:

```
h1 {text-indent: 0; background: none;}
```

Deklaracja `background: none` jest w tym wypadku tylko zabezpieczeniem, ponieważ praktycznie nikt nie włącza opcji drukowania tła. Jednak na wszelki wypadek reguła ta uniemożliwi wydrukowanie tekstu elementu `h1` na obrazie graficznym.

Jednym z przypadków, w których ta technika nie zdaje egzaminu, jest sytuacja, gdy w przeglądarce użytkownika wyłączone jest wyświetlanie obrazów albo, co zdarza się częściej, jeśli obraz z jakiegoś powodu nie zostanie wczytany. Nagłówek będzie wtedy po prostu niewidoczny, co przyniesie skutek taki, jak widać na rysunku 3.10.



Rysunek 3.10. Efekt niedostępności obrazu

Istnieje kilkanaście rozmaitych technik zastępowania tekstu, z których każda rozwiązuje problem w inny sposób. Niektóre polegają na umieszczeniu treści elementu w elemencie `span` i wyłączeniu jego wyświetlania bądź wyrzuceniu go poza ekran. Inne natomiast polegają na dodaniu obrazu jako treści odzwierciedlającej obraz w tle.

Warto także wspomnieć o jeszcze jednej bardzo prostej technice zastępowania tekstu, którą jest po prostu wstawienie obrazu w treści. Na przykład:

```
<h1><img src="page-header.gif" alt="Dive Into Fishing"></h1>
```

Ten obraz będzie widoczny zarówno na ekranie, jak i na wydruku, ponieważ przeglądarki drukują obrazy znajdujące się w treści stron. Ponadto technika ta jest bardzo przyjazna dla czytników ekranu, które wiedzą, że w przypadku napotkania obrazu należy przeczytać zawartość ich atrybutu `alt`. Oczywiście, jeśli użytkownik wyłączy wyświetlanie obrazów, to nic z tego nie będzie. Jeśli jednak obraz nie zostanie wczytany z jakiegoś innego powodu, to zamiast niego zostanie wyświetlony przynajmniej tekst zastępczy z atrybutu `alt`.

STYLE DLA DRUKU

Jeśli nie tworzysz stylów dla druku w swojej witrynie, to może nadszedł czas, aby to jeszcze raz przemyśleć. Nawet jeśli chcesz, aby Twoje strony po wydrukowaniu wyglądały tak samo jak na ekranie, możesz skorzystać z okazji i zoptymalizować kontrast kolorów (najpewniej odcieni szarości), pamiętając, że nie będzie koloru ani obrazów tła.

To nic trudnego. Arkusz stylów dla druku możemy dołączyć do strony na jeden z trzech sposobów:

```
<style type="text/css" media="print">…</style>
<link type="text/css" rel="stylesheet" media="print" href="print.css">
@import url(print.css) print;
```

Najczęściej stosuje się metodę z użyciem elementu `link`, ponieważ osadzanie całego arkusza stylów na każdej stronie to mało efektywne rozwiązanie, a do importu arkusza stylów również potrzebne jest osadzenie arkusza stylów na każdej ze stron. Ponadto w importowaniu arkuszy stylów dla druku przez wiele lat przeszkadzały błędy przeglądarek związane z ich obsługą.

W arkuszu stylów dla druku można np. usunąć takie efekty jak zastępowanie tekstu (zobacz poprzedni podrozdział). Warto też w tym miejscu zapewnić ciemny kolor tekstu, gdyż biały tekst na czarnym tle na ekranie prawie na pewno zmieni się w druku w biały tekst na białym papierze, co bez wątpienia znacznie utrudnia odczyt.

Ma to miejsce dlatego, że obrazy zamieszczone w tle prawie nigdy nie są drukowane. Oczywiście prawie każda przeglądarka ma opcję drukowania tych grafik, ale domyślnie jest ona wyłączona i jeśli się nad tym zastanowić, jest to bardzo dobre rozwiązanie (wyobraź sobie, jak szybko ubywałoby atramentu w drukarce, gdybyśmy drukowali biały tekst na ciemnoniebieskim tle). Ponieważ prawie nikt nie zmienia tego ustawienia, należy przyjąć założenie, że w druku nie zostaną wyświetlone żadne tła. Dlatego najlepiej jest je w arkuszu dla druku zwyczajnie usunąć.

Można to zrobić ruchem zamaszystym:

```
* {background: transparent; color: black;}
```

albo wybiórczo wyznaczyć elementy, które wymagają poprawki:

```
body, #navbar, #aside, .warning, .blockquote {
        background: transparent; color: black;}
```

PISANIE ARKUSZY STYLÓW DLA DRUKU

Jaki jest najlepszy sposób na pracę nad arkuszami stylów dla druku? Bezpośrednio w przeglądarce, chyba że wolisz skorzystać jakiś milion razy z opcji podglądu wydruku. Już wyjaśniam, o co chodzi.

Zapewne masz już jeden czy dwa arkusze stylów z definicją układu elementów na stronie. Załóżmy, że są dołączone do strony w następujący sposób:

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

Mimo iż nie jest to nigdzie bezpośrednio napisane, oba te arkusze mają zastosowanie do wszystkich typów mediów. Innymi słowy, powyższy zapis jest równoznaczny z dodaniem atrybutu `media="all"`.

```
<link type="text/css" rel="stylesheet" href="basic.css" media="all">
<link type="text/css" rel="stylesheet" href="theme.css" media="all">
```

Czy na pewno chcesz, aby te wszystkie style zostały zastosowane w druku? Jeśli nie, to wartość `all` lepiej zmienić na `screen`.

```
<link type="text/css" rel="stylesheet" href="basic.css" media="screen">
<link type="text/css" rel="stylesheet" href="theme.css" media="screen">
```

Mamy już obsłużoną domyślną sytuację. Teraz musimy jeszcze dodać arkusz stylów dla druku:

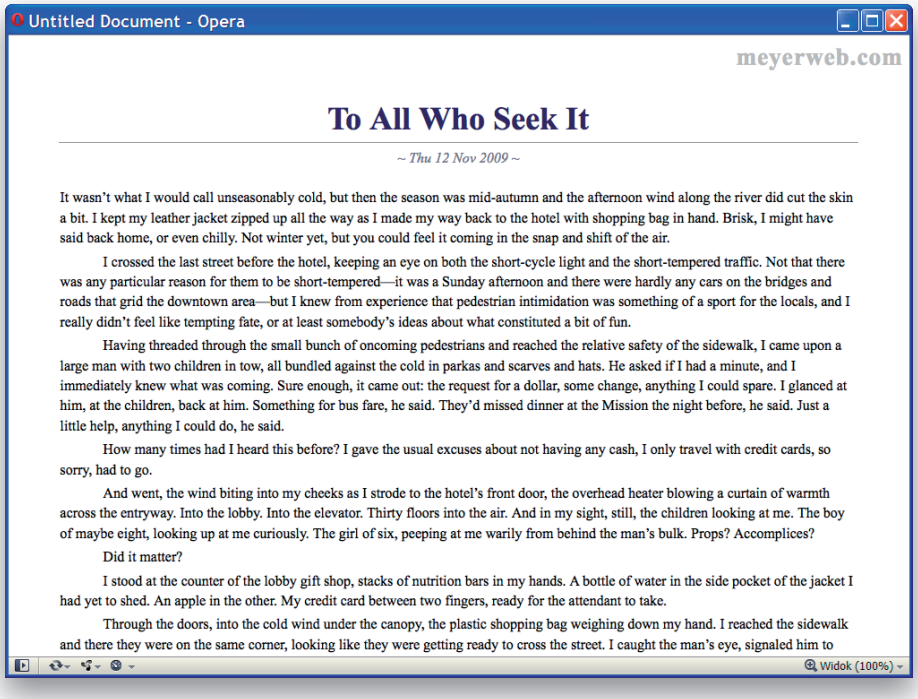
```
<link type="text/css" rel="stylesheet" href="basic.css" media="screen">
<link type="text/css" rel="stylesheet" href="theme.css" media="screen">
<link type="text/css" rel="stylesheet" href="print.css" media="print">
```

Świetnie! Tylko że po odświeżeniu strony nadal widzimy to samo, ponieważ korzystamy z medium typu `screen`. Ponieważ nie chcielibyśmy w nieskończoność włączać i wyłączać podglądu wydruku ani też drukować strony po naniesieniu każdej poprawki w arkuszu, musimy zmusić stronę, aby wyświetlała się na ekranie tak, jakby była drukowana.

