O'REILLY®

Responsywne strony WWW dla każdego

TWÓRZ STRONY WWW DLA WSZYSTKICH URZĄDZEŃ!

HELION

Clarissa Peterson

Tytuł oryginału: Learning Responsive Web Design

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-0035-4

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

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

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

Printed in Poland.

Spis treści

ROZDZIAŁ 4.

Style CSS w responsywnych stronach

Kod HTML nadaje strukturę całej treści Twojej strony WWW, natomiast kod CSS instruuje przeglądarkę, jak treść ma być przedstawiona.

Ten rozdział, podobnie jak poprzedni, będzie dla doświadczonych programistów stanowił przypomnienie wiadomości. Opisuje on jednak bardziej szczegółowo, w porównaniu z HTML, podstawową koncepcję języka CSS, z tego prostego powodu, że CSS jest kodem decydującym o responsywności strony. Dogłębne zrozumienie budowy responsywnej strony jest niemożliwe bez poznania takich pojęć jak „kaskada stylów" i „model pudełkowy".

Najpierw dowiesz się, na czym polega wersjonowanie języka CSS i w jaki sposób stosować prefiksy w celu prawidłowego zinterpretowania nowych właściwości stylów przez różne przeglądarki, nawet gdy właściwości te są jeszcze na etapie testów.

Następnie poznasz różne sposoby umieszczania stylów w swojej stronie WWW, czy to przez osadzanie ich za pomocą arkusza stylów obejmującego wiele stron lub całą witrynę, czy za pomocą wstawiania, obejmującego tylko poszczególne elementy strony.

W dalszej kolejności poznasz pojęcie **kaskady**, od którego wzięła się nazwa *Cascading Style Sheets* (CSS — kaskadowe arkusze stylów). Kaskada określa kolejność stosowania stylów i ich dobór przez przeglądarkę w przypadku wystąpienia konfliktu stylów. Poznasz również zalecany, wymagający minimalnego nakładu pracy sposób zastosowania kaskady przy implementacji stylów w swojej stronie.

W kolejnej części rozdziału zapoznasz się z pojęciem **modelu pudełkowego**, określającym sposób wyświetlania elementów na stronie. Każdy element jest traktowany jak pudełko z przypisanymi mu wartościami (niekiedy zerowymi) opisującymi szerokość, wysokość, marginesy, odstępy i ramki. Poznasz zasady wyświetlania i pozycjonowania elementów określające ich położenie na stronie.

Na zakończenie wrócimy do naszej przykładowej strony i zastosujemy w niej kilka stylów czcionki i położenia elementów, wzbogacając w ten sposób jej wygląd.

Jak działają style CSS

Poniżej, dla wszystkich czytelników, którzy nie znają języka CSS, zostało umieszczone krótkie przypomnienie, z czego składa się kod w arkuszu stylów. Opis nie obejmuje wszystkich możliwych sposobów wykorzystania stylów, ale jest wystarczający do prawidłowego odczytywania i zrozumienia zawartości plików CSS.

Zacznijmy od pojęcia **reguły**, oznaczającego każdorazowe zastosowanie stylu w danym elemencie:

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

Każda reguła składa się z dwóch osobnych sekcji. **Selektor** opisuje element języka HTML, bez nawiasów, w którym będzie zastosowany dany styl. W powyższym przypadku litera **p** odpowiada elementowi HTML `<p>`, a sama reguła określa sposób, w jaki przeglądarka wyświetli akapit na stronie.

Za selektorem znajduje się jedna lub kilka **deklaracji**. Wszystkie deklaracje są umieszczone w nawiasach i definiują styl, który będzie zastosowany w elemencie.

Deklaracja zawiera **właściwość** elementu, która ma być zmieniona, na przykład kolor lub szerokość. Jest jej przypisywana **wartość**, na przykład `orange` lub `50%`. Po każdej właściwości znajduje się dwukropek i wartość, na przykład `color: red`, jak w powyższym przykładzie, w którym określony jest czerwony kolor tekstu.

Jeżeli w regule umieszczonych jest kilka deklaracji, są one rozdzielone średnikami:

```
p { color: red; font-size: 1.5em; }
```

Umieszczenie średnika po ostatniej (lub jedynej) deklaracji nie jest obowiązkowe, ale wielu programistów go stosuje w celu ujednolicenia kodu. Wszystkie deklaracje zamknięte w nawiasach klamrowych są zwane **grupą deklaracji**.

Dzięki klasom i identyfikatorom style można stosować dla podzbioru elementów:

```
.nazwa_klasy { color: blue; }
#identyfikator { color: green; }
```

Powyższy kod dotyczy elementów należących do klasy `nazwa_klasy` (na przykład `<p class=` ↪`"nazwa_klasy">`) lub elementów oznaczonych identyfikatorem `identyfikator` (na przykład `<div id=" identyfikator">`).

Klasy i identyfikatory funkcjonują podobnie i mogą dotyczyć dowolnego elementu na stronie. Jednak identyfikator można przypisać tylko jednemu, natomiast klasę dowolnej liczbie elementów na stronie.

Powinieneś stosować opisowe nazwy klas i identyfikatorów. Ich treść, w odróżnieniu od języka HTML, nie ma dla przeglądarki żadnego znaczenia, za to programiście jest łatwiej kodować, gdy wie, do czego służy dana klasa lub identyfikator, na przykład:

```
<p class="wprowadzenie">...</p>
<nav id="podstawowy">...</nav>
```

Elementy możesz określać bardziej precyzyjnie za pomocą **selektorów pochodnych**, opisujących elementy znajdujące się wśród innych elementów, na przykład:

```
.nazwa_klasy p { color: purple; }
```

Powyższy przykład pokazuje, że spośród wszystkich elementów należących do klasy `nazwa_klasy`, elementy `<p>` będą miały kolor fioletowy (nie dotyczy to elementów `<p>` spoza klasy `nazwa_klasy`).

Jeżeli chcesz zastosować ten sam styl w kilku selektorach, to zamiast tworzyć kilka osobnych reguł, możesz je zebrać w grupę, oddzielając przecinkami:

```
h1, h2 { color: green; }
```

W tym przykładzie styl instruuje przeglądarkę, że elementy `<h1>` i `<h2>` muszą mieć kolor zielony.

Wpisywanie spacji i podziałów wierszy podczas tworzenia pliku CSS nie jest obowiązkowe, z wyjątkiem spacji w selektorach pochodnych, na przykład `.nazwa_klasy p`, jak w powyższym przykładzie. Arkusz stylów może więc wyglądać jak poniżej. Jest on w zupełności poprawny, aczkolwiek zupełnie nieczytelny:

```
p{color:green}div{width:50%;backgroundcolor:blue}.nazwa_klasy{color:yellow}
```

Usuwając spacje, czyli *optymalizując* arkusz stylów, zmniejszasz wielkość pliku CSS (liczy się każdy bajt), dzięki czemu Twoja strona będzie ładować się szybciej (w rozdziale 11. dowiesz się, jak korzystać z oprogramowania automatycznie usuwającego spacje).

UWAGA

Trudno czyta się zoptymalizowane arkusze stylów, ponieważ wszystkie opisy są w nich umieszczone jeden za drugim. W Internecie są dostępne narzędzia wstawiające z powrotem spacje i podziały wierszy w odpowiednich miejscach pliku, dzięki czemu jest on bardziej czytelny. Wypróbuj narzędzie Clean CSS (*http://www.cleancss.com*) lub inne optymalizatory on-line.

Wersje języka CSS

Podobnie jak w przypadku języka HTML, są różne wersje języka CSS. Pierwsza pojawiła się w roku 1996, zaledwie kilka lat po utworzeniu pierwszej strony w języku HTML. Najnowsza wersja ma oznaczenie CSS3.

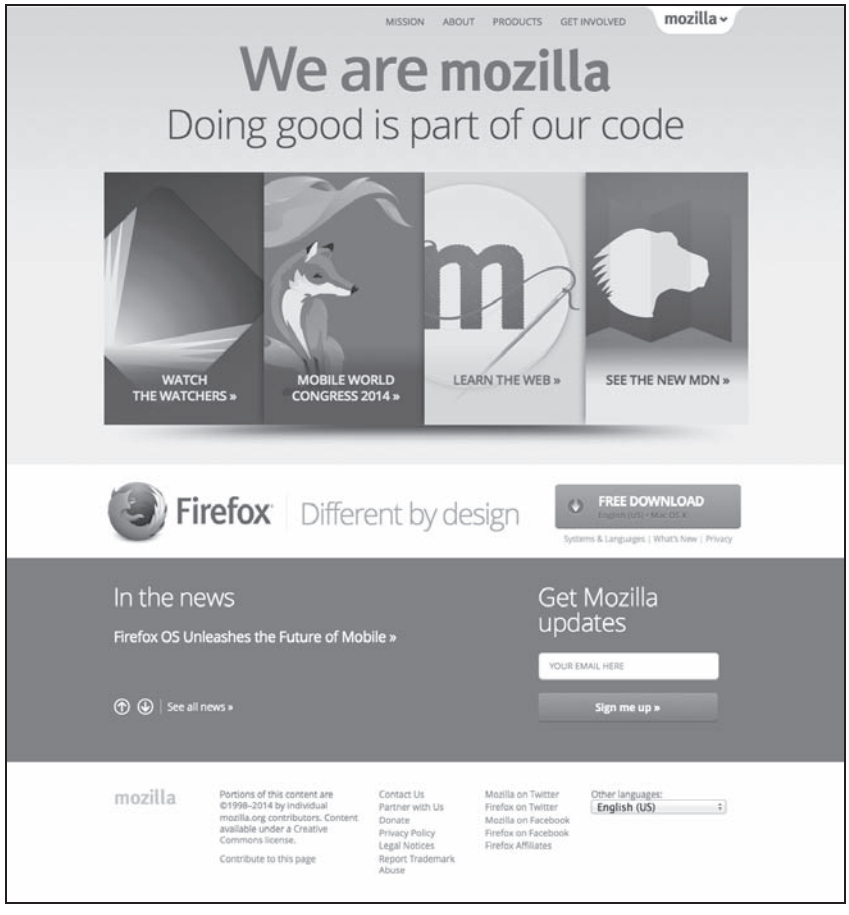
Każda wersja języka CSS obejmuje wszystkie właściwości elementów, w których można zastosować styl. Właściwości są to cechy elementów HTML zmieniane za pomocą stylów, na przykład kolor, czcionka, wielkość i położenie na stronie.

W wersji CSS3 zostało wprowadzonych wiele nowych właściwości elementów. Jedną z największych zmian są zapytania o medium, dzięki którym możliwe jest tworzenie stron responsywnych. W rozdziale 5. dowiesz się dokładniej, jak działają zapytania o media.

Pamiętaj, że podobnie jak w przypadku języka HTML, nie wszystkie przeglądarki potrafią interpretować i prawidłowo wyświetlać na stronie wszystkie właściwości zdefiniowane w wersji CSS3. W tej książce są wyróżnione właściwości, na które powinieneś zwrócić szczególną uwagę.

Również, podobnie jak w HTML, nie wszystkie przeglądarki interpretują wszystkie style CSS dokładnie tak samo. Powinieneś pamiętać o tych różnicach i testować strony za pomocą różnych przeglądarek i urządzeń, aby upewnić się, że wyglądają tak, jak tego oczekujesz.

Na rysunku 4.1 przedstawiona jest strona wyświetlona zgodnie z oczekiwaniami w przeglądarce Mozilla, natomiast na rysunku 4.2 jest pokazana ta sama strona z usuniętymi stylami CSS.



Rysunek 4.1. Strona wyświetlona zgodnie z oczekiwaniami w przeglądarce Mozilla

Prefiksy przeglądarek

Niektóre części języka CSS3 są wciąż w trakcie opracowywania przez grupę W3C i testowania przez twórców przeglądarek. Dlatego ostateczna specyfikacja każdego nowego stylu może się nieco różnić od sposobu, w jaki przeglądarki aktualnie go interpretują.

Zamiast pozostawiać programistów w niepewności, jak będzie wyglądał ich kod wyświetlony w danej przeglądarce, zostały wprowadzone **prefiksy przeglądarek** (zwane też **prefiksami producentów**), umożliwiające programistom utrzymanie lepszej kontroli nad nowymi stylami CSS.

W praktyce nie trzeba przejmować się *wszystkimi* dostępnymi przeglądarkami, ponieważ większość z nich opiera swoje działanie na jednym z czterech **silników wizualizacyjnych**, określających sposób wyświetlania strony. W celu wyświetlenia strony przeglądarka pobiera całą jej

Rysunek 4.2. Strona wyświetlona w przeglądarce Mozilla bez stylów CSS

treść (plik HTML i obrazy), style (plik CSS) i skrypty JavaScript, po czym podejmuje decyzję, jak strona będzie wyświetlona na ekranie. Wszystkie przeglądarki wykorzystujące ten sam silnik będą wyświetlały poszczególne fragmenty kodu w taki sam sposób.

Te cztery silniki wizualizacyjne, do których będzie się odwoływał fragment kodu umieszczony kilka akapitów niżej, to:

- **WebKit**, wykorzystywany przez przeglądarki Safari firmy Apple oraz Chrome firmy Google.
- **Gecko** firmy Mozilla, wykorzystywany w przeglądarce Firefox.
- **Trident** firmy Microsoft, stosowany w przeglądarce Internet Explorer.
- **Presto** firmy Opera, stosowany w przeglądarce Opera.

