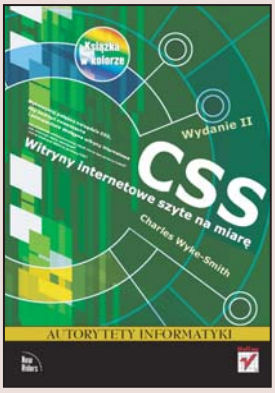
CSS. Witryny internetowe szyte na miarę. Autorytety informatyki. Wydanie II



Autor: Charles Wyke-Smith
Tłumaczenie: Łukasz Piwko
ISBN: 978-83-246-1756-2
Tytuł oryginału: Stylin with CSS: A Designers Guide (2nd Edition) (Voices That Matter)
Format: 168x237, stron: 320

Wykorzystaj potężne narzędzia CSS, aby tworzyć nowoczesne i powszechnie dostępne witryny internetowe

- Jak zbudować wielokolumnowy układ strony bez pomocy tabeli?
- Jak utworzyć menu rozwijalne?
- Jak formatować tekst za pomocą CSS?

Technologia CSS, czyli kaskadowych arkuszy stylów, powstała w celu odseparowania struktury dokumentu od formy jego prezentacji. Wykorzystanie kaskadowych arkuszy stylów przyspiesza tworzenie stron internetowych, a zgromadzenie wszystkich informacji dotyczących wyglądu tekstu i układu elementów w jednym miejscu ułatwia ich modyfikację oraz aktualizację. Ponadto technologia ta daje możliwość zdefiniowania wyglądu strony dla różnych mediów, takich jak ekran, palmtop, dokument w druku, czytnik ekranowy czy drukarki Braille'a. Krótko mówiąc, CSS umożliwia budowanie nowoczesnych, wygodnych i powszechnie dostępnych witryn internetowych.

W książce tej znajdziesz mnóstwo inspirujących technik i przypadków zaczerpniętych z prawdziwych projektów, a także bibliotekę szablonów stron, formularzy, menu, list oraz tabel, dzięki którym nauka i tworzenie własnych stron będzie przebiegać znacznie efektywniej. Korzystając z tego podręcznika, nauczysz się precyzyjnie pozycjonować elementy przy użyciu własności pływania, marginesów ujemnych, dopełnienia oraz pozycjonowania bezwzględnego. Dowiesz się, jak tworzyć układ strony automatycznie dopasowujący się do rozmiaru okna oraz jak ustawiać obrazy tła. Poznasz wszystkie reguły, które pozwolą Ci kreować wyjątkowe projekty zgodne z obowiązującymi standardami.

- XHTML – strukturalizacja treści
- Podstawy CSS
- Style lokalne, osadzone i zewnętrzne
- Klasy i identyfikatory
- Selektory i deklaracje
- Formatowanie tekstu
- Pozycjonowanie elementów
- Projektowanie komponentów interfejsu
- Formularze, listy i menu
- Tworzenie kompletnych witryn internetowych
- Architektura witryny

Wykorzystaj CSS i zrób wrażenie na użytkownikach Twojej witryny!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 4

Pozycjonowanie elementów

Jedną z kluczowych zmian związanych z wprowadzeniem standardów sieciowych była rezygnacja z tabel jako szkieletu strony. Tabele tak naprawdę nigdy nie były do tego przeznaczone. Ich funkcją było przechowywanie danych w podobny sposób, jak przechowuje się dane w Excelu. Jednak przed opracowaniem kaskadowych arkuszy stylów układ strony opierano na tabelach, które tworzyły siatkę utrzymującą poszczególne kolumny elementów. Wymuszało to stosowanie różnych sztuczek, jak puste obrazy GIF, złamania wiersza i twarde spacje. Za pomocą CSS można z dużą precyzją rozmieścić elementy na stronie bez potrzeby stosowania żadnych elementów prezentacyjnych.