W zasadzie już w poprzednim akapicie wyjaśniłem, jak to zrobić. Wystarczy arkusz stylów przeznaczony dla druku oznaczyć tymczasowo jako arkusz dla ekranu, a rzeczywiste arkusze dla ekranu — jako przeznaczone dla jakiegoś innego rodzaju medium

(rysunek 3.11). Ja np. używam typu medium `tty`, ponieważ jest chyba najmniej podobne do ekranu ze wszystkich możliwych, a ponadto ma krótką nazwę. Oto przykład:

```
<link type="text/css" rel="stylesheet" href="basic.css" media="tty">
<link type="text/css" rel="stylesheet" href="theme.css" media="tty">
<link type="text/css" rel="stylesheet" href="print.css" media="screen">
```



Rysunek 3.11. Podgląd stylu druku w przeglądarce

Teraz możesz sobie używać do woli, poprawiać i odświeżać aż do uzyskania takiego efektu, jaki sobie wymarzyłeś. Gdy skończysz pracę, zmień wartość `screen` z powrotem na `print`, a `tty` na `screen` i dokonaj jednego wydruku, aby się upewnić, że wszystko jest w porządku.

ŁĄCZA BLOKOWE

Podstawową techniką pozwalającą uzyskać wiele ciekawych opcji układu elementów na stronie przy niewielkim nakładzie kodu źródłowego są zabawy z własnością `display`. Dżokerem w tym rozdaniu jest możliwość zamiany łączy, które są śródliniowe, na elementy blokowe.

Aby zrozumieć, dlaczego tak jest, wyobraź sobie listę odnośników w pasku bocznym strony. Najczęściej umieszcza się je na liście nienumerowanej, w której każde łącze zawiera się w jednym elemencie tej listy. Porównajmy więc dwie identyczne listy łączy. Jedyna różnica

między nimi będzie polegała na tym, że jedna będzie utworzona na bazie odnośników blokowych, a druga — nie (rysunek 3.12). Specjalnie oznaczyłem kolorem tło odnośników, aby pokazać, na czym polega różnica.

Rysunek 3.12. Dwie listy odnośników — blokowych i nieblokowych

Odnośniki śródliniowe są znacznie mniej wygodne w użyciu, gdyż oferują zdecydowanie mniejsze pole do klikania. Co więcej, gdybyśmy chcieli utworzyć efekt podświetlania tła po najechaniu na łącze kursorem, w przypadku odnośników śródliniowych zmieniłoby się tylko tło pod tekstem, a nie całego pola.

Aby zmienić odnośnik na blokowy, nie trzeba dużo pracy:

```
#sidebar ul a {display: block;}
```

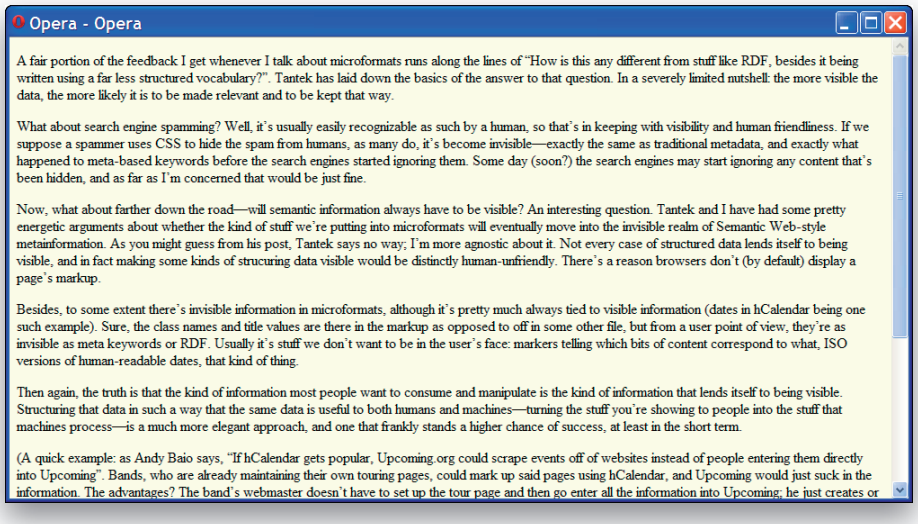
Ta reguła została użyta do zmiany odnośników na rysunku 3.12.

Gdy łącze wygeneruje pole blokowe, zachowuje się ono identycznie jak zwykłe pola blokowe generowane przez akapity, nagłówki, elementy `div` itp., ponieważ jest to w istocie ten sam rodzaj pola. Można definiować marginesy, dopełnienie i wszystkie inne własności elementów blokowych.

MARGINES CZY DOPEŁNIENIE?

Czy kiedykolwiek zastanawiałeś się — tak naprawdę intensywnie — nad wcięciami list albo tą niewielką, pustą przestrzenią, która jest tworzona domyślnie wokół każdej strony? Jeśli tak, to wiesz może, skąd się one biorą? Jak się okazuje, przyczyny ich powstawania nie zawsze są takie same.

Zacznijmy od pustej przestrzeni wokół strony. Jak wiadomo, treść strony jest oddzielona od krawędzi przeglądarki około 8-pikselowym odstępem, jak widać na rysunku 3.13. Przestrzeń tę można zlikwidować za pomocą stylów resetujących albo poprzez nadanie odpowiedniego stylu elementowi `body`. Jak należy to zrobić? Czy mamy usunąć margines, czy dopełnienie?



Rysunek 3.13. Pusta przestrzeń wokół treści strony

Jeśli chcesz przypodobać się wszystkim przeglądarkom, to musisz zrobić i jedno, i drugie, ponieważ większość z nich tworzy ten odstęp za pomocą marginesu, ale Opera używa do tego celu dopełnienia.

Zanim zaczniesz szukać starożytnych norweskich zaklęć, musisz sobie zdać sprawę z tego, że twórcy przeglądarek nie popełnili w tej kwestii błędu. Nie ma specyfikacji, w której szczegółowo opisano by sposób tworzenia odstępu między krawędzią okna przeglądarki a treścią stron internetowych (ani nawet takiej, w której by napisano, że w ogóle musi on występować). Istnieje ważny argument za tym, aby w tej roli stosować dopełnienie. Nie ma to jednak znaczenia, gdyż przeglądarki stosują różne rozwiązania. Zatem piszmy:

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

To rozwiąże problem w każdej znanej przeglądarce (może oprócz Netscape 4, ale kogo to obchodzi).

Analogicznie wcięcia w listach zostały w niektórych przeglądarkach utworzone za pomocą marginesów, a w innych — przy użyciu dopełnienia. Jeśli więc napiszemy

```
ul, ol {margin-left: 0;}
```

to usuniemy wcięcie list tylko w niektórych przeglądarkach. Jeśli chcemy uzyskać ten sam efekt wszędzie, musimy też usunąć lewą stronę dopełnienia.

```
ul, ol {margin-left: 0; padding-left: 0;}
```

Oczywiście wcięcia nie musimy wcale usuwać. Gdy już usuniesz ustawienia zarówno lewego marginesu, jak i dopełnienia w celu zmiany wcięcia list, możesz później ustawić dowolne ich wartości. Możesz np. definiować wcięcie list za pomocą dopełnienia. Wówczas wystarczy napisać:

```
ul, ol {margin-left: 0; padding-left: 2.5em;}
```

Możesz też rozłożyć tę wartość na obie własności:

```
ul, ol {margin-left: 1.25em; padding-left: 1.25em;}
```

Niektórzy preferują korzystanie z samych marginesów:

```
ul, ol {margin-left: 2.5em; padding-left: 0;}
```

W wielu przypadkach nie ma znaczenia, którą metodę zastosujemy, ale wszystko się zmieni, gdy zdefiniujemy tło naszych list (rysunek 3.14). Ma to związek z tym, że markery elementów list — punktory, kwadraty, litery, liczby itp. — są umieszczane obok elementów, tak jakby były pozycjonowane bezwzględnie (w istocie nie używa się do tego celu pozycjonowania bezwzględnego, ale efekt jest bardzo podobny). Jeśli będziemy chcieli, aby markery elementów listy znalazły się „w obrębie" widocznego tła, to wcięcie musimy ustalić przy użyciu dopełnienia. Jeśli natomiast chcemy, aby były poza tłem — powinniśmy zastosować margines.