Ponieważ każdy z silników może w inny sposób wyświetlać style znajdujące się w fazie opracowywania, każdy z nich stosuje swoje właściwości z prefiksami, będące odmianami podstawowej właściwości.

Na przykład, nie zostały jeszcze ukończone prace nad właściwością `hyphens`, więc dostępne są jej odmiany dla czterech silników wizualizacyjnych, a każdą z nich możesz umieścić w swoim kodzie:

```
p {
  -webkit-hyphens: auto;
  -moz-hyphens: auto;
  -ms-hyphens: auto;
  -o-hyphens: auto;
  hyphens: auto;
}
```

Pięć powyższych deklaracji zawiera różne właściwości, przy czym ostatnia, która będzie ostatecznie zdefiniowana przez W3C, nie jest poprzedzona prefiksem.

Cztery pierwsze właściwości dotyczą poszczególnych silników wizualizacyjnych. Która właściwość dotyczy którego silnika, można poznać po pierwszej części nazwy — `webkit`, `moz`, `ms` lub `o`. Zwróć uwagę, że nazwy właściwości z prefiksami zaczynają się od myślnika.

W powyższym przykładzie, jeżeli właściwość `hyphens` jest wciąż testowana przez silnik WebKit, przeglądarka odczyta i zastosuje deklarację `-webkit-hyphens`. Przeglądarka nie rozpozna innych właściwości z prefiksami lub bez nich i je pominie.

Ale co się stanie, gdy specyfikacja zostanie zatwierdzona? Gdy w silniku WebKit zostanie zaimplementowana oficjalna właściwość `hyphens`, wtedy właściwość `-webkit-hyphens` będzie wciąż rozpoznawana, przynajmniej przez pewien czas. Ponieważ jednak właściwość `hyphens` bez prefiksu jest określona niżej w arkuszu stylów, również zostanie rozpoznana i zastąpi właściwość `-webkit-hyphens`.

W większości przypadków, gdy będziesz wykorzystywał właściwości, nad którymi prace jeszcze nie zostały zakończone, będziesz musiał stosować wszystkie cztery prefiksy, podobnie jak w pokazanym wyżej przykładzie.

Kolejność umieszczenia właściwości z prefiksami nie ma znaczenia. Większość programistów preferuje kolejność jak w powyższym przykładzie (tj. od najdłuższego do najkrótszego), ponieważ arkusz stylów jest wtedy bardziej czytelny. Jednak na końcu zawsze jest umieszczana właściwość bez prefiksu.

Jak zauważyłeś, w powyższym przykładzie każda deklaracja jest umieszczona w osobnym wierszu, czego zazwyczaj się nie robi w kodzie CSS. Ten przypadek jest wyjątkowy, ponieważ stosując podział na osobne wiersze, o wiele łatwiej jest sprawdzić, czy zostały uwzględnione wszystkie prefiksy i czy każdy z nich ma tę samą wartość (aczkolwiek w niektórych sytuacjach możesz stosować inne wartości niektórych właściwości).

W ostatecznej wersji projektu będziesz mógł zrezygnować z prefiksów i stosować tylko właściwości bez nich (jednak wciąż będą się pojawiały nowe właściwości, które będą wymagały użycia prefiksów).

Kilka nowych właściwości wprowadzonych w wersji CSS3 jest już obsługiwanych przez wszystkie najważniejsze przeglądarki. W prawidłowym wyświetlaniu strony nie przeszkadzają właściwości z prefiksami, które już nie są potrzebne. Z tego powodu na końcu listy umieszcza się właściwość bez prefiksu, aby zastąpiła wszystkie wcześniejsze. Powiększa to jednak niepotrzebnie plik CSS.

Jeżeli jakaś właściwość wprowadzona w wersji CSS3 wymagała w chwili pisania tej książki zastosowania prefiksów, zostało to wyraźnie zaznaczone. Jednak gdy czytasz tę książkę, status tych właściwości może się już zmienić.

Odwiedź stronę „Can I Use…” (*http://caniuse.com*) i sprawdź, jakie właściwości CSS są obsługiwane przez różne wersje przeglądarek i czy muszą być stosowane prefiksy.

Gdzie są zdefiniowane style CSS

Style możesz stosować na swojej stronie na kilka sposobów, czy to wykorzystując osobny plik zwany arkuszem stylów, czy wbudowując je w kod strony. Sposób zdefiniowania stylów CSS wpływa na ich zastosowanie w stronie.

Style wbudowane

Jeżeli chcesz, aby style były stosowane tylko w jednej stronie witryny, możesz umieścić je bezpośrednio w sekcji `<head>` w pliku HTML, używając elementu `<style>`. Wewnątrz tego elementu umieść po prostu jedną lub kilka reguł CSS w pokazany niżej sposób:

```
<head>
<style>
p { color: blue; }
</style>
</head>
```

Zazwyczaj stylów nie stosuje się tylko w jednej stronie, ale w całej witrynie. Nawet jeżeli określony styl będzie zastosowany tylko na jednej stronie, i tak umieszcza się go w osobnym arkuszu stylów. Dzięki umieszczeniu wszystkich stylów w jednym miejscu można je później łatwo znaleźć, szczególnie gdy nad kodem strony pracuje kilka osób.

Natomiast style wbudowane stosuje się podczas tworzenia kodu HTML dla wiadomości e-mail, ponieważ użytkownicy poczty zazwyczaj nie mają możliwości zaimportowania zewnętrznego arkusza stylów.

Arkusze stylów

Zazwyczaj deklaracje CSS są umieszczane w jednym lub kilku oddzielnych plikach, zwanych arkuszami stylów (ang. *stylesheets*), połączonych z plikami HTML. Style opisujące czcionkę, kolory i układ strony powinny być jednolite dla całej witryny. Tworząc arkusze, możesz zadeklarować style tylko raz i stosować je w każdej stronie swojej witryny.

Element `<link>` umożliwia połączenie dokumentów (plików) CSS i JavaScript z kodem HTML. Element ten umieszcza się w sekcji `<head>` pliku HTML:

```
<head>
<link rel="stylesheet" href="style/moje_style.css">
</head>
```

Z każdą stroną WWW może być połączonych kilka arkuszy stylów. Możesz w tym celu również użyć reguły `@import`, umieszczając ją wewnątrz sekcji `<head>` dokumentu i elementu `<style>`:

```
<head>
<style>
@import url(style/moje_style.css)
</style>
</head>
```

Łączenie i importowanie arkuszy stylów oznacza w rzeczywistości tę samą operację pobierania stylów wykonywaną przez przeglądarkę. Jednak wybór konkretnej metody może mieć wpływ na szybkość otwierania strony, o czym się dowiesz w rozdziale 11. Zazwyczaj powinieneś stosować element `<link>` i minimalizować liczbę stosowanych arkuszy stylów.

Style wstawiane

Niekiedy będziesz chciał kodować bardziej precyzyjnie i stosować style tylko w jednym elemencie bezpośrednio w kodzie HTML. W tym celu umieść w elemencie atrybut stylu (`style="..."`), podając w cudzysłowie dowolne style, które chcesz zastosować:

```
<p style="color: green; font-size: 2em;">To jest akapit.</p>
```

Powinieneś jednak unikać stosowania na stronie stylów wstawianych, ponieważ łatwo zapomnieć, gdzie są zdefiniowane poszczególne style i co definiują, przez co stronę będzie potem trudniej utrzymywać. Jednak style te mogą być przydatne podczas testowania zmian wprowadzanych na stronie.

Kaskada stylów

Jeżeli potrafisz stosować style CSS, to w tej części rozdziału znajdziesz znane Ci informacje, które jednak warto przypomnieć.

Jak już wiesz, style mogą być zdefiniowane w zewnętrznych arkuszach stylów bądź w sekcji `<head>`. Mogą to także być style wstawiane. Dowiedziałeś się też, że style mogą być przypisywane elementom, klasom i identyfikatorom. Jeżeli do elementu zostanie przypisany więcej niż jeden styl, skąd przeglądarka będzie wiedziała, który z nich zastosować?

W języku CSS obowiązują bardzo ścisłe reguły stosowania stylów, zwane **kaskadami**. Kaskada określa, które reguły mają priorytet względem innych reguł.

Pojęcie kaskady jest dość skomplikowane, ale w większości przypadków jej zastosowanie w kodzie CSS będzie całkiem proste, a działanie zrozumiałe. Opisane zostanie działanie nieco uproszczonej wersji kaskady, ale jeżeli będziesz bardzo dużo pracował nad kodem, to musisz się o kaskadzie dowiedzieć więcej.

Jak to działa

Każdy element HTML ma kilka właściwości. Na przykład właściwości elementu `<p>` to kolor tekstu, kolor tła, wielkość i krój czcionki, pogrubienie, pochylenie, obecność ramki i jej wygląd, pozycja akapitu na stronie itp. (jest jeszcze kilkanaście innych właściwości).

Przeglądarka podczas wyświetlania strony analizuje każdy element po kolei i na podstawie wartości jego właściwości decyduje, w jaki sposób go przedstawi.

Przeglądarka podczas rozstrzygania o stylach zastosowanych w poszczególnych elementach stosuje ściśle określoną procedurę — hierarchię reguł. Poniżej wymieniona jest kolejność reguł według priorytetów, począwszy od najwyższej:

1. Reguły oznaczone jako ważne.
2. Reguły stylów wstawianych.
3. Reguły zawierające identyfikatory.
4. Reguły zawierające klasy, atrybuty i pseudoklasy.
5. Reguły zawierające elementy i pseudoelementy.
6. Reguły odziedziczone.
7. Wartości domyślne.

Kolejność stosowania powyższych reguł jest często zwana **precyzją CSS**, ponieważ zaczyna się od najbardziej szczegółowej reguły (stosowanej tylko w jednej lub kilku instancjach danego elementu), a kończy na najbardziej ogólnej (stosowanej we wszystkich instancjach elementu).

Przeglądarka, analizując każdy element strony, sprawdza reguły zgodnie z hierarchią, aż znajdzie styl dotyczący konkretnej właściwości. Załóżmy na przykład, że potrzebne jest określenie koloru tekstu akapitu (właściwość `color`). Przeglądarka zacznie od sprawdzania reguł oznaczonych jako ważne, potem przejdzie do stylów wstawianych, stylów z identyfikatorami itd. Jeżeli znajdzie kilka kolorów tekstu akapitu, wybierze ten, który ma najwyższy priorytet w kaskadzie.

Oznaczenie ważności

Jeżeli chcesz, aby określona reguła była stosowana bez względu na okoliczności, możesz oznaczyć ją jako ważną — `!important`. Oznaczenie może być użyte w stylach zwykłych lub wstawionych. Przeglądarka zawsze sprawdza oznaczenie `!important` przed sprawdzeniem następnej reguły.

W poniższym przykładzie właściwość `color` została oznaczona jako `!important`, ale `font-weight` już nie. Oznaczenie `!important` umieszczone jest bezpośrednio przed średnikiem i dotyczy tylko jednej deklaracji, a nie całej grupy:

```
p { font-weight: bold; color: blue !important; }
```

Ten akapit będzie zawierał niebieski tekst, niezależnie od innych kolorów zdefiniowanych w innych deklaracjach dotyczących tego samego elementu.

Chociaż oznaczenie `!important` wydaje się użytecznym narzędziem, należy je stosować bardzo, bardzo rzadko, tylko wtedy, gdy naprawdę nie da się osiągnąć zamierzonego efektu za pomocą klas, identyfikatorów i innych selektorów. Gdy użyjesz tego oznaczenia, będzie miało ono bezwzględny priorytet i nie da się go — potocznie mówiąc — przeskoczyć.

Style wstawiane

Jeżeli przeglądarka nie znajdzie reguł z oznaczeniem `!important` opisujących kolor, wtedy będzie szukać stylów wstawianych. Są to style przypisane bezpośrednio do elementów w dokumencie HTML. Wszystkie style wstawiane mają wyższy priorytet od innych stylów zdefiniowanych w zewnętrznym arkuszu lub wbudowanych w sekcji `<head>`:

```
<p style="color: blue;">Tekst akapitu.</p>
```

Identyfikatory

Jeżeli nie są zdefiniowane style wstawiane opisujące kolor, wtedy przeglądarka będzie szukać identyfikatorów zastosowanych w danym elemencie, czy to bezpośrednio, czy za pomocą selektora pochodnego:

```
<p id="przykład">Tekst akapitu.</p>
#przykład { color: blue; }
```

Klasy, atrybuty i pseudoklasy

Jeżeli nie ma w kodzie identyfikatorów określających kolor, przeglądarka będzie szukać jakiejkolwiek klasy, atrybutu lub pseudoklasy, odnoszących się do danego elementu. Podobnie jak poprzednio, mogą one być umieszczone w selektorze pochodnym:

```
.przyklad { color: blue; }
.przyklad p { color: green; }
```

Elementy i pseudoelementy

Jeżeli nie ma żadnych stylów zawierających klasy, atrybuty lub pseudoklasy, przeglądarka będzie szukać stylów zawierających tylko elementy lub pseudoelementy w selektorze:

```
<p>Tekst akapitu.</p>
p { color: blue; }
```

Reguły odziedziczone

Jeżeli do tego miejsca kaskady przeglądarka nie znalazła stylów bezpośrednio określających kolor naszego elementu `<p>`, to będzie szukać stylów odziedziczonych, które będzie mogła zastosować.