Przy użyciu własności CSS, jak marginesy, dopełnienie i obramowanie, oraz takich technik, jak elementy pływające i własność clear, można uzyskać takie same, a nawet lepsze efekty jak kiedyś. Wszystko to jest możliwe przy zachowaniu czystości kodu XHTML oraz współdzieleniu reguł stylistycznych przez te same elementy na różnych stronach. Dzięki temu powstają lekkie i łatwe w modyfikowaniu strony.

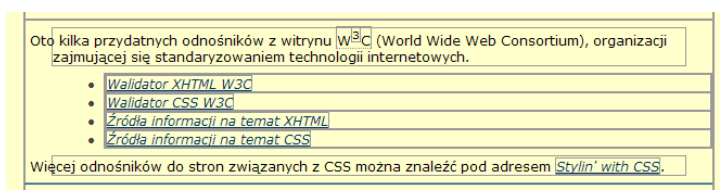
Sukces twórcy strony używającego wymienionych technik jest całkowicie uzależniony od tego, jak dobrze pozna i zrozumie **model blokowy** (ang. *box model*) oraz własności `position` i `display`. Na model blokowy składają się narzędzia służące do kontroli położenia elementów na stronie. Własność `position` służy do określania wzajemnych relacji położenia elementów. Własność `display` pozwala określić, czy elementy mają układać się jeden pod drugim, jeden obok drugiego lub czy w ogóle mają być widoczne. Zaczniemy od modelu blokowego.

Model blokowy

Każdy element użyty w kodzie XHTML tworzy na stronie blok. W związku z tym strona XHTML w rzeczywistości składa się z pewnej liczby wzajemnie ułożonych bloków.

Domyślnie obramowanie każdego bloku jest niewidoczne, a tło przezroczyste, dlatego niektórzy mogli się zastanawiać, gdzie są te bloki. W CSS włączenie obramowania i pokolorowanie tła jest banalnie proste. Pozwala to na zobaczenie struktury strony w całej krasie.

Na przykład rysunek 4.1 przedstawia tę samą stronę, którą zajmowaliśmy się pod koniec poprzedniego rozdziału, tylko z pokazanymi ramkami wokół elementów.



RYSUNEK 4.1. Dzięki pokazaniu ramek widać, że elementy liniowe łączy ciasno otaczają swoją zawartość, natomiast elementy blokowe, jak elementy listy, rozciągają się na całą szerokość strony (z wyjątkiem tych, które mają ustawione marginesy). Należy również zauważyć, że ujemne marginesy akapitów wychodzą poza blok elementu

W tym przypadku pozwalamy przeglądarce automatycznie ustawić elementy na stronie. Aby tworzyć bardziej interesujące układy strony niż domyślny, zaprezentowany na powyższym rysunku, trzeba nauczyć się sterować wyglądem i położeniem bloków elementów. Pierwszym krokiem w tym kierunku jest poznanie modelu blokowego (rysunek 4.2), który definiuje własności każdego bloku.

RYSUNEK 4.2. Rysunek ten przedstawia relacje między marginesami, obramowaniem i dopełnieniem w elemencie XHTML. W modelu blokowym elementy frontowe, najczęściej tekst i grafika, dodawane są za pomocą kodu XHTML, natomiast kolory i obrazy w tle ustawia się za pomocą CSS

Za pomocą CSS można zmienić trzy cechy bloku:

- **Obramowanie**: można ustawić grubość, styl oraz kolor ramki.
- **Margines**: można ustawić odstęp oddzielający blok od innych sąsiadujących z nim elementów.
- **Dopełnienie**: można ustawić odległość między treścią bloku a jego krawędziami.

Mówiąc prościej, marginesy odpychają wszystko, co znajduje się wokół elementu, a dopełnienie odpycha w stronę środka wszystko, co znajduje się wewnątrz elementu. Ponieważ blok ma cztery krawędzie, własności definiujące margines i dopełnienie mają cztery rodzaje ustawień: `top`, `right`, `bottom` i `left`.