Rysunek 3.14. Porównanie różnych rodzajów wcięcia list

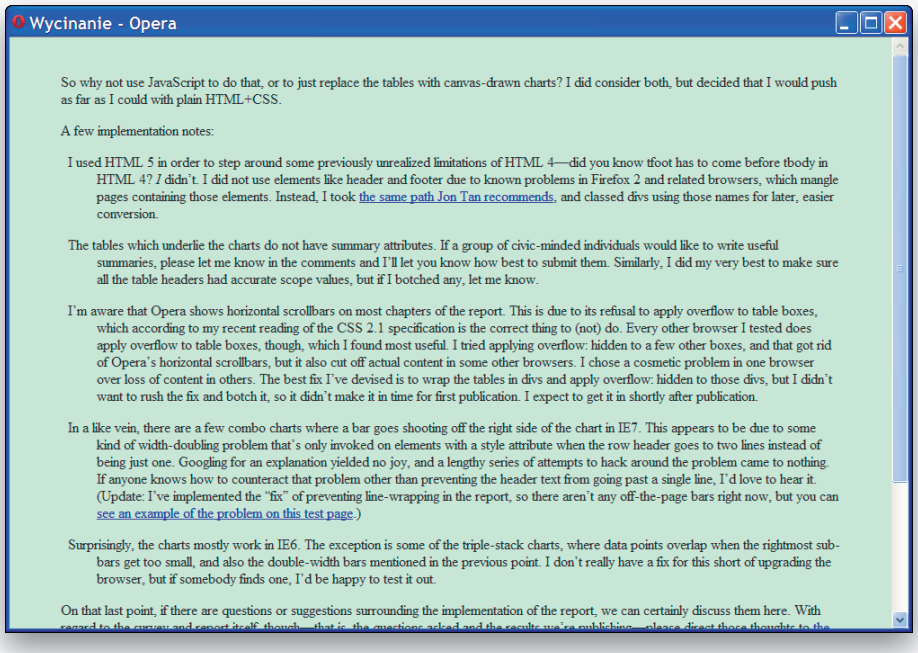
WYCINANIE LIST

W poprzednim podrozdziale była mowa o wcinaniu list, a teraz będzie o — wycinaniu. Czy pojęcie „wycinanie list" w ogóle funkcjonuje? Może i nie, ale i tak jest lepsze niż „wiszące wcięcie", które również można czasami spotkać w opisach tego typu rzeczy, a które jest kompletnie bez sensu.

To, o czym teraz mówimy, to technika wysuwania pierwszego wiersza elementu listy w lewo, tak że „wisi" po lewej stronie reszty treści elementu (rysunek 3.15).

Ten przyjemny dla oka efekt pozwala wyróżnić wizualnie listy bez zaśmiecania strony niepotrzebnymi punktorami czy czymś podobnym. Ponadto naprawdę łatwo można go uzyskać:

```
ul {text-indent: -2em; list-style: none;}
```



Rysunek 3.15. Wycięcie

To wszystko. Zwróć uwagę na deklarację `list-style: none`. Gdybym jej nie wpisał, to pierwszy wiersz każdego elementu listy byłby wycięty, ale jego tekst nachodziłby na punktory. Dlatego nie należy mieszać wycinania list z markerami.

Wycinać można oczywiście wszystko, co się chce, a więc także akapity, nagłówki, elementy `div` i `pre` czy komórki tabeli. Tylko że w listach technika ta znajduje najczęstsze zastosowanie.

DEFINIOWANIE PUNKTORÓW LIST

Punktory do listy można dodać na wiele sposobów. Najprostszym i zarazem najmniej precyzyjnym z nich jest użycie własności stylu listy kaskadowych arkuszy stylów.

Załóżmy, że mamy listę gwiazd znajdujących się najbliżej Ziemi i chcielibyśmy, aby każdy element tej listy był oznaczony małą gwiazdką zamiast kółkiem, prostokątem czy dyskiem (rysunek 3.16).

```
ul.stars{list-style-image: url(star.gif);}
```

Łatwizna. Potencjalny problem z tym związany polega na tym, że nie mamy żadnej kontroli nad położeniem tych obrazów. Ich odległość od lewego brzegu tekstu i wyrównanie w pionie względem pierwszego wiersza całkowicie zależą od przeglądarki i nie mamy tu nic do powiedzenia.

Teraz załóżmy, że chcemy użyć zwykłych markerów listy — powiedzmy typu `disc` — ale chcemy, aby miały inny kolor niż treść elementów (rysunek 3.17).

- ★ Słońce
- ★ V645 Centauri (Proxima Centauri)
- ★ Alpha Centauri A
- ★ Alpha Centauri B
- ★ Gwiazda Bernarda
- ★ Wolf 359
- ★ Lalande 21185
- ★ Sirius A
- ★ Sirius B
- ★ Luyten 726-8 A
- ★ Luyten 726-8 B
- ★ Ross 154
- ★ Ross 248
- ★ Epsilon Eridani
- ★ Lacaille 9352
- ★ Ross 128
- ★ EZ Aquarii A
- ★ EZ Aquarii B
- ★ EZ Aquarii C
- ★ Procjon A
- ★ Procjon B

Rysunek 3.16. Gwiazdy gwiazd

- Słońce
- V645 Centauri (Proxima Centauri)
- Alpha Centauri A
- Alpha Centauri B
- Gwiazda Bernarda
- Wolf 359
- Lalande 21185
- Sirius A
- Sirius B
- Luyten 726-8 A
- Luyten 726-8 B
- Ross 154
- Ross 248
- Epsilon Eridani
- Lacaille 9352
- Ross 128
- EZ Aquarii A
- EZ Aquarii B
- EZ Aquarii C
- Procjon A
- Procjon B

Rysunek 3.17. Zmiana koloru markerów

To niestety wymaga zastosowania pewnej sztuczki. Każdy element listy musimy umieścić w jakimś elemencie — np. `div` albo `span`. Zademonstruję to na przykładzie tego pierwszego elementu.

```
ul.stars {color: red; list-style: disc;}
ul.stars div {color: black;}

<ul class="stars">
<li><div>Słońce</div></li>
<li><div>V645 Centauri (Proxima Centauri)</div></li>
<li><div>Alpha Centauri A</div></li>
...
</ul>
```

W tym konkretnym przypadku moglibyśmy zamienić element `div` na `span` bez wpływu na ostateczny efekt. Ale gdybyśmy chcieli dodać jeszcze jakieś obramowanie albo tło, to różnice w zastosowaniu tych dwóch elementów byłyby ogromne (zgoda, można by je zlikwidować przy użyciu własności `display`).

Jeśli sądzisz, że w CSS powinny być jakieś sposoby zmieniania stylu samych markerów list bez konieczności wstawiania dodatkowych znaczników do dokumentów, to poinformuję Cię, że one istnieją, tylko że nie zaimplementowano ich w żadnej przeglądarce.