Pojęcie **dziedziczenia** jest dość proste. Jeżeli w jakimś elemencie nie jest zastosowany styl, wtedy zostanie użyty styl odziedziczony po elemencie nadrzędnym:

```
<p>Tekst w tym akapicie jest <strong>pogrubiony</strong>.</p>
p { color: blue; }
```

Ponieważ do elementu `<strong>` nie jest przypisana wartość określająca jego kolor, zostanie odziedziczona wartość z elementu nadrzędnego, w tym przypadku `<p>`. Kolor tego elementu jest niebieski (`blue`) i taki zostanie nadany elementowi `<strong>`.

Nie wszystkie właściwości mogą być dziedziczone, jednak bardzo łatwo można rozpoznać, które z nich mogą. Na przykład wielkość czcionki jest dziedziczona, ponieważ po ustawieniu jej w elemencie `<p>` cały tekst w tym elemencie powinien mieć tę samą wielkość, nawet wewnątrz elementów `<strong>` lub `<em>`.

Natomiast właściwości ramek nie są dziedziczone. Jeżeli określisz ramkę elementu `<div>`, nie spodziewaj się, że każdy akapit wewnątrz tego elementu będzie miał własną ramkę.

Wartości domyślne

Jeżeli przeglądarka dotarła do tego miejsca kaskady i nie znalazła koloru naszego elementu `<p>`, to jeszcze nie wszystko stracone, ponieważ każdy element ma swoje wartości domyślne.

Wartości domyślne większości właściwości są określone w specyfikacji języka CSS. Na przykład każdy element tekstowy ma domyślny kolor czarny. Jeżeli nie określisz koloru, przeglądarka użyje domyślnego.

Jeżeli więc nie będzie zdefiniowany styl koloru, który można by zastosować w naszym elemencie `<p>`, będzie on zawierał czarny tekst.

Jak rozstrzygane są konflikty?

Oprócz przedstawionych informacji jest jeszcze pewien szczegół, o którym należy pamiętać.

Oprócz stosowania stylów zgodnie z hierarchią, sprawdzana jest liczba atrybutów w selektorze. Większa liczba atrybutów oznacza wyższy priorytet (tj. styl zawierający dwa identyfikatory ma wyższy priorytet niż styl z tylko jednym identyfikatorem).

Jeżeli mimo tego dwa style mają ten sam priorytet i wciąż nie wiadomo, który z nich powinien być zastosowany, wybrany będzie ten drugi w kolejności, w jakiej są zdefiniowane (w kierunku od początku arkusza stylów do jego końca).

To, co przeczytałeś w tej części rozdziału, stanowi w zasadzie całą wiedzę, jaką powinieneś posiadać na temat kaskady stylów. Jeżeli jednak chciałbyś dowiedzieć się więcej na temat reguł stosowania stylów lub użycia bardziej zaawansowanych funkcjonalności języka CSS, zapoznaj się z artykułem Vitaly'ego Friedmana pod tytułem *CSS Specificity: Things You Should Know* („Precyzja stylów CSS: rzeczy, o których powinieneś pamiętać", *http://www.smashingmagazine.com/2007/07/27/css-specificity-things-you-should-know/*).

Stosowanie kaskady stylów

Cały powyższy proces prawdopodobnie wydaje się dość skomplikowany, ale w rzeczywistości jest całkiem prosty, ponieważ to *Ty* decydujesz o tym, gdzie są stosowane style.

Podczas określania stylów na stronie w rzeczywistości stosujesz odwrotną kolejność niż opisana wyżej. Style powinieneś stosować w najszerszym możliwym zakresie i rzadko zagłębiać się w szczegóły.

Na przykład, nie będziesz raczej umieszczał na stronie tekstu w kilkunastu kolorach za pomocą wielu różnych stylów, które będą ze sobą konkurować. Określisz jeden lub dwa style i będziesz dokładnie wiedział, gdzie je zastosować. Będziesz stosował reguły w następującej kolejności:

1. Wartości domyślne.
2. Reguły odziedziczone.
3. Reguły zawierające elementy i pseudoelementy.
4. Reguły zawierające klasy, atrybuty i pseudoklasy.
5. Reguły zawierające identyfikatory.
6. Reguły stylów wstawianych.
7. Reguły oznaczone jako ważne.

Wartości domyślne i reset stylu

Przede wszystkim zacznij od stosowania wartości domyślnych. W większości przypadków będzie to oczywiste. Gdy nie będą użyte żadne style, jak na stronie utworzonej w rozdziale 3., tekst będzie czarny, czcionka nagłówka większa niż czcionka akapitu i nie będzie ramek. To są domyślne wartości.

Jednak przeglądarki nie zawsze stosują domyślne wartości, których oczekujemy. Dlatego aby mieć pewność, że startujemy z tego samego miejsca niezależnie od użytej przeglądarki, należy zresetować niektóre wartości do zera. W przeciwnym wypadku niewielkie pozostałości stylów będą miały niewłaściwe domyślne wartości, które będą wpływały na wygląd strony i powodowały, że będzie ona różnie wyświetlana przez różne przeglądarki.

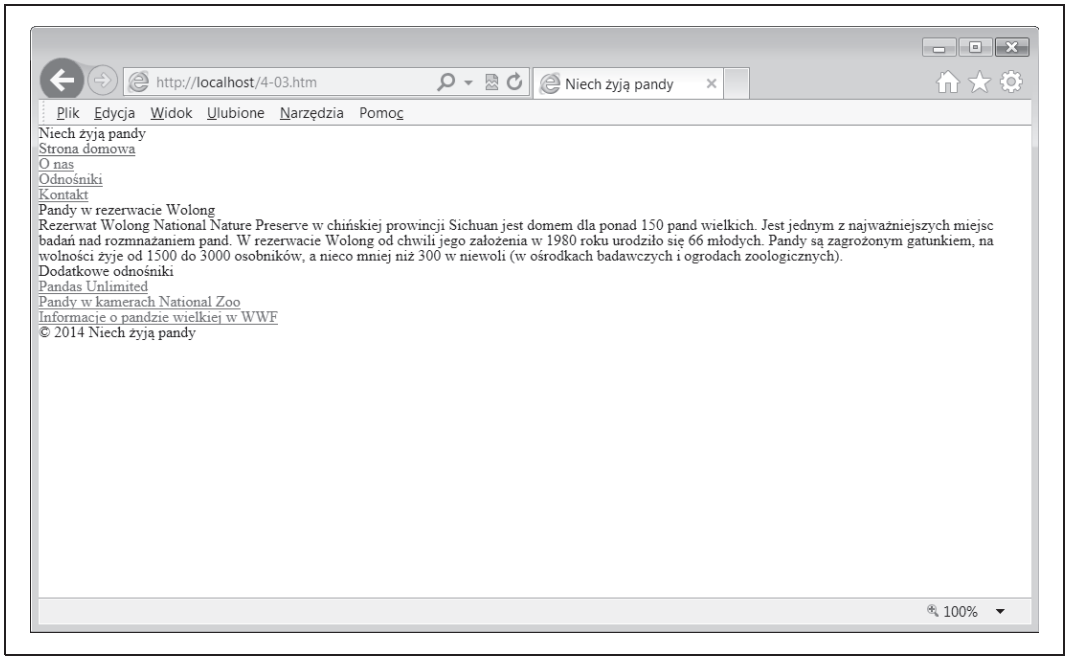
Reset wartości

W celu zresetowania stylu należy ustawić wszystkie marginesy i odstępy na zero, a niektóre właściwości czcionki na standardowe wartości bazowe. Po zresetowaniu wszystkich wartości można ustawić wszystkie właściwości stylów na żądane wartości. Sekcja pliku CSS, w której resetowane są wszystkie wartości, jest często nazywana **stylem resetującym** lub **arkuszem resetującym**.

Jeżeli na przykład zastosujemy reset CSS w naszej stronie o pandzie, otrzymamy efekt przedstawiony na rysunku 4.3.

Poprzednia strona wyglądała lepiej, prawda? Nie przejmuj się, za chwilę to naprawimy.

Wyszukanie wszystkich elementów, które mogą się różnić, i napisanie deklaracji CSS resetujących ich wartości wymagałoby włożenia mnóstwa pracy. Na szczęście, inni zrobili to już za nas i utworzyli bloki kodu CSS zawierające wszystkie zresetowane style, które są potrzebne. Bloki te można po prostu skopiować i wkleić do własnego arkusza stylów.



Rysunek 4.3. Nasza przykładowa strona ze zresetowanymi stylami

Jednym z najpopularniejszych kodów jest *Reset CSS* autorstwa Erica Meyera (*http://meyerweb.com/eric/tools/css/reset*). Jest on powszechnie dostępny, więc używaj go śmiało i modyfikuj według potrzeb.

Innym arkuszem jest plik *Normalize.css* (*http://necolas.github.com/normalize.css*) utworzony przez Nicolasa Gallaghera.

Możesz również utworzyć własny arkusz resetujący. Szczegółowe informacje zawarte są w serwisie Nettuts+ w artykule Jeffreya Waya *Quick Tip: Create Your Own Simple Reset.css File* („Szybka porada: tworzenie własnego prostego pliku *reset.css*", *http://net.tutsplus.com/tutorials/html-css-techniques/weekend-quick-tip-create-your-own-resetcss-file*).

Arkusz resetujący powinien być przetwarzany przez przeglądarkę na samym początku, aby nie zastąpił żadnych Twoich stylów. Dlatego powinien być podłączony w pierwszej kolejności w sekcji `<head>` lub umieszczony w pierwszych wierszach kodu CSS, jeżeli nie jest zawarty w osobnym pliku.

Reguły odziedziczone

W następnym kroku przyjrzymy się regułom odziedziczonym. Większość stylów w Twojej stronie będzie odziedziczonych.

Jeżeli ustawisz kolor różowy tekstu w sekcji `<body>` (chociaż nie sądzę, że tak zrobisz), wówczas cały tekst na stronie będzie różowy. Dzięki temu nie musisz zaprzątać sobie głowy ustawianiem kolorów akapitów, list, elementów `<div>` itp., chyba że chcesz, aby ich kolor był inny:

```
body { color: pink; }
```

Od tej zasady jest jeden wyjątek: odnośniki (<a>) nie dziedziczą koloru, są one domyślnie niebieskie (lub fioletowe, jeżeli zostaną odwiedzone), dzięki czemu są wizualnie wyróżnione na stronie. Możesz pozostawić kolory niebieski i fioletowy lub wybrać inne poprzez przypisanie ich do elementu <a>.

Upewnij się, że nie umieściłeś powtarzających się deklaracji stylów. Jeżeli element <body> ma kolor różowy, nie umieszczaj stylu, który ustawia ten kolor dla wszystkich elementów <p>. Został im już nadany kolor różowy, więc byłoby to wpisanie niepotrzebnego kodu.

Elementy

W pierwszej kolejności stosuj style o największym zasięgu. Jeżeli na przykład w całym tekście ma być użyta czcionka Helvetica, zastosuj odpowiednią regułę w elemencie <body>. W ten sposób wszystkie inne elementy odziedziczą tę czcionkę.

Jeżeli chcesz, aby we wszystkich najważniejszych nagłówkach była dla odróżnienia użyta czcionka Georgia, zastosuj ją w stylu dla elementu <h1>, dzięki czemu wszystkie nagłówki będą zapisane czcionką Georgia, a cały pozostały tekst czcionką Helvetica.

Gdy będziesz definiował jakikolwiek styl, spróbuj najpierw odszukać element na najwyższym poziomie hierarchii i w nim zastosuj dany styl.

W naszym przypadku chcemy zmienić czcionkę nagłówków tak, aby była ona większa niż pozostały tekst i pogrubiona:

```
h1 { font-size: 2em; }
h2 { font-size: 1.3em; }
h1, h2 { font-weight: bold; }
p, ul { font-size: 1em; }
```

Jednostka em jest jednostką względną. Wartość 2em oznacza, że tekst w elementach <h1> będzie dwa razy większy niż wartość bazowa (wielkość pozostałego tekstu na stronie). Więcej informacji na temat jednostki em znajduje się w części „Wymiary" niniejszego rozdziału).

W akapitach i listach zostanie zastosowana domyślna wielkość czcionki, jednak aby nie było niejasności, zdefiniujemy ją w naszym pliku CSS, nadając jej wartość 1em.