Należy pamiętać, że rzeczywiste grubości reprezentowane przez słowa kluczowe `thin`, `medium` i `thick` nie są zdefiniowane w specyfikacji CSS, przez co mogą występować w tym zakresie różnice między przeglądarkami. Style linii, poza `solid`, także nie są określone w specyfikacji CSS. Na przykład linia przerywana (`dashed`) może w różnych przeglądarkach mieć inne odstępy między kreskami, a i same kreski również mogą się różnić.

Obramowanie

Z własnością `border` są związane trzy inne własności:

- **`width`** (szerokość): przyjmuje wartości `thin`, `medium`, `thick` oraz wszystkie jednostki miary (`em`, `px`, `%` itd.);
- **`style`** (styl): przyjmuje wartości `none`, `hidden`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset` oraz `outset`;
- **`color`**: przyjmuje wszystkie wartości reprezentujące kolory (na przykład w formacie RGB, szesnastkowym i jako słowa kluczowe).

Obramowanie bloku często ustawia się z wszystkich stron na ten sam kolor, styl i grubość. Można każdą z tych własności ustawić osobno, w następujący sposób:

```
p.warning {border-width:4px}
p.warning {border-style:solid}
p.warning {border-color:#F33;}
```

Dodałem też dwa piksele dopełnienia z wszystkich stron, aby tekst nie stykał się z krawędziami.

W takim przypadku lepiej jednak jest użyć skróconego zapisu własności `border`:

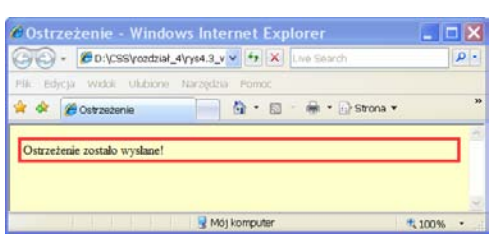
```
p.warning {border:4px solid #F33; padding:2px}
```

Bez względu na to, którego z przedstawionych sposobów użyjemy, każdy akapit należący do klasy `warning` będzie otoczony ciągłą czerwoną ramką o grubości czterech pikseli, która mocno przykuwa uwagę (rysunek 4.3).

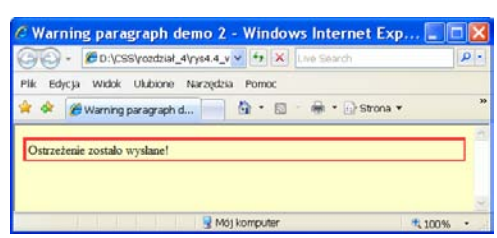
Skrócona własność `border` pozwala zastosować to samo formatowanie do wszystkich czterech krawędzi obramowania elementu. Jeśli jednak chcemy, aby poszczególne krawędzie różniły się, możemy to zrobić z łatwością. Chcemy, aby obramowanie było ciągłą czerwoną linią z wszystkich czterech stron, ale dla stworzenia lepszego efektu wizualnego chcemy też, aby krawędzie prawa i dolna były nieco cieńsze. Do uzyskania tego efektu potrzebne są dwie reguły stylistyczne. Pierwsza będzie ustawiać własności wspólne wszystkim czterem krawędziom, a druga ustawi za pomocą własności `border-width` inną grubość dwóch wybranych krawędzi.

```
p.warning {border:solid #F33; padding:2px;}
p.warning {border-width:4px 2px 2px 4px;}
```

Rezultat przedstawia rysunek 4.4.



RYSUNEK 4.3. Ponieważ wszystkie cztery krawędzie mają być takie same, można użyć skróconego zapisu własności border. Dopełnienie zapobiega stykaniu się tekstu z obramowaniem



RYSUNEK 4.4. Dzięki rozdzieleniu stylów na dwie reguły, krawędzie bloku mogą mieć zarówno wspólne cechy formatowania (kolor i dopełnienie), jak i osobne (grubość linii)

Podczas pracy nad stroną pomocne jest tymczasowe wyświetlenie ramki elementu, dzięki czemu wyraźniej widać efekt formatowania marginesów i dopełnienia. Domyślne ustawienia bloków są następujące: `border-width: medium`, `border-style: none` i `border-color: black`. Ponieważ własność `border-style` jest ustawiona na `none`, obramowania nie widać. Dlatego aby szybko wyświetlić obramowanie akapitu, można napisać następującą regułę:

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

Ustawia ona styl obramowania na linię ciągłą, dzięki czemu ramka staje się widoczna — kolor i grubość są ustawione domyślnie. Należy jednak pamiętać, że dodanie krawędzi może mieć wpływ na układ strony, ponieważ grubość obramowania jest dodawana do rozmiaru całego bloku. Może to mieć znaczenie w zależności od tego, w którym miejscu strony znajduje się element. Innym sposobem na wyświetlenie ramki wokół elementu jest ustawienie koloru tła, dzięki czemu widać powierzchnię bloku. W tym przypadku blok nie zmienia rozmiaru.

Skrócony zapis własności

Wpisywanie za każdym razem wszystkich czterech wartości podczas definiowania marginesów, dopełnienia lub ramki elementu bywa nużące. Dlatego w CSS można używać skróconego zapisu, pozwalającego na zdefiniowanie wszystkich czterech wartości w jednej deklaracji. Kolejność wpisywania wartości w takich deklaracjach jest zawsze następująca: góra, prawa, dół, lewa, czyli zgodna z ruchem wskazówek zegara, zaczynając od godziny 12. W związku z tym, aby ustawić marginesy dla elementu, poniższy zapis:

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

można zastąpić następującym:

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

Poszczególne wartości są oddzielane pojedynczą spacją. Nie stosuje się żadnego znaku rozdzielającego, jak przecinek. Nie ma konieczności definiowania wszystkich czterech wartości. Jeśli któraś wartość zostanie pominięta, przeglądarka użyje wartości z przeciwnej strony.

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

W powyższym przykładzie brakuje ostatniej wartości (lewa krawędź). Dlatego w jej miejscu zostanie użyta wartość zdefiniowana dla prawej strony (`10px`).

W poniższym przykładzie zostały ustawione tylko dwie wartości `top` i `right` (góra i prawa). Zatem wartości `bottom` i `left` (dół i lewa) będą wynosić odpowiednio 12 i 10 pikseli.

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

W końcu, jeśli będzie podana tylko jedna wartość, zostanie ona odniesiona do wszystkich czterech stron:

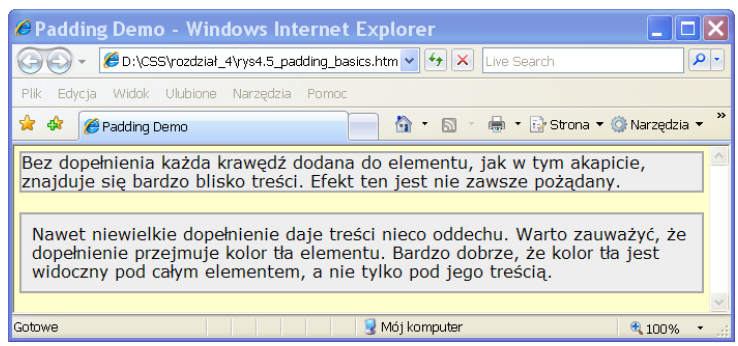
```
{margin:12px;}
```

Za pomocą tej skróconej notacji nie da się tylko zdefiniować dolnej i lewej krawędzi bez podawania wartości dla prawej i górnej, nawet jeśli wynoszą one zero. W takim przypadku można napisać 0 bez żadnej jednostki:

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

Dopełnienie

Dopełnienie to przestrzeń między treścią bloku a jego krawędziami. Ponieważ stanowi wewnętrzną część bloku, przyjmuje jego kolor tła. Rysunek 4.5 przedstawia dwa akapity — jeden z dopełnieniem i jeden bez.



RYSUNEK 4.5. W elementach z widoczną ramką prawie zawsze stosuje się dopełnienie, mające na celu odsunąć treść od ramki

W miejscach, w których kiedyś projektanci musieli stosować dopełnienie tabeli i puste GIF-y, co wymagało dużej ilości dodatkowego kodu prezentacyjnego, dziś wystarczy tylko niewielka ilość kodu CSS.