PUNKTORY W TLE

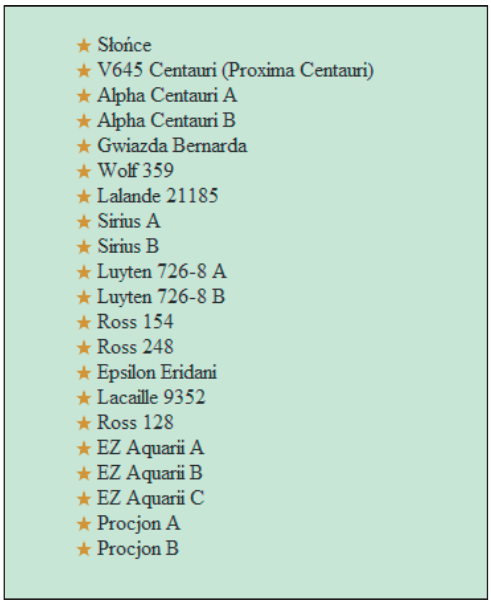
Chcemy zatem zastosować własne markery do elementów listy, ale nie podoba nam się to, że przeglądarka umieści je, gdzie zechce. Nie ma problemu. Możesz wyłączyć standardowe wyświetlanie markerów i umieścić swoje obrazki w tle elementów listy (rysunek 3.18).

```
ul.stars {list-style: none;}
ul.stars li {background: url(star.gif) 0 0.1em no-repeat; padding-left: 16px;}
```

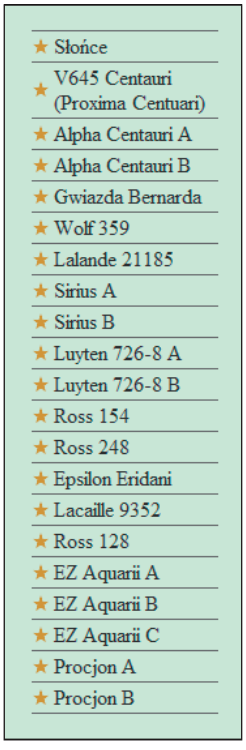
Dzięki temu, że nad położeniem obrazu w tle elementu mamy o wiele większą kontrolę, ta technika daje znacznie większe możliwości niż użycie standardowej własności `list-style-image`. Oczywiście trzeba pamiętać o dodaniu dopełnienia z lewej strony, aby treść elementu nie została wyświetlona na obrazie punktora!

Wielką sztuką jest wyrównanie obrazu z pierwszym wierszem tekstu elementu. W zasadzie nie można mieć 100% pewności, że obraz pozostanie idealnie wyrównany co do jednego piksela np. z linią podstawową pisma pierwszej linii tekstu. Można uzyskać efekt bardzo zbliżony do ideału i w wielu przypadkach uda się go osiągnąć, ale nigdy nie można być tego pewnym. Jest to jeden z przypadków, kiedy musimy zaakceptować możliwość wystąpienia drobnych usterek albo zastosować inne rozwiązanie.

Największą zaletą tej metody jest to, że wcale nas ona nie ogranicza do pierwszego wiersza tekstu. Markery będące w tle można wyrównać w pionie względem całego elementu, nawet jeśli jego treść składa się z wielu wierszy. Jeśli tę technikę połączy się z odpowiednio zdefiniowanymi krawędziami obramowania, to można uzyskać naprawdę bardzo fajne efekty (rysunek 3.19).



Rysunek 3.18. Punktory w tle

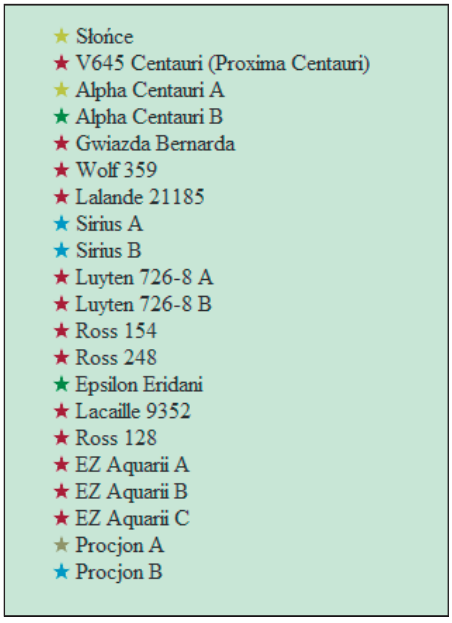


Rysunek 3.19. Markery w tle wyrównane w pionie

Jeśli chcesz różne rodzaje elementów listy oznaczyć różnymi rodzajami markerów (jak na rysunku 3.20), to wystarczy, że oznaczysz je odpowiednimi klasami i dla każdej klasy zdefiniujesz inny obraz.

```
ul.stars {list-style: none;}
ul.stars li {background: 0 0.1em no-repeat;
    padding-left: 16px;}
ul.stars li.m {background-image: url(star-m.gif);}
ul.stars li.k {background-image: url(star-k.gif);}

<ul class="stars">
<li class="g">Słońce</li>
<li class="m">V645 Centauri (Proxima Centauri)</li>
<li class="g">Alpha Centauri A</li>
<li class="k">Alpha Centauri B</li>
...
</ul>
```



Rysunek 3.20. Różne markery w tle

Jedną z wad tego podejścia jest to, że markery umieszczone w tle elementów prawie nikomu się nie wydrukują. Dlatego w arkuszu stylów dla druku warto je zastąpić zwykłymi markerami.

GENEROWANIE MARKERÓW

Istnieje jeszcze jeden bardziej zaawansowany sposób na utworzenie własnych markerów list, ale nie działa on w starszych przeglądarkach. W technice tej wykorzystuje się połączenie „wycinania" z generowaniem treści (rysunek 3.21).

Rysunek 3.21. Generowanie markerów

```
ul.stars li:before {content: url(star.gif);margin-right: 8px;}
ul.stars li {text-indent: -20px; list-style: none;}
```

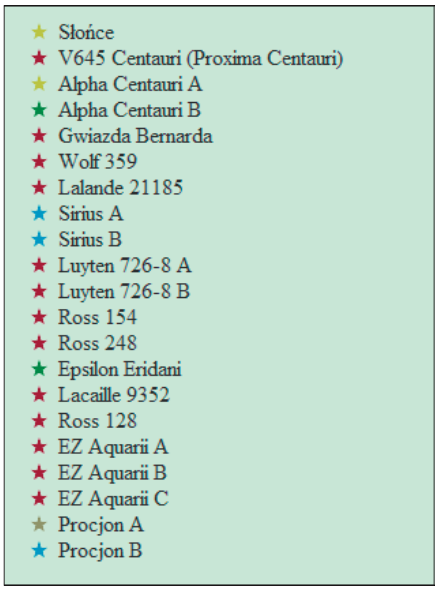
To wszystko. Nie trzeba dodawać żadnych nowych elementów, gdyż są one dodawane na początku każdego elementu listy automatycznie w postaci treści generowanej. Treść generowana oznacza, że obrazy są wstawiane jako elementy śródliniowe, a więc można je wyrównywać pionowo względem linii podstawowej pisma itp.

Oczywiście można zastosować różne ikony, odwołując się do różnych klas (rysunek 3.22).

```
ul.stars li.m:before {content: url(star-m.gif);}
ul.stars li.k:before {content: url(star-k.gif);}
```

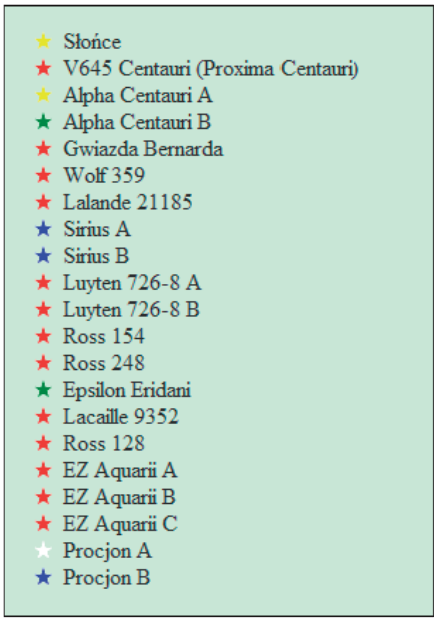
Dzięki temu, że te markery są wstawiane do treści strony, będą widoczne także na wydruku, tak jakby zostały dodane za pomocą elementu `img` albo własności `list-style-image`.

Korzyść z tego jest taka, że zamiast wczytywać obrazy, można wstawiać zwykłe znaki, które można formatować niezależnie od treści, a ponadto nie trzeba dodawać żadnych nowych elementów HTML. Wcześniejsze style możemy zastąpić poniższym arkuszem (wynik jego działania pokazano na rysunku 3.23):



Rysunek 3.22. Generowanie różnych markerów


```
ul.stars li {text-indent: -1.25em; list-style: none;}
ul.stars li:before {content: "\2605"; margin-right: 0.75em;}
ul.stars li.m:before {color: red;}
ul.stars li.k:before {color: orange;}
```



Rysunek 3.23. Generowanie markerów Unicode

Ta technika nie daje tak dużej kontroli nad markerami jak obrazy w tle, przez co istnieje ryzyko, że tekst w pierwszej linii nie będzie idealnie wyrównany z tekstem w dalszych linijkach. Ogólnie jednak rzecz biorąc, można uzyskać bardzo dobry efekt, a poza tym problem ten dotyczy tylko elementów list zawierających więcej niż jeden wiersz tekstu.