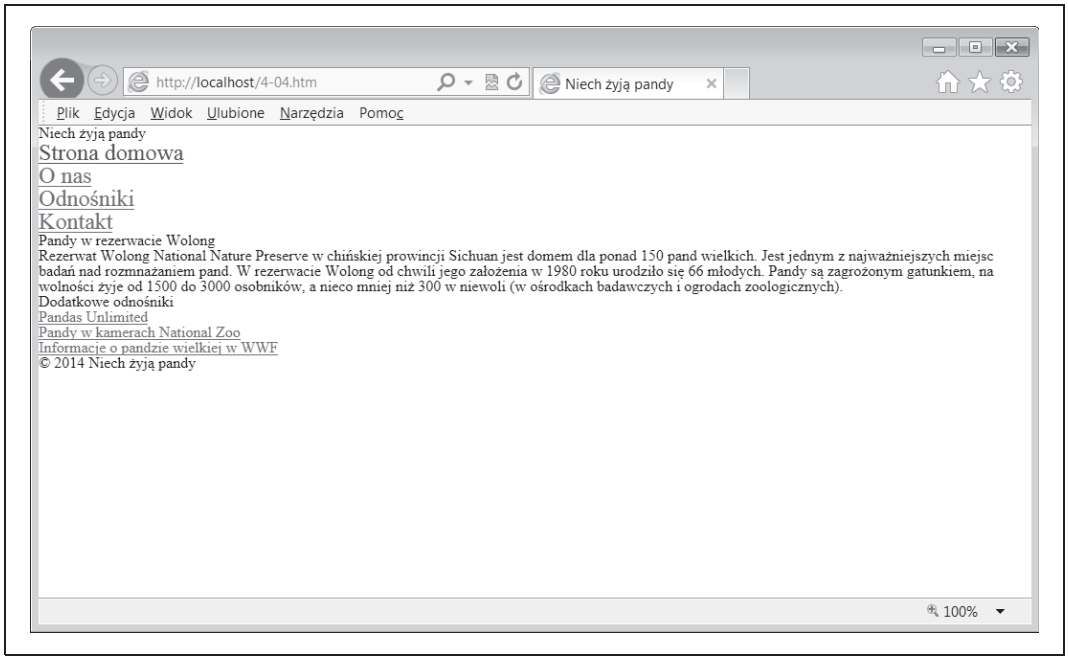
Cała reszta

Jeżeli styl użyty w jednym elemencie musi różnić się od stylu użytego we wszystkich pozostałych instancjach tego elementu, możesz to osiągnąć na kilka sposobów. Na przykład możesz określić, aby tekst odnośników w elemencie <nav> był większy:

```
nav li { font-size: 1.5em; }
```

Wszystkie pozostałe elementy <li> będą dziedziczyły zdefiniowaną dla nich wielkość tekstu, więc jak pokazuje rysunek 4.4, czcionka odnośników w sekcji *Dodatkowe odnośniki* nie zmieni wielkości.

Zacznij od zastosowania selektorów pochodnych, jak w poprzednim przykładzie, a następnie zdefiniuj klasy, za pomocą których wyróżnisz wybrane elementy spośród pozostałych.



Rysunek 4.4. Wielkość czcionki w liście w sekcji nawigacji będzie większa, ale w pozostałych listach będzie miała początkową wielkość

Nie używaj jednak selektorów pochodnych ani klas, jeżeli nie jest to konieczne. W poprzednim przykładzie został zastosowany styl bezpośrednio w elemencie `<nav>`, a nie selektor pochodny, ponieważ odnośniki stanowią jedyny tekst wewnątrz tego elementu. Ten sam efekt dałby następujący kod:

```
nav { font-size: 1.5em; }
```

Najbardziej precyzyjnymi selektorami są identyfikatory, style wstawiane oraz te z oznaczeniem `!important`, jednak powinieneś unikać ich stosowania, chyba że będziesz to robił z pełną świadomością.

Prostota

Możesz się dziwić, dlaczego trzeba maksymalnie upraszczać arkusz CSS. Czy nie można po prostu zdefiniować klas i stylów z oznaczeniami `!important`, skoro pozwalają osiągnąć zamierzony efekt?

Jest tak dlatego, że niemal w każdej stronie będą kiedyś wprowadzane zmiany w stylach CSS — niewielkie korekty kosmetyczne lub generalna przebudowa strony. Im prostszy będzie arkusz CSS na samym początku, tym wprowadzanie później zmian będzie łatwiejsze, szczególnie dla kogoś innego niż autor strony.

Jak wspomniano wcześniej, niniejsza książka zawiera tylko bardzo podstawowe informacje o funkcjonowaniu stylów CSS.

Jeżeli jakiś element jest wyświetlany na stronie niezgodnie z oczekiwaniami, powodem może być użycie w kaskadzie innego stylu niż zamierzony. Wykonuj różne próby i wprowadzaj zmiany w kodzie, aż dowiesz się, co jest tego przyczyną. Najprostszym sposobem weryfikacji, jak działają wszystkie style, jest wypróbowanie różnych metod i sprawdzenie efektu.

Zawsze testuj swoją stronę za pomocą różnych przeglądarek i urządzeń, aby mieć pewność, że same style CSS będą stosowane w ten sam sposób. Więcej informacji na temat testów jest zawartych w rozdziale 8.

Komentarze

W kodzie CSS, podobnie jak w HTML, możesz umieszczać komentarze, aczkolwiek w tym celu stosowane są inne symbole.

Cały tekst zawarty w komentarzach jest po prostu pomijany przez przeglądarkę.

Komentarze stosuje się w arkuszach stylów najczęściej w celu umieszczenia opisu lub objaśnienia, do czego używany jest jakiś styl, albo w celu dodania notatki z informacją, kiedy została wprowadzona zmiana:

```
/* To jest komentarz w pliku CSS */
/* Komentarz może składać się z jednego lub kilku wierszy,
a nawet może być w tym samym wierszu, co kod stylu */ h1 { color: #7b0000; }
```

Porządek w arkuszu stylów

Poza tym, że style muszą być zdefiniowane w określonej kolejności, aby były poprawnie zastosowane, sam arkusz może być dowolnie zorganizowany. Jednak najlepiej jest tworzyć go w usystematyzowany sposób, aby później można było w nim wszystko łatwiej znaleźć. Gdy umieścisz w nim jeszcze zapytania o media, musisz wiedzieć, gdzie co jest.

Zawsze musisz zaczynać od stylów resetujących. Może to być oddzielny plik albo pierwsze wiersze kodu w podstawowym arkuszu.

Niektórzy programiści porządkują arkusz stylów według sekcji strony, umieszczając komentarze opisujące każdą sekcję arkusza. Możesz w komentarzach umieszczać myślniki lub inne symbole, aby rysować „linie" pomiędzy sekcjami, dzięki czemu łatwiej będzie odnaleźć określony kod:

```
/* Podstawowa treść ---------------------------- */
Style kroju czcionki, układu strony itp. dla podstawowej treści
/* Nagłówek ------------------------------ */
Style kroju czcionki, układu strony itp. dla nagłówka
/* Stopka ------------------------------ */
Style kroju czcionki, układu strony itp. dla stopki
```

Inni programiści preferują umieszczanie podobnych stylów razem, niezależnie od sekcji strony, w których są stosowane, aby mieć lepsze rozeznanie, co się dzieje na stronie:

```
/* Czcionki ------------------------------ */

p { ... }
nagłówek p { ... }
stopka p { ... }
.nazwa_klasy p { ... }
```

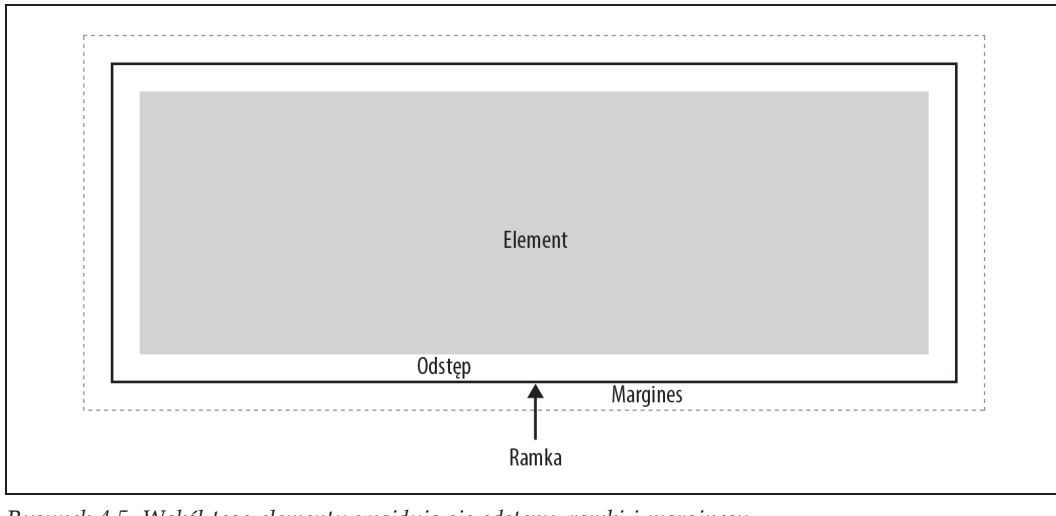
```
li { ... }
stopka li { ... }
.nazwa_klasy li { ... }

/* Układ ---------------------------- */

nagłówek { ... }
stopka { .... }
```

W swojej przeglądarce możesz przejrzeć kod HTML i CSS dowolnej strony. Przyjrzyj się arkuszom CSS różnych stron i sprawdź, jak zorganizowane są w nich style. Nie obowiązuje tu określona zasada, trzeba jednak być konsekwentnym.

Model pudełkowy

Najważniejszym pojęciem w języku CSS jest „model pudełkowy" strony. Oznacza ono, że każdy element HTML na stronie jest prostokątnym pudełkiem. Każde pudełko może mieć ramki, odstępy (puste miejsca wewnątrz ramki) i marginesy (puste miejsce na zewnątrz ramki), ale nie musi.

Jak pokazuje rysunek 4.5, zawartość pudełka znajduje się w jego środku. Zawartość może stanowić tekst, obrazy lub elementy podrzędne (na przykład akapit wewnątrz elementu `<div>`). Wokół elementu są odstępy, następnie ramki, potem marginesy, z których każdy może mieć szerokość lub wysokość równą zero (ramka o szerokości i wysokości równej zero jest niewidoczna).



Rysunek 4.5. Wokół tego elementu znajdują się odstępy, ramki i marginesy

Wymiary

Wysokość, szerokość, marginesy, odstępy i ramki elementów mogą być wyrażone w dowolnych jednostkach.

Tradycyjnie stosuje się jednostki bezwzględne, aby zagwarantować, że element zostanie umieszczony w określonym miejscu na ekranie. W przypadku stron responsywnych zazwyczaj wykorzystuje się jednostki względne, dzięki czemu elementy mogą dostosowywać się do wielkości ekranu.

Najczęściej stosowane względne jednostki wymiarów to:

`%`
: Wartość procentowa opisuje element zawarty wewnątrz innego elementu.

`em`
: Em jest to względna jednostka wielkości tekstu elementu.

`rem`
: Rem jest to względna jednostka określająca wielkość tekstu dokumentu (tj. elementu `<html>`).

Poniżej wymienione są najczęściej stosowane jednostki bezwzględne:

`px`
: Piksel to jednostka bezwzględna, jednak nie zawsze jest taka sama dla wszystkich urządzeń.

`in`, `cm` oraz `mm`
: Fizyczne jednostki przeliczane na piksele, które również nie muszą być takie same dla różnych urządzeń. Na jeden cal przypada 96 pikseli, na jeden centymetr 37,8 piksela, a na jeden milimetr 3,78 piksela.

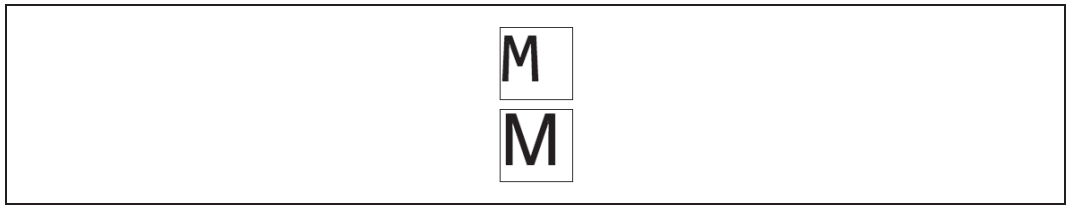
`pt`
: Punkt odpowiada 1/72 części cala. Częściej jest stosowany w wydrukach. W przypadku stron WWW może być przydatny w arkuszach stylów opisujących stronę do wydruku.

Jednostka em

W responsywnych stronach będziesz bardzo często stosował jednostkę em.

W minionej epoce składu drukarskiego jednostka **em** określała wielkość metalowej płytki, odpowiednio szerokiej, aby mogła się na niej zmieścić najszersza litera, czyli wielka litera **M.**

Wielu programistów przyjmuje, że komputerowa jednostka em również odpowiada szerokości litery M, ale tak w rzeczywistości nie jest. Określa ona po prostu wielkość czcionki użytej w elemencie. Jak pokazuje rysunek 4.6, jednostka ta nie zależy od wybranego kroju czcionki.

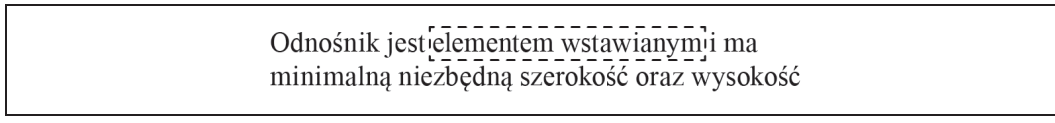


Rysunek 4.6. Oba prostokąty mają wysokość i szerokość równą dokładnie 1 em

Wysokość i szerokość elementów

W modelu pudełkowym każdy element ma określoną wysokość i szerokość, które niekiedy mogą być zmieniane za pomocą stylów CSS.