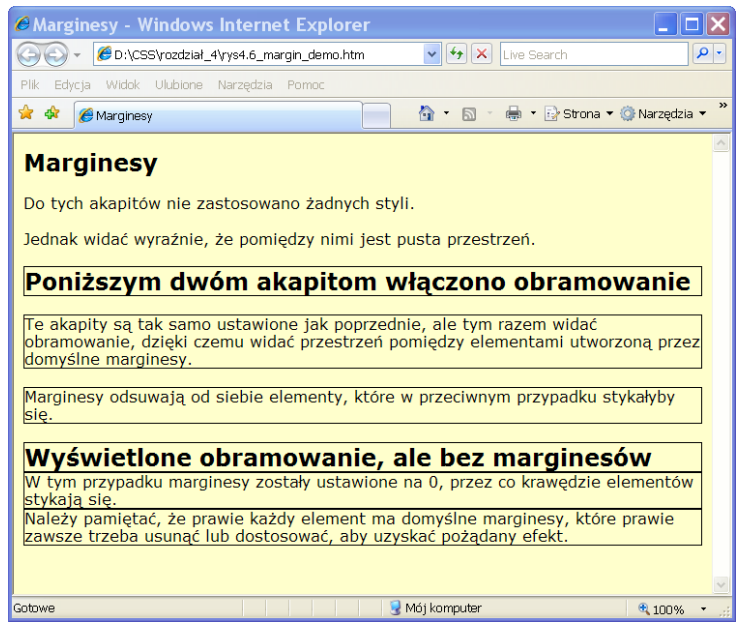
Marginesy

Marginesy są nieco bardziej skomplikowane niż obramowanie i dopełnienie. Po pierwsze, większość elementów blokowych (akapity, nagłówki, listy itp.) mają domyślne marginesy, o czym pisałem już wcześniej.

Rysunek 4.6 przedstawia trzy zestawy złożone z nagłówka i dwóch akapitów. Pierwszy z nich pokazuje domyślne formatowanie przeglądarki. W drugim przypadku zastosowano marginesy i pokolorowano tło, dzięki czemu widać, jak marginesy tworzą pustą przestrzeń. Trzeci przykład pokazuje, co się dzieje, kiedy marginesy zostaną ustawione na zero — elementy się stykają.

Do dobrych zwyczajów należy umieszczenie poniższej deklaracji na początku każdego arkusza stylów:

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



RYSUNEK 4.6. Kontrolowanie marginesów jest kluczową umiejętnością — nie można zapominać, że prawie każdy element ma ustawione marginesy domyślne

Reguła ta ustawia marginesy i dopełnienie **wszystkich** elementów na zero. Dzięki temu nie trzeba pamiętać, które elementy mają te własności ustawione domyślnie, a które nie. Po wstawieniu tej reguły do arkusza stylów wszystkie domyślne marginesy i dopełnienia znikną. Teraz można niniejsze własności określić osobno, tylko dla wybranych elementów. Później dowiemy się, że różne przeglądarki stosują odmienne domyślne wartości marginesów i dopełnienia dla poszczególnych grup elementów, jak formularze czy listy. Dzięki usunięciu domyślnych ustawień i zastosowaniu własnych można uzyskać bardziej jednolity efekt we wszystkich przeglądarkach.

Często w ustawieniach marginesów dla elementów tekstowych stosuje się mieszane jednostki miary. Na przykład lewy i prawy margines akapitu można zdefiniować w pikselach, aby tekst zawsze znajdował się w tej samej odległości od bocznego menu. Natomiast marginesy górny i dolny można ustawić w jednostkach `em`, dzięki czemu odległość między akapitami w pionie będzie zależała od rozmiaru tekstu, na przykład:

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

W tym przypadku odległość między akapitami będzie zawsze równa trzem czwartym wysokości pisma. Jeśli ogólny rozmiar tekstu w znaczniku `body` zostanie zwiększony, zwiększą się nie tylko akapity, ale także proporcjonalnie odstępy między. Marginesy lewy i prawy pozostaną bez zmian, ponieważ zostały zdefiniowane w pikselach. Bliżej przyjrzymy się temu zagadnieniu w rozdziale 5, kiedy zaczniemy projektować układy stron.