MASZ DO DYSPOZYCJI WIĘCEJ KONTENERÓW, NIŻ MYŚLISZ

Bardzo często stosowanym zabiegiem jest umieszczanie całej treści strony w elemencie-kontenerze `div` w taki sposób:

```
<body>
<div class="wrapper">
…
</div>
</body>
```

Najczęściej tłumaczy się to tym, że ułatwia to ustawienie treści na środku okna albo zdefiniowanie jeszcze kilku innych kontenerów nienależących do treści. W tym przykładzie mamy elementy `body` i `div`, dla których typowo definiuje się następujące style:

```
body {background: #ABACAB; text-align: center;}
div.wrapper {width: 800px; margin: 0 auto; text-align: left;)
```


To klasyczny kod ustawiający projekt na środku okna przeglądarki i działający nawet w starych przeglądarkach IE, które nie rozpoznawały wartości `auto` marginesów i myślały, że własność `text-align` służy do wypośrodkowywania bloków.

Ale przecież treść strony już przed dodaniem elementu `div` znajdowała się w dwóch kontenerach — elementach `body` i `html`. Tak, element `html` również można formatować. Czemu by nie? Dla CSS jest to zwykły element, jak wszystkie inne. Nie ma w nim nic niezwykłego oprócz faktu, że stoi na najwyższym poziomie hierarchii w drzewie dokumentu, a więc jest jego „korzeniem".

W związku z tym powyższe dwie reguły możemy zmodyfikować następująco:

```
html {background: #ABACAB; text-align: center;}
body {width: 800px; margin: 0 auto; text-align: left;)
```

Teraz ten sam efekt (rysunek 3.24) uzyskaliśmy bez dodatkowego elementu `div`, który możemy usunąć.

Rysunek 3.24. Układ

Gdy już zdasz sobie sprawę, że za pomocą CSS można formatować także elementy `html` i `body`, otworzą się przed Tobą nowe interesujące możliwości. Załóżmy np. że chcemy zaprojektować stronę z dwoma kolorowymi paskami wzdłuż górnej krawędzi okna i logo umieszczonym na dolnym pasku. Pewnie dałoby się to jakoś zrobić przy użyciu dodatkowego elementu `div`, ale poniżej przedstawiam alternatywne rozwiązanie, które nie wymaga dodawania tego elementu (rysunek 3.25):

```
html {border-top: 5px solid navy;}
body {border-top: 55px solid silver; margin: 0; padding: 0;}
img.logo {position: absolute; top: 10px; left: 10px;}
```

Podobny efekt można oczywiście uzyskać przy użyciu zwielokrotniania obrazów w tle (rysunek 3.26).

```
html {background: url(stars-m.png) 14px 41px repeat-y;}
body {background: url(stars-k.png) 54px -20px repeat-x;}
img.logo {position: absolute; top: 10px; left: 10px;}
```

Rysunek 3.25. Paski i gwiazdki

Rysunek 3.26. Dużo gwiazd

TŁA DOKUMENTU

Wszyscy przyzwyczailiśmy się do tego, że aby wypełnić tłem całe okno przeglądarki, ustawiamy tło elementu body. Ale nie zgadniesz, co się stanie, gdy dodatkowo ustawisz tło elementu html.

```
html {background: #ABACAB;}
body {background: #DED;}
```

Spójrz na rysunek 3.27. Okno przeglądarki wypełnia tło elementu html, a tło elementu body jest widoczne tylko w obszarze treści i jego dopełnienia. Dzieje się tak bez względu na to, czy element body sięga dolną krawędzią do dolnej krawędzi okna przeglądarki, czy nie. Jeśli nie, to pod spodem widać tło elementu html. To samo by było, gdyby element body miał ustawiony dolny margines, a element html miał dolne dopełnienie, albo gdyby oba te warunki były spełnione.

Rysunek 3.27. Element body nie zawsze wypełnia widoczny obszar okna

Jeśli usuniemy z arkusza stylów regułę definiującą tło elementu html, to całe okno zapełni się kolorem elementu body.

Dzieje się tak dlatego, że w specyfikacji języka HTML zapisano, iż kanwa, czyli obszar, w którym rysowana jest strona internetowa, pobiera swoje tło z elementu html. Jeśli element html nie ma zdefiniowanego tła, to jest ono pobierane z elementu body. Jeśli element body również nie ma tła, to zostaje zastosowany jakiś domyślny kolor.

Jest to specjalny przypadek, którego opis znajduje się w specyfikacji. W przypadku innych elementów nie jest tak, że tło (lub jakakolwiek inna własność CSS) jest przekazywane w górę drzewa dokumentu. Pamiętaj o tym, gdy będziesz ustawiać tło dla elementu `html`.

ARKUSZE SERWEROWE

Ile razy zdarzyła Ci się następująca sytuacja:

1. Wprowadzasz zmiany w arkuszu stylów na swoim komputerze.
2. Wysyłasz zmodyfikowany arkusz na serwer testowy.
3. Przechodzisz do okna przeglądarki i odświeżasz je.
4. Nic się nie zmienia.
5. Robisz twarde odświeżenie i dalej nic.
6. Sprawdzasz, czy wysyłanie plików zakończyło się powodzeniem.
7. Jeszcze raz odświeżasz i nadal nic.
8. Dodajesz kilka reguł `!important`. Wysyłasz plik na serwer, odświeżasz, nic.
9. Zaczynasz przeklinać swój komputer.
10. Sprawdzasz w Firebugu, co przesłania Twoje nowe style. Spostrzegasz, że w ogóle nie zostały zastosowane.
11. Szukasz przyczyn jeszcze kilka minut, aż w końcu odkrywasz, że odświeżałeś stronę na serwerze produkcyjnym, a nie testowym.
12. Przełączasz się na serwer testowy i widzisz wszystkie zmiany.
13. Klniesz ze złości, że jesteś taki głupi.

Takie sytuacje przydarzały mi się tak często, że aż wstyd się przyznać. Ostatnim razem, gdy to miało miejsce, w końcu zdałem sobie sprawę, że mógłbym przecież na serwerze testowym serwować specjalny arkusz stylów, taki, który pozwalałby od razu się zorientować, że jestem na tym serwerze, bez potrzeby demolowania całego projektu. To pozwoliłoby mi zaoszczędzić mnóstwo nerwów.

```
html {background: url(staging-bg.png) 100% 50% repeat-y;}
```

Istnieje wiele sposobów na zrealizowanie tego pomysłu. Najelegantsza metoda polega na użyciu nagłówków HTTP do wysyłania specjalnego, dodatkowego arkusza stylów. Jeśli korzystasz z serwera Apache, wystarczy, że dodasz do pliku *.htaccess* w katalogu głównym poniższy wiersz kodu:

```
Header add Link "</staging.css>;rel=stylesheet;type=text/css"
```

Teraz musisz jeszcze tylko zapisać w katalogu głównym swojego serwera testowego plik *staging.css* i gotowe. Oczywiście plik ten możesz zapisać w dowolnym innym miejscu, tylko pamiętaj, aby odpowiednio zmienić ścieżkę w nawiasach ostrokątnych w swojej dyrektywie. Można także stosować adresy bezwzględne, typu *http://example.com/staging.css*. Nie zapomnij tylko o nawiasach ostrokątnych, bo są wymagane.

Zawsze istnieje ryzyko, że przeniesiesz pliki *staging.css* i *.htaccess* na serwer produkcyjny. Jeśli się tego obawiasz, zrezygnuj z pliku *.htaccess* i zamiast tego wysyłaj plik *staging.css* za pomocą odpowiednich dyrektyw w pliku *httpd.conf*:

```
<Directory /ścieżka/do/katalogu/głównego/witryny>
Header add Link "</staging.css>;rel=stylesheet;type=text/css"
</Directory>
```

Ścieżkę `/ścieżka/do/katalogu/głównego/witryny` należy zastąpić właściwą własną ścieżką do katalogu, w którym znajdują się pliki Twojej witryny. Zaletą tej metody jest to, że ryzyko skopiowania pliku *httpd.conf* na inny serwer jest znacznie mniejsze. Nie jest to niemożliwe, ale bardzo mało prawdopodobne.