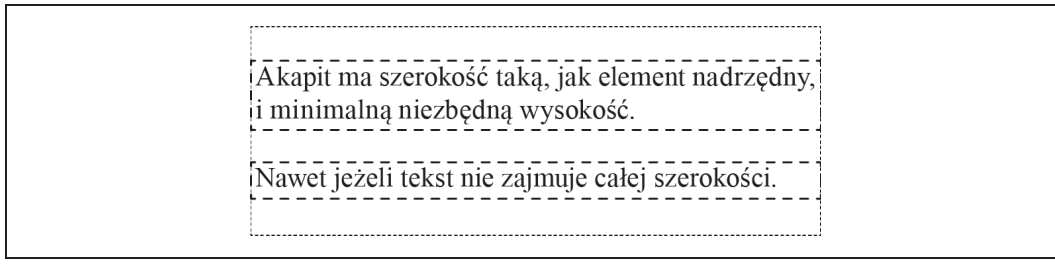
Wysokość i szerokość elementów wstawianych są dziedziczone po elementach nadrzędnych i nie można ich zmieniać. Elementy wstawiane są domyślnie na tyle szerokie i wysokie, aby mogła być wyświetlona ich zawartość, jak pokazuje rysunek 4.7.



Rysunek 4.7. Elementy wstawiane mają minimalną szerokość i wysokość niezbędną do wyświetlenia zawartości

Szerokość elementu blokowego jest domyślnie równa 100% szerokości elementu nadrzędnego, nawet jeżeli wyświetlana zawartość nie zajmuje całego miejsca.

Domyślnie elementowi nadawana jest wysokość odpowiednia do zawartej w nim treści. Na przykład akapit jest na tyle wysoki, aby zmieścił się w nim cały tekst zapisany w kilku wierszach, jak pokazuje rysunek 4.8.



Rysunek 4.8. Elementy blokowe mają domyślną szerokość odpowiadającą elementowi nadrzędnemu, natomiast wysokość odpowiednią do zawartej w nim treści

Wysokość i szerokość elementu blokowego możesz zmienić za pomocą stylu CSS:

```
div { width: 75%; height: 200px; }
```

Pamiętaj, że szerokość wyrażona w procentach odnosi się do szerokości elementu nadrzędnego, a więc w powyższym przykładzie szerokość elementu `<div>` będzie równa 75% szerokości innego elementu, w którym będzie on umieszczony.

Marginesy i odstępy

Każdy element umieszczony na stronie ma marginesy, czyli puste miejsce wokół zewnętrznych krawędzi, oraz odstępy, czyli puste miejsce wewnątrz krawędzi, otaczające zawartość. Marginesy i odstępy często dają ten sam efekt, chyba że element ma ramkę, która znajduje się na zewnątrz odstępów i wewnątrz marginesów.

Przeglądarki stosują domyślne odstępy i marginesy dla każdego elementu, ale nie są one takie same we wszystkich przeglądarkach. Z tego powodu stosowane są opisane wyżej arkusze resetujące, które ustawiają odstępy i marginesy wszystkich elementów na zero. Dzięki temu unika

się przypadkowego ustawienia różnych marginesów i odstępów w różnych przeglądarkach, przez co strona mogłaby nie wyglądać tak, jak byś tego oczekiwał.

Zazwyczaj w przypadku responsywnych stron szerokość lewego i prawego marginesu i odstępu określa się w procentach, dzięki czemu układ strony może być dostosowany do szerokości obszaru widoku. Na szerokich ekranach margines wokół treści będzie odpowiednio duży, natomiast na małych ekranach cenne miejsce nie będzie tracone na szerokie marginesy.

Marginesy i odstępy górne i dolne mogą być wyrażone w pikselach, ponieważ nie muszą być dostosowywane do wielkości obszaru widoku.

Każda z czterech stron elementu może mieć określoną własną szerokość marginesu i odstępu.

W poniższym przykładzie ustawiane są osobne wartości parametrów każdego brzegu elementu. Kolejność brzegów jest zgodna z ruchem wskazówek zegara — górny, prawy, dolny, lewy:

```
div {
  padding: 1px 2px 3px 4px;
  margin: 1px 2px 3px 4px;
}
```

W tym przykładzie górny margines i odstęp są ustawiane na 1 piksel, prawy na 2 piksele, dolny na 3 piksele i lewy na 4 piksele.

Jeżeli wymiary po przeciwnych stronach lub wszystkie wymiary są takie same, można je skonsolidować. Obowiązuje wtedy ta sama kolejność, jak poprzednio, a brakującym wymiarom zostaną nadane wartości takie same, jak po przeciwnej stronie pudełka. W poniższym przykładzie górny wymiar jest równy 1 pikselowi, prawy 2 pikselom, dolny jest równy górnemu (1 piksel), a lewy prawemu (2 piksele):

```
div {
  padding: 1px 2px;
  margin: 1px 2px;
}
```

Jeżeli zostaną podane trzy wartości, zostanie zastosowana ta sama zasada. Jeżeli określisz wymiary górny, prawy i dolny, wówczas brakującemu lewemu wymiarowi zostanie nadana wartość równa wymiarowi prawemu (2 piksele):

```
div {
  padding: 1px 2px 3px;
  margin: 1px 2px 3px;
}
```

Jeżeli zostanie podana tylko jedna wartość, będzie ona użyta do określenia każdego z czterech brzegów pudełka:

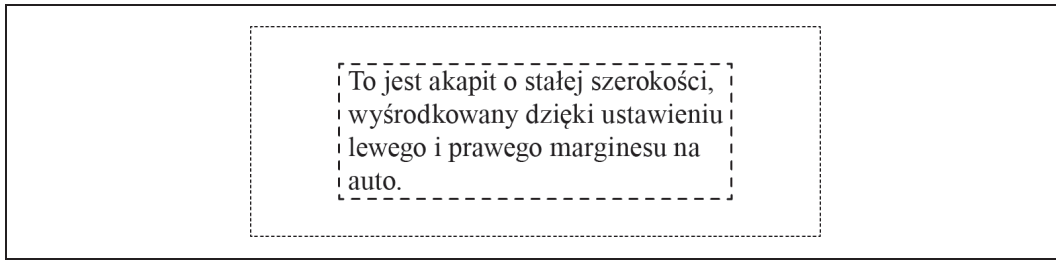
```
div {
  padding: 1px;
  margin: 1px;
}
```

Możesz również jeden z brzegów zdefiniować osobno, na przykład określić marginesy dla wszystkich akapitów, ale ustawić inną szerokość jednego z marginesów określonego akapitu:

```
p { margin: 5px 10%; }
.inny p { margin-top: 10px; }
```

Za pomocą atrybutów `padding-top`, `padding-right`, `padding-bottom`, `padding-left`, `margin-top`, `margin-right`, `margin-bottom` i `margin-left` możesz określić osobne szerokości każdego brzegu elementu. Oczywiście, jeżeli będziesz chciał je określić dla elementów tego samego typu, możesz je umieścić w jednej deklaracji w opisany wcześniej sposób.

Jeżeli chcesz umieścić element blokowy w środku nadrzędnego elementu, możesz to osiągnąć, ustawiając wartość lewego i prawego marginesu na `auto`. W ten sposób element zostanie umieszczony w środku kontenera, a wolne miejsce zostanie równo podzielone między lewy i prawy margines, jak pokazuje rysunek 4.9.



Rysunek 4.9. Ustawienie lewego i prawego marginesu na auto powoduje umieszczenie elementu w środku kontenera

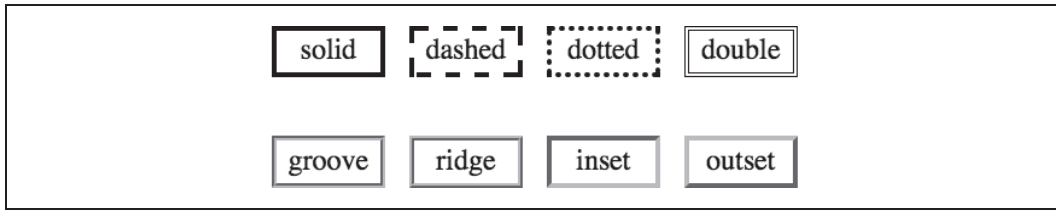
Ten wizualny efekt zostanie osiągnięty wtedy, gdy szerokość środkowanego elementu jest mniejsza niż 100% szerokości elementu nadrzędnego. W przeciwnym wypadku element wypełni całe dostępne miejsce.

Ramki

Ramki znajdują się między odstępami a marginesami i stanowią obwiednię elementu.

Definiując ramkę, musisz określić szerokość linii, jej styl i kolor. Szerokość ramki zazwyczaj jest wyrażana w pikselach.

Atrybut `border-style` może przyjmować między innymi wartości `solid` (linia ciągła), `dotted` (kropkowana) lub `dashed` (przerywana). Wartość `double` pozwala narysować podwójną linię, natomiast `groove`, `ridge`, `inset` oraz `outset` umożliwiają uzyskanie różnych efektów trójwymiarowych. Przykłady użycia tych wartości przedstawione są na rysunku 4.10.



Rysunek 4.10. Dostępne style ramek

Ramki możesz zdefiniować w jednej deklaracji, umieszczając wartości w dowolnej kolejności. W poniższym kodzie jest ustawiana ciągła czerwona ramka o szerokości 1 piksela dla elementu `<div>`:

```
div { border: 1px solid red; }
```

Każdy atrybut możesz też określić osobno, jeżeli chcesz to zrobić pojedynczo:

```
div { border-width: 2px; }
div { border-style: dotted; }
div { border-color: green; }
```

Możesz również określać każdy brzeg ramki osobno:

```
div { border-left-width: 2em; }
div { border-right-style: inset; }
div { border-top-color: blue; }
```

I tak dalej. Każda z trzech właściwości może być określona osobno dla górnego, dolnego, lewego i prawego brzegu ramki.

Wielkość pudełka

Teraz, gdy znasz już sposoby określania poszczególnych wymiarów pudełka, powinieneś wiedzieć, że określenie wielkości pudełka na stronie WWW jest trochę skomplikowane. Czy odstępy, ramki i marginesy stanowią części pudełka? To zależy od sytuacji.

Są dwa zasadnicze sposoby wskazywania przeglądarce, w jaki sposób mają być określane wszystkie wymiary. Nowa właściwość `box-sizing`, wprowadzona w wersji języka CSS3, pozwala wybrać jeden z dwóch modeli wymiarowania: `border-box` lub `content-box`.

W modelu `border-box` po określeniu szerokości elementu blokowego wszystkie odstępy i ramki są umieszczone wewnątrz elementu. W przeciwnym wypadku, w domyślnym modelu `content-box`, odstępy i ramki są umieszczone na zewnątrz elementu.

Model `border-box` jest zazwyczaj prostszy w użyciu, jednak nie jest on stosowany domyślnie, więc aby go użyć, musisz określić właściwość `box-sizing: border-box`.

Ta właściwość nie jest obsługiwana przez wszystkie przeglądarki, więc musisz użyć prefiksów producentów. Przeglądarki Internet Explorer w wersji 8 i nowszej oraz Opera obsługują tę właściwość, więc musisz użyć tylko prefiksów dla przeglądarek WebKit i Mozilla:

```
div {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
```

Możesz również określić właściwość `border-box` dla całej witryny za pomocą następującego kodu:

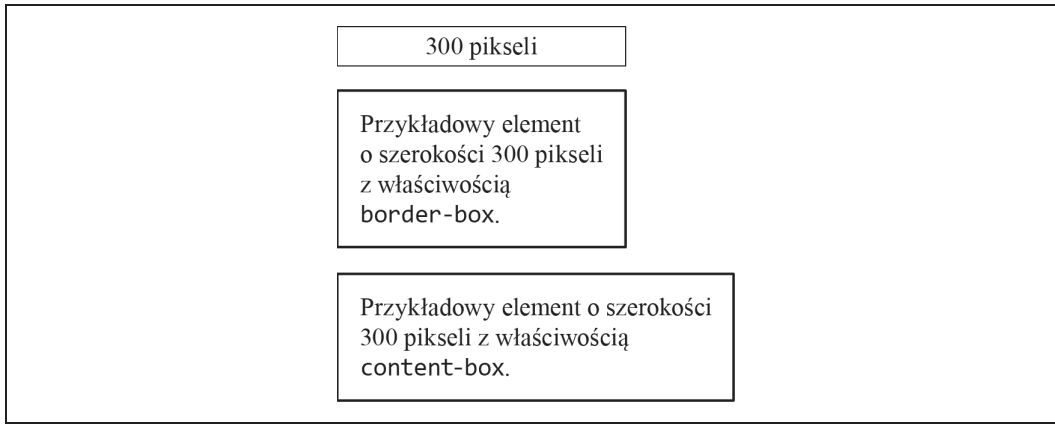
```
*, *:before, *:after {
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
}
```

Gwiazdki oznaczają, że styl CSS powinien być zastosowany w każdym elemencie.