Jedną z wad serwowania arkuszy stylów za pomocą nagłówków HTTP jest fakt, że metoda ta nie działa w przeglądarkach Internet Explorer i Safari. Dlatego właśnie technika ta jest rzadko stosowana do serwowania CSS w publicznie dostępnych witrynach internetowych. Nie ma oczywiście żadnych przeciwwskazań do korzystania z niej w środowisku testowym, jeśli używamy np. Firefoksa albo Opery.

Teraz wyobraź sobie, że nie korzystasz z serwera Apache albo nie masz możliwości grzebania w jego plikach konfiguracyjnych, a chcesz zastosować opisywaną technikę.

Użytkownicy serwera IIS mogą wysyłać arkusze stylów poprzez nagłówki HTTP, korzystając ze wskazówek podanych na stronie *http://technet.microsoft.com/en-us/library/cc753133(WS.10).aspx*. Można to zrobić zarówno z poziomu interfejsu IIS Managera, jak i wiersza poleceń.

Jeśli wszystkie swoje strony zapisujesz w formacie PHP, to nie musisz bawić się w konfigurację serwera, ale na każdej stronie, na której chcesz włączyć swój arkusz stylów, musisz umieścić odpowiednią dyrektywę PHP. Ta metoda działa we wszystkich przeglądarkach:

Najprościej jest dodać poniższy kod do każdej strony PHP:

```
<?php if ($_SERVER['HTTP_HOST'] == "staging.example.com") { ?>
<link rel="stylesheet" href="/staging.css" type="text/css" />
<?php } ?>
```

Ten kod wpisuje do dokumentu łącze do arkusza stylów, a jeśli jakaś przeglądarka go nie obsługuje, to i tak nie pokaże żadnego stylu.

Ta metoda działa doskonale w przypadku serwowania plików z domeny *staging.example.com*. Lepszym rozwiązaniem, które będzie działać z każdego serwera z określonym łańcuchem w nazwie domeny, a nawet z serwera lokalnego, działającego na naszym komputerze osobistym, jest:

```
<?php
if(preg_match("/staging|test|dev|localhost|127\.0\.0\.1/", $_SERVER['HTTP_HOST'])){ ?>
<link rel="stylesheet" href="/staging.css" type="text/css" />
<?php } ?>
```

Można też zastosować warunkowe wysyłanie nagłówków HTTP przy użyciu PHP, ale jeśli na każdej stronie chcesz stosować wykrywanie serwera, to równie dobrze możesz zastosować wstawianie łącza do arkusza stylów.

Na pewno podobne rozwiązania można zaprogramować przy użyciu innych języków programowania używanych do tworzenia stron internetowych. Przedstawione urywki kodu pokazują, czego należy szukać.

Za powyższy kod PHP dziękuję dwóm uczonym dżentelmenom: Zachary'emu Johnsonowi (*http://www.zachstronaut.com/*) i Alanowi Hoganowi (*http://alanhogan.com/*), natomiast trzeciemu, Peterowi Wilsonowi (*http://peterwilson.cc/*) — za pokazanie mi strony ze wskazówkami dotyczącymi serwera IIS.

Skorowidz

!important, 52, 53
$, 69
*, 57, 67
:active, 48
:after, 48, 75, 76
:before, 48, 75, 76
:checked, 49
:disabled, 49
:empty, 40, 49
:enabled, 49
:first-child, 48
:first-letter, 48
:first-line, 48
:first-of-type, 49
:focus, 48
:hover, 48
:lang(), 48
:last-child, 49
:last-of-type, 49
:link, 48
:not(), 40, 49
:nth-child(), 49, 244, 246, 248
:nth-child(even), 245
:nth-child(odd), 245
:nth-last-of-type(), 49
:nth-of-type(), 49
:only-child, 49
:only-of-type, 49
:root, 49
:target, 49, 51
:visited, 48
@media, 241
^, 68
~, 64, 65

A

adres URL, 66, 67
aktywny element, 48
alt, 40
arkusze serwerowe, 111
arkusze stylów dla mediów, 237
  arkusze stylów dla druku, 94, 95
arkusze stylów zerujące domyślne ustawienia, 43
ASCII, 76
aspect-ratio, 242
atrybuty
  class, 58
  href, 63
  id, 58
  media, 95
  scope, 212
  selektory, 62
  wartości, 66
audio, 235
automatycznie adaptująca się siatka, 245

B

background, 93, 110, 249, 254, 255, 256
background-color, 250, 256, 257
background-image, 257
background-position, 257
background-repeat, 257, 258
biblioteki JavaScript, 234
body, 108, 110
  tło elementu, 110
border, 86
border-bottom-left-radius, 183
border-bottom-right-radius, 183
border-collapse, 214
border-radius, 181
border-style, 86
border-top-left-radius, 183
border-top-right-radius, 183
boxpunch, 174
box-shadow, 253

C

caption, 224
chmurki, 170
cienie, 252
class, 58
  selektory atrybutów, 64
clear, 125, 243
clearfix, 125
color, 242, 250
color-index, 242
Complexspiral, 166
CSS, 17
CSS 2.1, 48
CSS 3, 40, 49
  kolory RGBA, 249
  pseudoklasy, 49
  zaokrąglanie rogów, 181
CSS parallax, 191
czcionki, 83

D

dane tabelaryczne, 224
definiowanie
  punktory list, 100
  tło elementów body i html, 110
  wiele obrazów w tle elementów, 254
device-aspect-ratio, 242
device-height, 242
device-width, 242
diagnostyczny arkusz stylów, 40
display, 88, 89
  block, 97
  none, 88
div, 64, 107
document.getElementById(), 88
dodawanie obramowania do kontenera, 175
dokładność selektora, 51
dołączanie skryptu IE9.js do strony, 46
domyślne style w przeglądarkach, 42
dopełnienie, 97
Dragonfly, 33
  Styles, 33
  Układ, 33, 34
drukowanie, 94
dzieci, 70
  formatowanie wybranych elementów, 242

E

efekty, 165
  Complexspiral, 166
  menu CSS, 172
  menu podręczne, 170
  niedostępność obrazu, 93
  nieregularne kształty, 174
  nieregularne kształty pływające, 193
  paralaksa CSS, 191
  pola obrazów, 201
  przesuwane drzwi, 185
  przesuwane drzwi z przycinaniem, 189
  Sliding Doors, 185
  sprajty CSS, 183
  Super Ragged Floats, 197
  supernieregularne kształty pływające, 197
  zanikające tło, 51
  zaokrąglanie rogów, 177
elementy blokowe, 96
elementy HTML 5, 234
elementy pływające, 121
elementy siostrzane, 73
em, 84, 85, 150

F

fałszywe kolumny, 132
figcaption, 235
figure, 235
Firebug, 18
  badanie elementów, 20
  edycja kodu CSS, 23
  kod CSS, 19
  prezentacja stylów, 20
  pseudoelementy, 22
  pseudoklasy, 22
  struktura dokumentu, 18
  Styl, 19, 20
  Układ, 23
  układ elementów, 19
  wartości stylów, 22
  Zbadaj element, 21

Firefox, 18
fixed positioning, 161
float, 123
Fluid Grids, 146
fokus, 48
font, 83
font-size, 83, 85
fonty, 83
footer, 58
formatowanie
  elementy HTML 5, 234
  łącza, 70
  wybrane elementy-dzieci, 242
  wybrane kolumny, 246

G

generowanie
  markery, 105, 106
  treść, 75
GIF89a, 189
grid, 242

H

header, 58
height, 226, 242
Holy Grail, 142
href, 40, 63
HSL, 250
hsl(), 251
HSLA, 250
hsla(), 251
html, 108, 110
  tło elementu, 110
HTML 5, 234
  audio, 235
  imitacja elementów za pomocą nazw klas, 236
  obrazy, 235
  stare przeglądarki, 235
  video, 235