Oba akapity na rysunku 4.11 mają szerokość 300 pikseli, z każdej strony odstępy o szerokości 20 pikseli i ramki o szerokości 5 pikseli. W pierwszym akapicie odstępy i ramka znajdują się wewnątrz, natomiast w drugim akapicie na zewnątrz elementu o szerokości 300 pikseli.

UWAGA

Model `box-sizing` nie jest obsługiwany przez przeglądarkę Internet Explorer w wersji 7 lub starszej, dlatego nawet jeżeli określisz tę właściwość, zostanie użyty domyślny model `content-box`. Podczas testowania swojej strony możesz sprawdzić, czy będą potrzebne dodatkowe style CSS rozwiązujące problemy z układem strony. Jest również dostępny kod wypełnienia, dzięki któremu właściwość `box-sizing` jest obsługiwana przez przeglądarki Internet Explorer w wersjach 6 i 7. Kod ten, utworzony przez autorów projektów Boilerplate, Modernizm i CSS3 Please, jest dostępny na stronie HTML5 Please (*http://html5please.com/#box-sizing*).



Rysunek 4.11. Różnica w wyświetlaniu elementu blokowego spowodowana jedynie wybraniem modelu border-box lub content-box

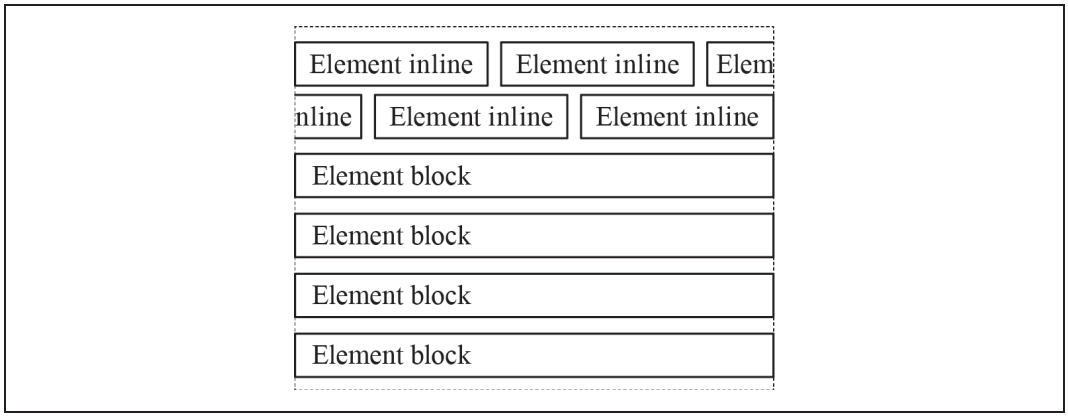
Właściwość display

Jedną z kluczowych właściwości elementów języka HTML jest `display`, wpływająca na sposób umieszczenia elementu na ekranie. W większości elementów ma ona wartość `block` (element blokowy) lub `inline` (wstawiany). Elementy blokowe są umieszczane w stosie jeden nad drugim, natomiast elementy wstawiane są wyświetlane jeden za drugim w tekście, jak pokazuje rysunek 4.12.

Właściwość `display` jest częścią stylu CSS i jej wartość można zmienić w dowolnym elemencie. Jednak powinieneś znać jej domyślną wartość dla różnych elementów, których używasz.

Przykładem elementu blokowego jest akapit. Popatrz na rysunek 4.12 i wyobraź sobie, że każdy element blokowy jest akapitem. Każdy z nich będzie automatycznie rozpoczynał się od nowego wiersza, a każdy następny również będzie umieszczany w nowym wierszu. Inne elementy blokowe to między innymi elementy strukturalne, na przykład `<nav>`, `<article>`, nagłówki oraz element `<div>`.

Elementy blokowe zawsze wypełniają w poziomie nadrzędny element, chyba że ręcznie ustawisz ich szerokość na inną wartość. Na przykład element blokowy umieszczony wewnątrz elementu `<body>` będzie miał taką samą szerokość, jak okno przeglądarki. Każdy element blokowy umieszczany jest na początku nowego wiersza, podobnie jak elementy znajdujące się za nim, niezależnie od tego, czy są to elementy blokowe czy wstawiane.



Rysunek 4.12. Elementy inline są umieszczane jeden za drugim w tekście, natomiast elementy block w stosie jeden nad drugim

Natomiast elementy wstawiane umieszczane są wewnątrz tekstu. Przykładowym elementem wstawianym jest `<strong>`, obejmujący kilka słów wewnątrz akapitu, które mają być pogrubione. Inne elementy wstawiane to `<em>`, `<span>` i odnośniki.

Elementy wstawiane mają niezbędną wymaganą wielkość. Słowa umieszczone wewnątrz elementu wstawianego są umieszczane w akapicie tam, gdzie powinny być umieszczone, i nie są przenoszone do nowego wiersza, podobnie jak znajdujący się za nimi tekst.

Oprócz elementów blokowych i wstawianych są jeszcze elementy `list-item`, stanowiące pozycje w liście punktowanej `<li>`. W większości przypadków są one wyświetlane podobnie jak elementy blokowe. Tabele, wiersze i komórki tabeli mają osobne właściwości określające sposób ich wyświetlania na stronie.

Właściwość `display` może przyjmować jeszcze kilka innych, rzadko używanych wartości (lub nieużywanych nigdy). Ich pełną listę możesz zobaczyć na stronie Mozilla Developer Network w sekcji *display* (*https://developer.mozilla.org/en-US/docs/CSS/display*).

Chociaż w większości elementów będziesz stosował domyślne wartości właściwości `display`, możesz jednak użyć kodu CSS, aby ją zmienić. Na przykład często zmienia się właściwość pozycji w liście na `inline`, dzięki czemu są one umieszczane poziomo, a nie pionowo:

```
li { display: inline; }
```

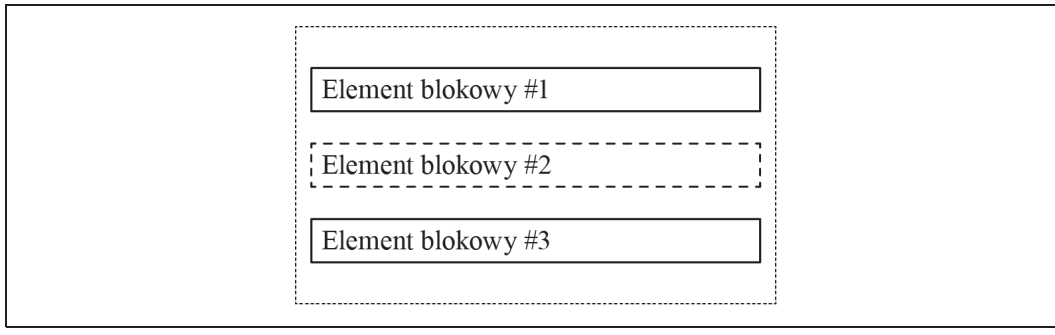
Pozycjonowanie elementów

Kluczową kwestią umożliwiającą umieszczenie elementów na stronie tam, gdzie powinny być, jest pozycjonowanie. Jest to dość skomplikowane zagadnienie, więc jeżeli nie jesteś programistą, nie przejmuj się, jeżeli nie wiesz dokładnie, na czym ono polega. Musisz jedynie znać istotne różnice pomiędzy różnymi sposobami pozycjonowania.

Domyślnie elementy wstawiane są umieszczane na stronie w jednym ciągu w kierunku od lewego do prawego brzegu, przy czym tam, gdzie trzeba, ciąg jest zawijany do nowego wiersza. Natomiast elementy blokowe są umieszczane w kierunku od góry do dołu, jak pokazuje rysunek 4.12. Ten porządek można zmienić za pomocą różnych właściwości.

Położenie statyczne

Domyślną wartością właściwości `position` jest `static`. Elementy są po prostu umieszczane tam, gdzie powinny być, jak pokazuje rysunek 4.13.

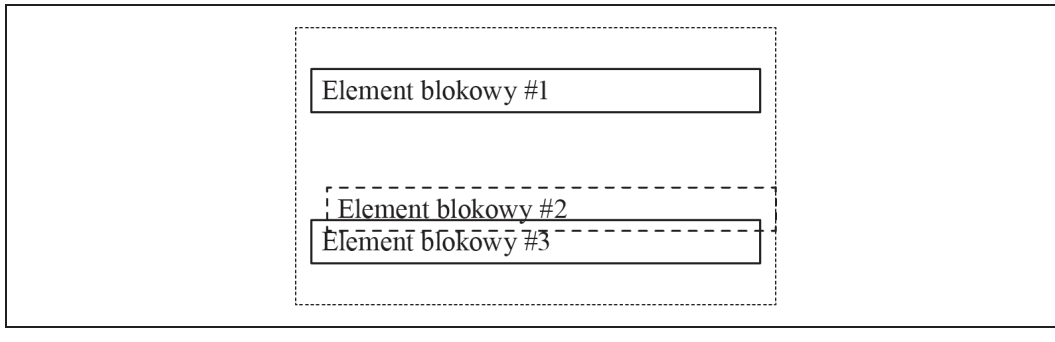


Rysunek 4.13. Elementy blokowe z domyślną wartością właściwości position są umieszczane w stosie

Położenie względne

Można określić względne położenie elementu, co spowoduje przeniesienie go w inne miejsce, bez wpływu na ciąg innych elementów na stronie. Na przykład drugi akapit na rysunku 4.14 ma położenie względne i jest przesunięty w dół o 40 pikseli i w lewo o 40 pikseli:

```
p.drugi { position: relative; left: 40px; top: 40px; }
```



Rysunek 4.14. Drugi element ma ustawione położenie względne

Przeglądarka pozostawi puste miejsce tam, gdzie pierwotnie *powinien* znajdować się dany element, po czym przemieści go w zadanym kierunku.

Jak widać, ten sposób powoduje nakładanie się elementów na stronie, a nie przesuwanie ich w celu zrobienia miejsca. Po prawej stronie ekranu nie ma miejsca, aby można było przesunąć element o 40 pikseli, więc część elementu wychodzi poza ekran i nie jest widoczna dla użytkownika.

Zwróć uwagę, że umieszczony w ten sposób element znajduje się na wierzchu pozostałych elementów na stronie.

Aby użyć tej metody pozycjonowania, musisz najpierw nadać właściwości `position` wartość `relative`, a następnie określić zmianę położenia elementu w kierunku poziomym, pionowym lub w obu kierunkach.

Położenie bezwzględne

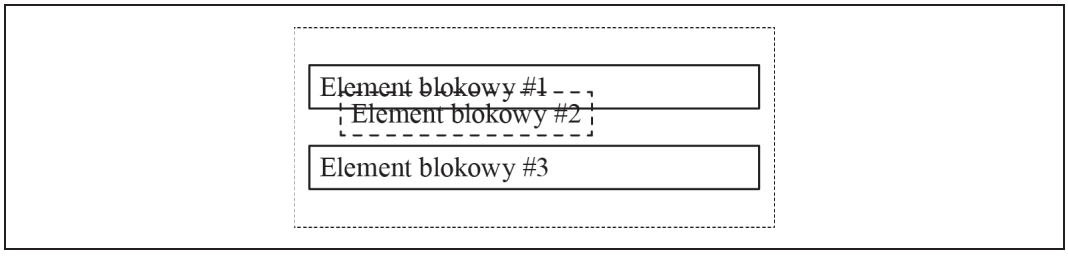
Położenie bezwzględne funkcjonuje inaczej. Przeglądarka przesuwa element zgodnie z definicją, ale nie pozostawia pustego miejsca tam, gdzie pierwotnie się on znajdował. Ponadto, zamiast przemieszczać go względem pierwotnej pozycji w ciągu elementów w dokumencie, przeglądarka umieszcza dany element w pierwszym elemencie nadrzędnym, którego właściwość `position` ma wartość inną niż `static`.

W poniższym przykładzie w elemencie `<div>` są umieszczone trzy elementy blokowe:

```
p.drugi { position: absolute; left: 40px; top: 40px; }
```

Element nr 2 ma położenie bezwzględne, więc nie jest ono uwzględniane w ciągu innych elementów. Zamiast tego element jest umieszczany w odległości 40 pikseli w dół i 40 pikseli w prawo od górnego lewego wierzchołka nadrzędnego elementu `<div>`.

Ponieważ nie jest pozostawiane puste miejsce po tym elemencie, po prostu nakłada się on na inne elementy, jak pokazuje rysunek 4.15.



Rysunek 4.15. Drugi element blokowy ma określone położenie bezwzględne

Nietypową rzeczą, którą zapewne zauważyłeś w przypadku położenia bezwzględnego, przedstawionego na rysunku 4.15, jest szerokość elementu. Odpowiada ona zawartości danego elementu, a nie wymiarowi elementu nadrzędnego, który jest normalnie przyjmowany w przypadku elementów blokowych. Elementy z położeniem bezwzględnym nie mają domyślnej pełnej szerokości, jak inne elementy.

Element zawierający więcej tekstu jest rozszerzany do momentu, aż osiągnie prawą krawędź elementu nadrzędnego, po czym tekst jest zawijany.