I

id, 58
identyfikator fragmentu, 50
identyfikatory, 58
  łączenie z klasami, 61
  selektory atrybutów, 65
  stosowanie, 60
IE7.js, 45
IE8.js, 45
IE9.js, 45, 46
  dołączanie skryptu do strony, 46
IEDT, 29
image replacement, 92
imitacja elementów HTML 5
  za pomocą nazw klas, 236
inset, 86, 88
Internet Explorer 6, 46, 62
  PNG, 181
  selektory dzieci, 72
Internet Explorer 7, 29
Internet Explorer 8, 32
Internet Explorer 9, 45, 46
Internet Explorer Developer Toolbar, 29
  CSS Selector Matches, 30, 31
  Show Default Styles, 31
  widoki, 30
IR, 92

J

JavaScript, 234
jednostki, 84
  em, 150
jedyny słuszny układ, 138
JQuery, 234

K

kanał alfa, 249, 250
klasy, 58
  łączenie z identyfikatorami, 61
  przypisywanie jednego elementu do wielu klas, 61
kleiste stopki, 161
kliring elementów przylegających, 126
kolejność fontów, 83
kolejność zapisu klas, 62
kolorowanie krawędzi, 55
kolory
  HSL, 250
  HSLA, 250
  obramowanie, 86, 87
  RGBA, 249
kolumny tekstu, 128
komentarze warunkowe, 46
kontekst pozycjonowania, 157
kontenery, 107
kontenery elementów pływających, 121

L

line-height, 83
  wartości bezjednostkowe, 84
link, 94
Liquid Bleach, 135
list-style, 100
list-style-image, 100
listy, 71
  generowanie markerów, 105
  markery w tle, 104
  punktory, 100
  punktory w tle, 102, 103
  wycinanie, 99

Ł

łącza, 63, 70
  łącza blokowe, 96
  łącza do adresów e-mail, 70
  łącza do usługi AOL Instant Messenger, 70
  łącza nawigacyjne, 59
  łącza szyfrowane, 70
  nieodwiedzone łącze, 48
  odwiedzone łącze, 48
  stany, 185
  strony przeznaczone do druku, 77
łączenie identyfikatorów z klasami, 61

M

macierz przekształcenia, 267
mapy, 217
margin, 98
marginesy, 97
  automatyczna szerokość, 120
  ujemne wartości, 154
  wysuwanie elementów poza kontenery, 155
margin-left, 98, 99
markery, 104
  generowanie, 105
matrix(), 267
max-device-height, 241
max-width, 202
media, 94, 95, 237
media="print", 95
media="screen", 95
menu CSS, 172
menu podręczne, 170
menu rozwijane, 173
min-device-height, 241
min-width, 238
model barw HSL, 251
model kolorów RGBA, 249
monochrome, 242
-moz-border-radius, 181
-moz-box-shadow, 253
-moz-transform, 258
-ms-transform, 258

N

nagłówek, 58, 161
  nagłówek z cieniem, 252
nagłówki HTTP, 113
narzędzia, 17
Narzędzia deweloperskie, 32
nazwy klas, 58
nieodwiedzone łącze, 48
nieregularne kształty, 174
nieregularne kształty pływające, 193
not, 241

O

obramowanie, 86, 116
  inset, 86
  kolor, 86, 87
  outset, 86
obrazy, 201, 235
  obrazy w tle, 108, 254
  obrazy zamiast tekstu, 92
  ograniczanie rozmiaru, 202
  pola obrazów, 201
obrót, 258
obrysy, 116
  grubość, 117

odnośniki śródliniowe, 97
odwiedzone łącze, 48
ograniczanie rozmiaru obrazów, 202
ol, 71
One True Layout, 138
only, 241
Opera, 33
orientation, 242
-o-transform, 258
outline, 116, 226
outset, 86, 88
overflow, 122
  auto, 128

P

padding, 98
padding-left, 98
padding-top, 162
paralaksa CSS, 191
paski narzędzi
  Internet Explorer Developer Toolbar, 29
  Web Developer, 24
Phark, 93
płynne siatki, 146
pływające elementy o różnej wysokości, 243
pływające kontenery elementów pływających, 123
PNG, 181, 189
podgląd stylu druku w przeglądarce, 96
pola obrazów, 201
poprawność kodu, 82
position, 90
  absolute, 90, 157
  fixed, 163
  relative, 92, 157
  static, 91
pozycjonowanie, 91
  obrazy w tle, 191
  pozycjonowanie bezwzględne, 91
  pozycjonowanie elementów w innym elemencie, 157
  pozycjonowanie na sztywno nagłówków i stopek, 161
  pozycjonowanie procentowe, 191
  pozycjonowanie stałe, 161
  pozycjonowanie w kontekście, 156
  wyjście poza kontener, 158
precyzja selektorów, 51, 57
  identyfikatory, 60
progressive enhancement, 77
prosty układ dwukolumnowy, 128
prosty układ trzykolumnowy, 129
przekształcenia dwuwymiarowe, 258
  kilka przekształceń naraz, 266
  kolejność przekształceń, 267
  macierz przekształcenia, 267
  obrót, 258
  przekształcenia macierzowe, 267
  skalowanie, 260
  skalowanie w poziomie, 261
  translacja, 262
  zniekształcenie, 264
przepełnienie, 121
przesuwane drzwi, 185
przesuwane drzwi z przycinaniem, 189
przezroczystość, 249
  obrazy PNG, 181
przypisywanie jednego elementu do wielu klas, 61
pseudoelementy, 48, 49
  :after, 48, 75, 76
  :before, 48, 75, 76
  :first-letter, 48
  :first-line, 48
  formatowanie, 76
pseudoklasy, 48, 49
  :active, 48
  :checked, 49
  :disabled, 49
  :empty, 49
  :enabled, 49
  :first-child, 48
  :first-of-type, 49
  :focus, 48
  :hover, 48
  :lang, 48
  :last-child, 49
  :last-of-type, 49
  :link, 48
  :not(), 49
  :nth-child(), 49
  :nth-last-of-type(), 49
  :nth-of-type(), 49
  :only-child, 49
  :only-of-type, 49
  :root, 49
  :target, 49
  :visited, 48
  CSS 3, 49
punktory, 100
  punktory w tle, 102, 103
pusta przestrzeń wokół treści strony, 98
px, 85

R

reguła precyzji, 51, 57
reguła ważności, 52
resolution, 242
rgb(), 249
RGBA, 249
rgba(), 249
rogi owalne, 182
rotate(), 258, 259
Rundle Method, 93

S

Safari, 35
scale(), 260
scaleX(), 261
scaleY(), 262
scan, 242
scope, 212
Selectoracle, 38
selektory, 47
  *, 57
  reguła precyzji, 51, 57
  reguła ważności, 52
  selektor uniwersalny, 51, 57
  selektory elementów siostrzanych, 73, 74
  selektory potomków, 70
selektory atrybutów, 62, 65
  $, 69
  ^, 68
  ~, 64, 65
  atrybut class, 64
  formatowanie łączy, 70
  identyfikatory, 65
selektory dzieci, 70
  IE 6, 72
selektory podłańcuchów, 66
  wartości atrybutów, 66, 68
separatory kolumn, 131
skalowanie, 260
  skalowanie w poziomie, 261
skew(), 264
skewX(), 264, 265
skewY(), 264, 265
skrócony zapis własności, 53
  wybiórcze przesłanianie własności, 55
skrypt IE9.js, 45
Sliding Doors, 135, 185
  wersja z przycinaniem, 189
sprajty CSS, 183
sprawdzanie poprawności kodu, 82
stany łączy, 185
stopka, 58, 161
stopniowe ulepszanie, 77
stosowanie obrysu, 116
strony przeznaczone do druku, 77
struktura tabeli, 208
styl obramowania, 86
style dla druku, 94
Super Ragged Floats, 197
supernieregularne kształty pływające, 197

Ś

Święty Graal, 142

T

tabele, 207
  formatowanie kolumn przy użyciu klas, 215
  formatowanie komórek nagłówka, 212
  formatowanie według kolumn, 213
  formatowanie wybranych kolumn, 246
  kolor tła kolumny, 215
  mapy, 217
  nagłówek, 208
  nagłówki wierszy, 211
  obramowanie kolumny, 215
  pozycjonowanie zawartości, 227
  stopka, 208
  struktura tabeli, 208
  treść główna, 208

tabele
  wizualne oddzielenie nagłówka i stopki od treści głównej, 210
  wykresy, 224
  wyrównanie zawartości komórek, 208
tabele kolorów HSL, 251
table, 208
tbody, 208, 209
techniki
  boxpunch, 174
  clearfix, 125
  fałszywe kolumny, 132
  Fluid Grids, 146
  IR, 92, 93
  Liquid Bleach, 135
  Sliding Doors, 185
  stopniowe ulepszanie, 77
text-align, 208
text-indent, 93, 99
tfoot, 209
th, 211
  scope, 212
thead, 208, 209
title, 40
tła dokumentu, 110
tr, 208
transform, 258, 265
  matrix(), 267
  rotate(), 258, 259
  scale(), 260
  scaleX(), 261
  scaleY(), 260, 262
  skew(), 264
  skewX(), 264, 265
  skewY(), 264, 265
  translate(), 262
transform-origin, 259
translacja, 262
translate(), 262, 264
treść generowana, 77
tworzenie
  arkusze stylów dla druku, 95
  mapy z danych tabelarycznych, 217
  wykresy, 224

U

ujemne marginesy, 176
  układ normalny, 154
układy, 115
  clearfix, 125
  fałszywe kolumny, 132
  Fluid Grids, 146
  Holy Grail, 142
  jedyny słuszny układ, 138
  kolumny, 128
  Liquid Bleach, 135
  obramowanie, 116
  obrysy, 116
  One True Layout, 138
  płynne siatki, 146
  pozycjonowanie na sztywno nagłówków i stopek, 161
  pozycjonowanie stałe, 161
  pozycjonowanie w kontekście, 156
  separatory kolumn, 131
  Święty Graal, 142
  ujemne marginesy w układzie normalnym, 154
  układ oparty na jednostkach em, 150
  układy dwukolumnowe, 128
  ustawianie bloków na środku, 118
  wielokolumnowe płynne układy, 135
  zmiana liczby kolumn, 241
układy trzykolumnowe, 129, 237, 238
  wąskie ekrany, 238
ukrywanie elementów, 91
umieszczanie
  elementy pływające w pływającym kontenerze, 123
  fragmenty treści „w jednej linii", 154
  łącza poza nagłówkiem, 159
Unicode, 76
URL, 66, 67
ustawianie bloków na środku, 118

V

video, 235
visibility, 89, 214

W

walidacja poprawności kodu, 82
walidator HTML W3C, 82
wartości atrybutów, 66, 67, 68
wcięcia list, 98
wcięty cytat blokowy, 176
Web Developer, 24
  identyfikatory, 27
  Informacje, 25, 27
  klasy, 27
  Narzędzia, 27, 28
  wyłączanie dołączonych arkuszy stylów, 26
  Zaznacz, 27
  zaznaczanie elementów, 27, 28
WebInspector, 35
  informacje o polu elementu, 36
  Programowanie, 35
  style nadane, 35, 36
-webkit-border-radius, 181
-webkit-box-shadow, 253
-webkit-transform, 258
widoczność elementu, 89
width, 118, 122, 226, 242
wiele obrazów w tle elementów, 254
wielokolumnowe płynne układy, 135
wielopoziomowe menu rozwijane, 172
własności pisma, 83
wskazywanie identyfikatora fragmentu, 50
wstawianie
  adresy URL do stylów dla druku, 77
  treść na początku elementów listy, 75
  znaki za pomocą specjalnych kodów, 76
wybiórcze przesłanianie własności w zapisie skróconym, 55
wybór elementów dzieci, 71
wybór podłańcucha na końcu wartości atrybutu, 69
wybór podłańcucha na początku wartości atrybutu, 68
wycinanie list, 99
wydzielanie elementu, 175
  elementy z obramowaniem, 176
wyjście poza kontener, 158
wykresy, 224
  formatowanie kontenerów słupków, 227
  legenda, 231
  słupki, 228
  wyrównanie szerokości słupków, 230
wyłączanie widoczności elementu, 89
wyłączanie wyświetlania elementów, 88
wyrównanie obrazu z pierwszym wierszem tekstu elementu, 102
wyróżnianie elementu docelowego, 50
wyrzucanie elementów poza ekran, 90
wyskakujące okienka, 171
wysokość linii, 83
wysuwanie elementów poza kontenery, 155
wysyłanie nagłówków HTTP, 113
wyśrodkowanie elementu w kontenerze, 118
wyświetlanie elementów, 88

X

XRAY, 37

Z

zakładki, 185, 186, 187
zanikające tło, 51
zaokrąglanie rogów, 177
  CSS 3, 181
zapis skrócony wartości, 53
zapytania o media, 237
  not, 241
  only, 241
  przecinki, 241
  składniki, 242
zastępowanie tekstu, 92
zerowanie stylów, 42
z-index, 161
zmiana koloru markerów, 101
zmiana liczby kolumn, 241
zniekształcenie, 264
zwielokrotnienie obrazów w tle, 108

OPANUJ NAJLEPSZE TECHNIKI TWORZENIA NOWOCZESNYCH UKŁADÓW STRON WWW!

Czym byłby dzisiaj internet, gdyby nie fantastyczne możliwości CSS? Przestrzenią pełną nudnych, podobnych do siebie i zapewne średnio atrakcyjnych dla współczesnych użytkowników stron WWW... Choć jeszcze dziesięć lat temu technologii tej wróżono odejście do lamusa, dziś swoją popularnością dorównała językowi HTML! Kaskadowe arkusze stylów spotkamy wszędzie: od przeglądarek internetowych, przez sklepy internetowe, po aplikacje do czatowania. CSS nadal rozkwita, a w branży twórców stron internetowych niewiele jest osób, które potrafią tak dobrze objaśnić wszystkie aspekty korzystania z języka CSS, jak słynny Eric Meyer — autor tej fantastycznej książki!

To właśnie z nim wyruszysz w podróż do najeżonego nowościami i zmianami świata najświeższych specyfikacji HTML 5 i CSS 3. W pierwszej części znajdziesz krótki przegląd przydatnych narzędzi i podstawowych technik, wliczając w to mało znane selektory CSS. Następnie poznasz ciekawe efekty oraz różne sposoby osiągania tego samego celu i tworzenia wydajnych układów przy użyciu CSS. Dowiesz się, jak CSS 3 współdziała z biblioteką JQuery. W ostatniej części znajdziesz opis technik zaawansowanych. Co istotne, każdy opis jest niezależny od pozostałych, możesz więc otworzyć książkę na dowolnej stronie i wykorzystać w swojej pracy to, co się na niej znajduje — bez obaw, że straciłeś coś ważnego.

- opis więcej niż piętnastu technik rozmieszczania elementów na stronie (clearfix, układy dwu- i trzykolumnowe, One True Layout, układy oparte na jednostce em, płynne siatki)
- sposoby ukrywania elementów i wyrzucania ich poza ekran
- metody definiowania tła elementów body i html w języku XHTML
- opis wielu efektów CSS (wyskakujące okienka CSS, tworzenie nieregularnych kształtów, sprajty CSS, Sliding Doors, Liquid Bleach)
- techniki formatowania tabel za pomocą CSS, w tym elementów thead, tbody, tfoot i nagłówków wiersy
- sposoby formatowania wybranych kolumn, stylizowania tabel przy użyciu JQuery, zmieniania tabel w wykresy i mapy
- przegląd niektórych nowości w języku CSS 3 (definiowanie wielu obrazów w tle elementów, model kolorów RGBA)

Eric A. Meyer jest znanym na całym świecie specjalistą od języków HTML i CSS oraz standardów sieciowych. Pracuje w tym zawodzie już od 1993 roku! Jest założycielem firmy Complex Spiral Consulting, której klientami są takie koncerny, jak Apple, Adobe czy Microsoft. Meyer jest również autorem pięciu bestsellerów poświęconych tematyce kaskadowych arkuszy stylów i projektowania stron, w tym takich jak „CSS według Erica Meyera. Sztuka projektowania stron WWW" (Helion 2006) i „CSS według Erica Meyera. Kolejna odsłona" (Helion 2008).

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Informatyka w najlepszym wydaniu

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach
http://helion.pl/nowosci

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

helion.pl
księgarnia internetowa

WILEY

ISBN 978-83-246-3242-8

Cena 49,00 zł