UWAGA

Jak zauważyłeś, w niektórych przypadkach elementy nakładają się na siebie. Aby określić, który element będzie umieszczony na wierzchu, zmień wartość właściwości `z-index`. Element z największą wartością zostanie umieszczony na wierzchu. Dopuszczalne są wartości ujemne:

```
p.pierwszy { z-index: 50; }
p.drugi { position: relative; left: 40px; top: 40px; z-index: 25; }
```

Położenie stałe

Położenie stałe jest podobne do położenia bezwzględnego, przy czym element jest umieszczany względem całej strony, a nie elementu nadrzędnego. Dlatego jeżeli w poprzednim przykładzie zostanie zastosowane położenie stałe, to drugi element będzie umieszczony w odległości 40 pikseli w dół i 40 pikseli w prawo od górnego lewego wierzchołka strony.

Każdy element z określonym położeniem stałym będzie znajdował się w zadanym miejscu strony, nawet jeżeli użytkownik będzie ją przewijał. Dlatego używaj tej metody, jeżeli chcesz, aby sekcja nawigacji lub inny element zawsze znajdowały się na górze strony. Pamiętaj, że inne elementy nie będą uwzględniały położenia stałego elementu i będą na siebie zachodzić.

W przypadku przeglądarek mobilnych położenie stałe funkcjonuje dość nietypowo. Szczegółowe informacje zawarte są w artykule Brada Frosta *Fixed Positioning in Mobile Browsers* („Położenie stałe w przeglądarkach mobilnych", *http://bradfrostweb.com/blog/mobile/fixed-position*).

Kierunki w położeniu elementów

Jeżeli określone jest względne, bezwzględne lub stałe położenie elementu, to podane liczby oznaczają odpowiednio przesunięcie w lewo, w prawo, w górę lub w dół.

Możesz użyć właściwości `left` lub `right` (w lewo lub w prawo, ale nie obu jednocześnie), aby przesunąć element poziomo, albo `top` lub `bottom` (w górę lub w dół, ale nie obu jednocześnie), aby przesunąć element pionowo. Jeżeli przez przypadek umieścisz sprzeczne deklaracje (na przykład `left` i `right`), wtedy przeglądarka pominie pierwszą z nich.

Jeżeli nie chcesz przesuwać elementu poziomo i pionowo *jednocześnie*, możesz określić tylko jedną właściwość położenia.

Zwróć uwagę, że podczas określania kierunku element jest przemieszczany *z zadanej*, a nie *w zadaną* stronę. Na przykład deklaracja `left: 40px;` powoduje *odsunięcie* elementu o 40 pikseli od lewego brzegu, więc w rzeczywistości element jest przesuwany w prawo.

Możesz również stosować liczby ujemne. W przypadku położenia względnego przesunięcie elementu o 40 pikseli w lewo daje ten sam efekt, co przesunięcie o –40 pikseli w prawo. Ta sama zasada dotyczy kierunków w górę i w dół, ponieważ przesunięcie następuje tylko względem początkowego położenia.

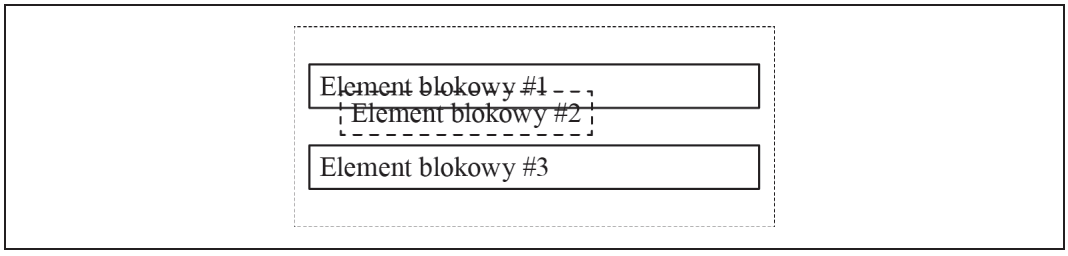
Jednak reguła ta nie obowiązuje w przypadku położenia bezwzględnego i stałego. W poniższym przykładzie element zostanie umieszczony w odległości 40 pikseli od górnego i 40 pikseli od lewego brzegu elementu nadrzędnego (patrz rysunek 4.16):

```
p.drugi { position: absolute; left: 40px; top: 40px; }
```

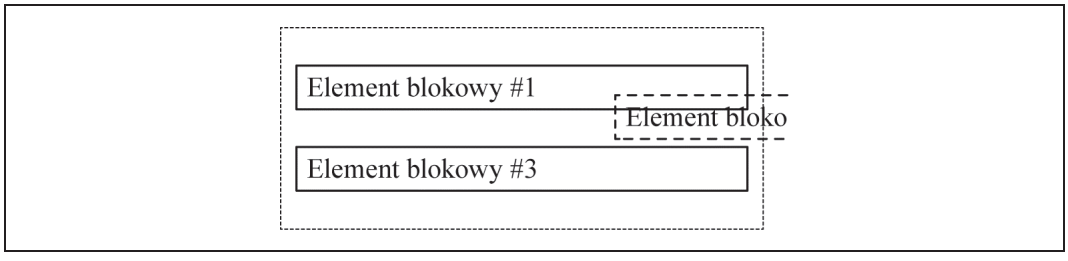
Natomiast umieszczenie tego elementu w odległości –40 pikseli od prawego brzegu spowoduje wyrównanie go do prawej krawędzi elementu nadrzędnego, a następnie przesunięcie o 40 pikseli w prawo (patrz rysunek 4.17):

```
p.drugi { position: absolute; right: -40px; top: 40px; }
```

Pozycjonowanie elementów może być dość skomplikowane. Najlepszym sposobem opanowania go jest utworzenie strony testowej i wypróbowanie wszystkich opisanych wyżej właściwości, aby lepiej poznać ich znaczenie.



Rysunek 4.16. Drugi element blokowy ma określone położenie bezwzględne, 40 pikseli od lewego górnego wierzchołka



Rysunek 4.17. Drugi element blokowy jest umieszczony w odległości –40 pikseli od prawej krawędzi

Właściwości float i clear

Właściwości `float` i `clear` również umożliwiają zmianę położenia elementu na stronie, aczkolwiek nie są one związane z właściwością `position`.

Właściwość `float` stosuje się w celu umieszczenia treści wokół określonego elementu. Załóżmy na przykład, że za pomocą poniższego kodu umieściłeś obraz pomiędzy dwoma akapitami (patrz rysunek 4.18):

```
<p>...</p>
<img src="obraz.jpg">
<p>...</p>
```

Jeżeli nie chcesz pozostawiać pustego miejsca na stronie, możesz wypełnić je tekstem, jak pokazuje rysunek 4.19.

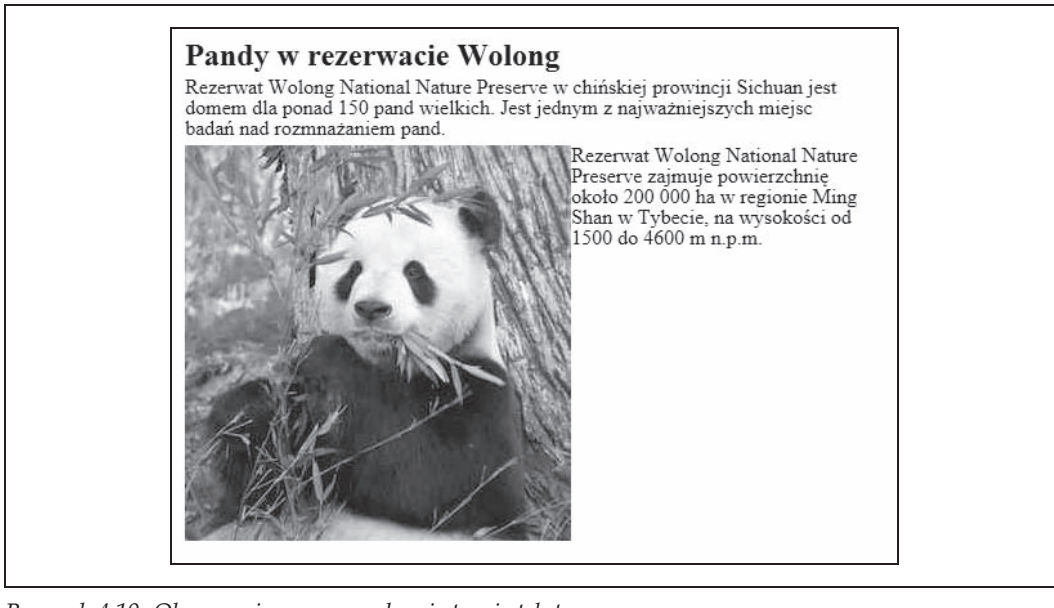
Właściwość `float` zleca przeglądarce umieszczenie elementu po lewej lub prawej stronie elementu nadrzędnego i otoczenie go następnymi elementami, blokowymi lub wstawianymi.

Oczywiście, nie możesz przesunąć tak samo wszystkich obrazów na stronie, musisz je w jakiś sposób rozróżniać, na przykład za pomocą klas. Możesz utworzyć klasę, na przykład `tekst_z_lewej`, którą zastosujesz we wszystkich elementach wyrównanych do prawej krawędzi:

```
.tekst_z_lewej { float: left; }
```

Pamiętaj, że wszystkie kolejne elementy będą umieszczane wokół przesuniętego elementu, dopóki nie zostanie wypełnione dostępne miejsce. W każdej chwili możesz anulować otaczanie, na przykład przez nagłówki, jak na rysunku 4.20.

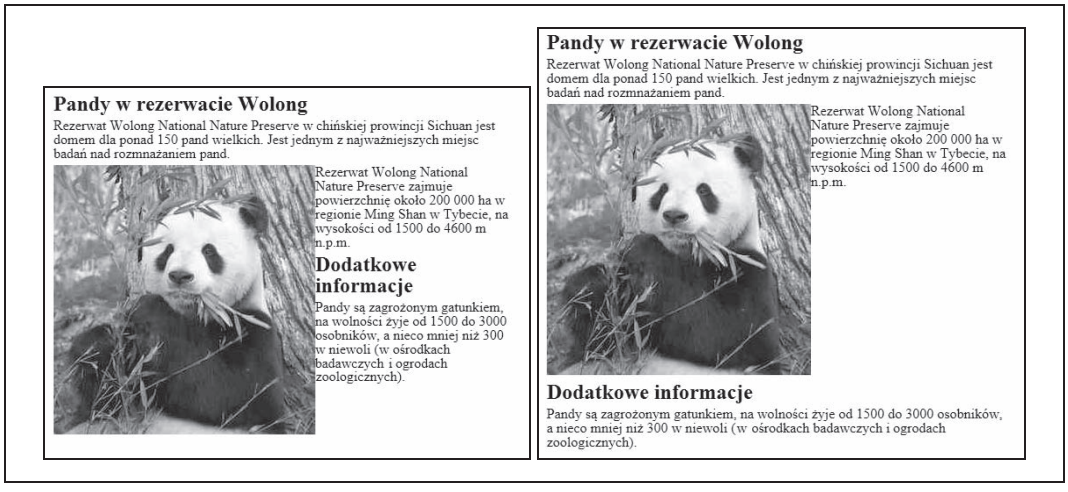
Rysunek 4.18. Obraz umieszczony pomiędzy dwoma akapitami bez otaczania



Rysunek 4.19. Obraz umieszczony po lewej stronie tekstu

Jeżeli chcesz całkowicie anulować otaczanie, użyj po prostu właściwości `clear` w pierwszym elemencie, który ma być umieszczony w nowym wierszu:

```
ul { clear: left; }
```



Rysunek 4.20. W przykładzie po prawej stronie anulowane jest otaczanie przesuniętego obrazu przez nagłówek

Przy anulowaniu otaczania właściwości `left` i `right` muszą odpowiadać poprzednio użytym właściwościom.

Style podstawowe

Wróćmy do naszej strony o pandzie i zastosujmy w niej kilka podstawowych stylów, aby nadać jej lepszy wygląd.

Przede wszystkim, aby zrobić to w prosty, a nie skomplikowany sposób, zastosujmy we wszystkich elementach omówioną wcześniej właściwość `box-sizing`:

```
*, *:before, *:after {
  -moz-box-sizing: border-box;
  -webkit-box-sizing: border-box;
  box-sizing: border-box;
}
```

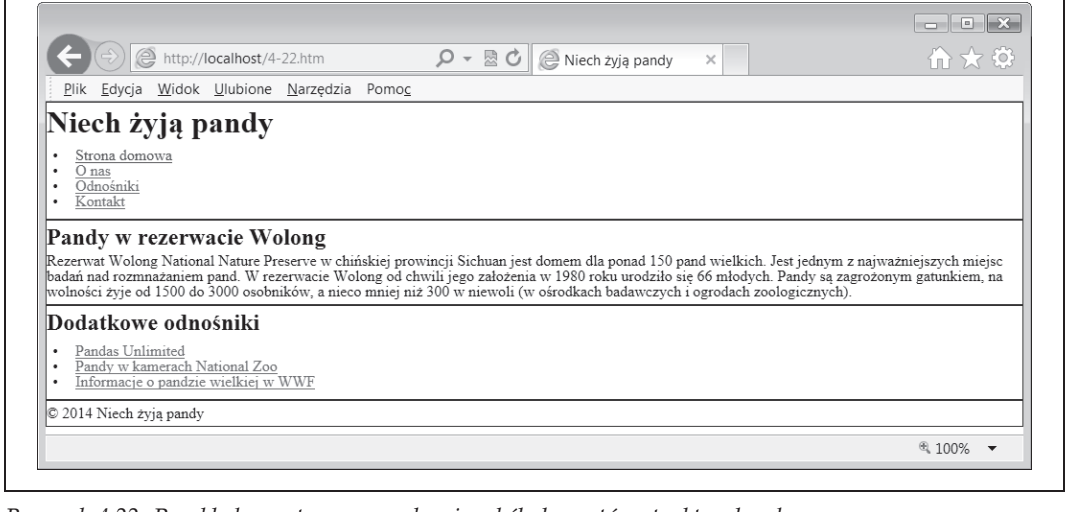
Określiliśmy już wielkość czcionki dla nagłówków, akapitów i list, jak również pogrubiliśmy nagłówki. Teraz określimy marginesy i odstępy, aby nieco rozsunąć elementy. Dodamy również punktowanie do listy, jak na rysunku 4.21:

```
h1 { font-size: 2em; }
h2 { font-size: 1.5em; }
h1, h2 { font-weight: bold; margin: 5px 0; }
p { font-size: 1em; margin: 5px 0; }
ul { padding-left: 10px; margin: 10px 0; list-styletype: disc; }
li { margin-left: 10px; padding-left: 10px; }
```

Następnie dodamy ramki do elementów strukturalnych, aby łatwiej było rozróżnić, gdzie co jest na stronie. Nawet jeżeli w ostatecznej wersji projektu nie zastosujesz ramek, warto je umieścić podczas tworzenia strony, ponieważ można dzięki temu łatwo sprawdzić efekt zastosowania marginesów i odstępów. Poniżej przedstawiony jest odpowiedni kod (efekt przedstawia rysunek 4.22):

```
header, article, aside, footer { border: 1px solid #111; }
```

Rysunek 4.21. Przykładowa strona z określonymi marginesami, odstępami i punktowanymi listami

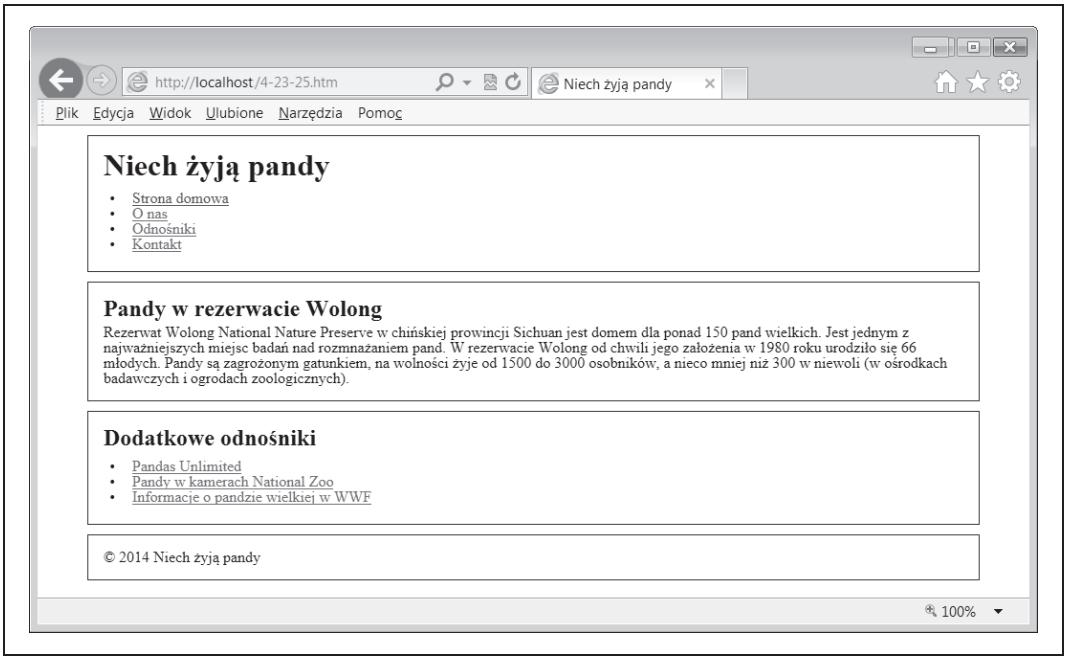


Rysunek 4.22. Przykładowa strona z ramkami wokół elementów strukturalnych

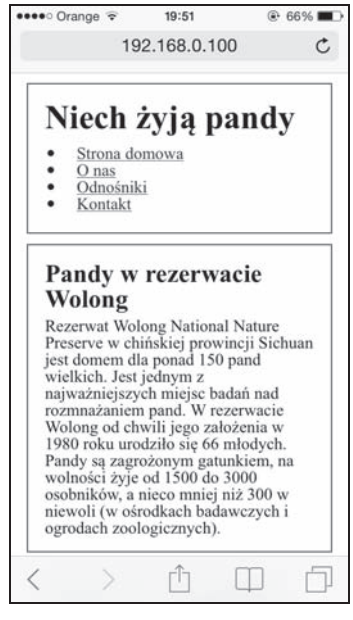
Aby strona wyglądała nieco lepiej, określimy odstępy i marginesy dla elementów strukturalnych, jak na rysunku 4.23:

```
header, article, aside, footer { border: 1px solid #111;
padding: 10px 1em; margin: 0 5% 10px; }
header { margin-top: 10px; }
```

Jak zapewne zauważyłeś, podczas definiowania stylów dla naszej strony nie został poruszony temat stron responsywnych. Jeżeli pamiętasz, zaczęliśmy od zwykłego kodu HTML bez żadnych stylów, który domyślnie stanowi stronę responsywną. Jak dotąd nie zrobiliśmy niczego, co sprawiłoby, że strona *nie* byłaby responsywna. Może być wyświetlona na ekranach o różnych wielkościach, na przykład takich jak na rysunkach 4.24 i 4.25, aczkolwiek w przypadku szerszych ekranów tekst jest rozciągnięty i mniej czytelny.

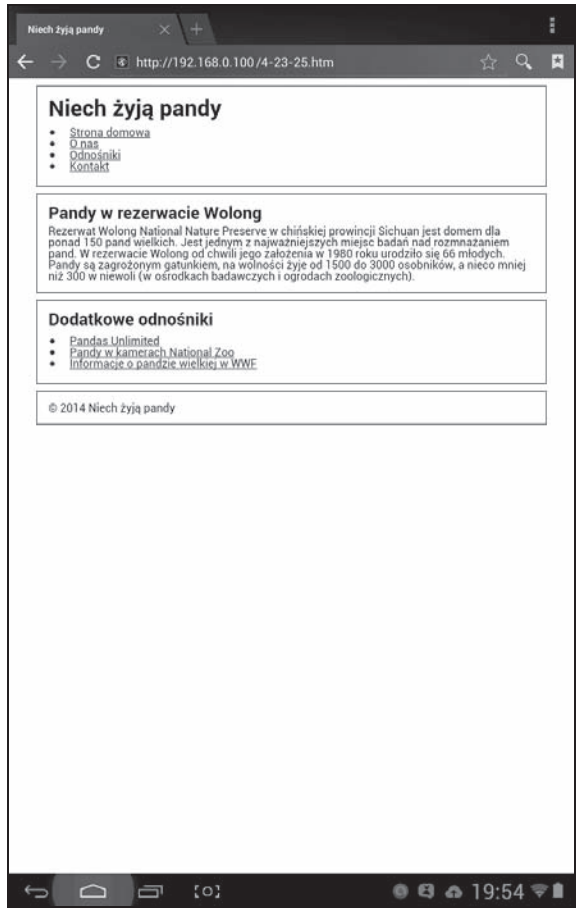


Rysunek 4.23. Przykładowa strona z odstępami i marginesami oddzielającymi elementy



Rysunek 4.24. Przykładowa strona wyświetlona na małym ekranie telefonu komórkowego

Gdy przejdziemy do omówienia zapytań o media w rozdziale 5., dowiesz się, jak sprawić, aby strona zmieniała swój układ w zależności od szerokości ekranu.



Rysunek 4.25. Przykładowa strona wyświetlona na średniej wielkości ekranie tabletu

Podsumowanie

W tym rozdziale dowiedziałeś się, że język CSS ma swoje wersje, podobnie jak HTML. Ostatnia wersja, CSS3, oferuje wiele nowych właściwości elementów, jak również zapytania o media, które są podstawą responsywnych stron.

Mimo że niektóre właściwości wprowadzone w wersji CSS3 są jeszcze w fazie testów, można je wykorzystywać w projektach, stosując prefiksy, dzięki którym przeglądarki właściwie zinterpretują testowe wersje właściwości elementów.

Style CSS mogą być zastosowane na stronie w formie osobnych plików, zwanych arkuszami stylów, lub umieszczone w kodzie jako style osadzone lub wstawione. Zazwyczaj lepszym sposobem jest użycie arkuszy stylów, ponieważ umożliwiają one globalne stosowanie stylów na całej stronie lub w jej poszczególnych sekcjach.

Kaskada jest to zestaw szczegółowych reguł określających zasady stosowania stylów na stronie — ich kolejność i priorytety względem siebie. Reguły oznaczone jako ważne mają najwyższy priorytet, po nich następują style wstawiane, następnie style z identyfikatorami, klasami, elementami, pseudoelementami, potem style odziedziczone i wartości domyślne. Przeglądarka rozstrzyga o użytym stylu na podstawie szczegółowości i kolejności stylów.

Kaskada może być bardzo skomplikowana, jednak jeżeli zostanie przemyślana przed zastosowaniem stylów w stronie, to używanie jej w sposób szeroki i uporządkowany może sprawić, że implementacja stylów będzie prosta, wymagająca mniejszego nakładu pracy i sprawiająca mniej kłopotów.

Dzięki ustawieniu na samym początku domyślnych wartości właściwości stylów CSS możesz mieć pewność, że wszystkie przeglądarki będą stosowały style zgodnie z oczekiwaniami. Jeżeli nie chcesz sam tego robić, w Internecie znajdziesz wiele arkuszy resetujących.

Nie ma jednej słusznej metody porządkowania stylów w arkuszu, ale zawsze warto stosować spójny system, dzięki czemu później łatwiej będzie odnaleźć potrzebny styl. Możesz grupować style według stron lub sekcji, według podobnych cech, na przykład wyglądu czcionki.

Najważniejszym pojęciem w stylach CSS jest model pudełkowy, określający sposób wyświetlania elementów na stronie. W tym modelu każdy element jest traktowany jak prostokątne pudełko. Każdy z nich ma określoną szerokość i wysokość, może mieć ramki, odstępy i marginesy.

Wszystkim właściwościom elementów są na początku przypisywane wartości domyślne, które można zmieniać, wykorzystując różne jednostki długości. W stronach responsywnych większość wymiarów powinna być wyrażona w jednostkach względnych, na przykład procentach, emach lub remach. Wymiary pionowe mogą być wyrażone w jednostkach bezwzględnych, na przykład pikselach. Możesz również stosować wartość `auto`, powodującą wyliczenie wartości wymiaru elementu na podstawie wymiarów przylegających lub nadrzędnych elementów.

Właściwość `box-sizing` umożliwia wybranie sposobu wyświetlania odstępów, marginesów i ramek, tj. określenie, czy ich wymiary mają zawierać się w szerokości elementu czy nie. Pozycjonowanie elementów określa ich położenie na stronie.

Teraz, kiedy poznałeś podstawy funkcjonowania stylów CSS, przejdziemy do rozdziału 5., w którym dowiesz się, jak stosować zapytania o media, aby utworzyć responsywną stronę.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z

O'REILLY®

Responsywne strony WWW
dla każdego

Responsywne strony WWW umiejętnie dostosowują się do rozmiaru każdego ekranu, na którym są wyświetlane. Ta ich właściwość pomaga również generować zyski. Po pierwsze, wystarczy utrzymywać tylko jedną wersję responsywnej strony. Po drugie, potencjalni klienci i użytkownicy mogą zapoznać się z Twoją ofertą lub skorzystać z niej praktycznie w dowolnym miejscu świata. I to niezależnie od tego, czy korzystają ze smartfona, z tabletu czy z komputera stacjonarnego!

Jeżeli chcesz nauczyć się tworzyć takie strony, ta książka będzie dla Ciebie najlepszym źródłem! Dowiesz się, w jaki sposób zbudować skalowalne strony WWW z wykorzystaniem najnowszych możliwości kaskadowych arkuszy stylów CSS3 oraz języka HTML5. Przekonasz się, jak łatwo możesz modyfikować style pod konkretne media oraz jak najlepiej zorganizować proces projektowania responsywnej strony WWW. Zaznajomisz się też z najlepszymi technikami poprawy wydajności Twoich stron WWW. To jest lektura obowiązkowa dla wszystkich programistów i projektantów stron WWW!

Dzięki tej książce:

- poznasz możliwości responsywnych stron WWW
- zoptymalizujesz pliki graficzne
- zaznajomisz się z procesem tworzenia strony responsywnej
- podniesiesz wydajność Twojej strony WWW

Odkryj możliwości responsywnych stron WWW!

helion.pl
księgarnia internetowa

Nr katalogowy: 28923

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

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

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

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-0035-4

Cena 59,00 zł