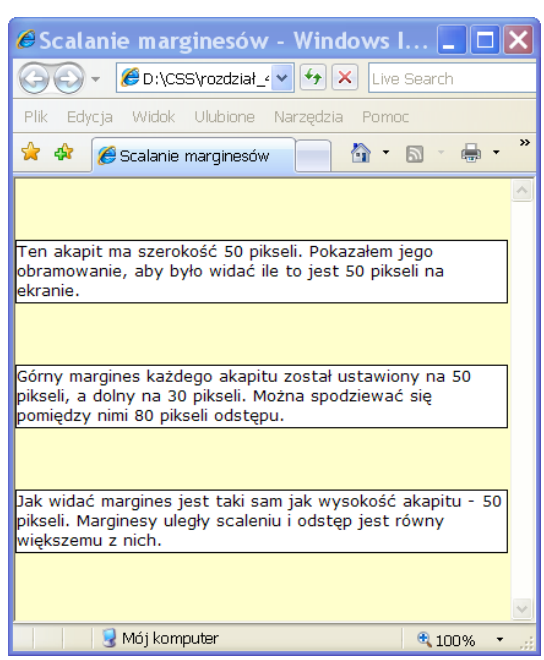
Scalanie marginesów

Powiedz to zdanie na głos: „Marginesy sąsiadujące w pionie są scalane". Koniecznie trzeba o tym pamiętać. Już wyjaśniam, co to znaczy i dlaczego jest takie ważne. Wyobraźmy sobie, że mamy trzy akapity, jeden pod drugim. Każdy z nich formatuje poniższa reguła:

```
p {width:400px; height:50px; border:1px solid #000;
margin-top:50px; margin-bottom:30px; background-color:#CCC;}
```

Wprawdzie marginesy sąsiadujące w pionie scalają się, jednak marginesy poziome nie. Te drugie zachowują się zgodnie z oczekiwaniami, czyli tworzą odstęp równy ich sumie.

Ponieważ dolny margines pierwszego akapitu styka się z górnym marginesem drugiego akapitu, można by się spodziewać, że odstęp między nimi wyniesie 80 pikseli (50+30). Nie jest to jednak prawda. **Kiedy górny i dolny margines stykają się, nachodzą na siebie, aż jeden z nich sięgnie krawędzi drugiego elementu.** W tym przypadku większy jest margines górny drugiego elementu, dzięki czemu właśnie on określa odstęp między elementami — 50 pikseli (rysunek 4.7). Efekt ten nazywa się **scalaniem marginesów** (ang. *margin collapsing*).



RYSUNEK 4.7. Marginesy sąsiadujące w pionie ulegają scaleniu

Efekt scalania marginesów pozwala na utrzymanie pierwszego i ostatniego elementu z grupy takich elementów jak nagłówki, akapity czy listy w odpowiedniej odległości od górnej lub dolnej części strony. Kiedy takie same elementy pojawiają się między innymi elementami, oba marginesy nie są potrzebne i zostają scalone w taki sposób, że odstęp określa większy z nich.

Rozmiary bloku

Zasada działania modelu blokowego należy do najtrudniejszych aspektów CSS zarówno dla ,początkujących jak i zaawansowanych użytkowników. Należy pamiętać, że poniższe informacje dotyczą elementów blokowych, takich jak nagłówki, akapity czy listy. Elementy liniowe zachowują się nieco inaczej.

Przeanalizujemy model blokowy szczegółowo, krok po kroku. Omówimy metody ustawiania szerokości bloków, które mają kluczowe znaczenie w tworzeniu układów kolumnowych. Te same zasady dotyczą także ustawiania wysokości bloku.

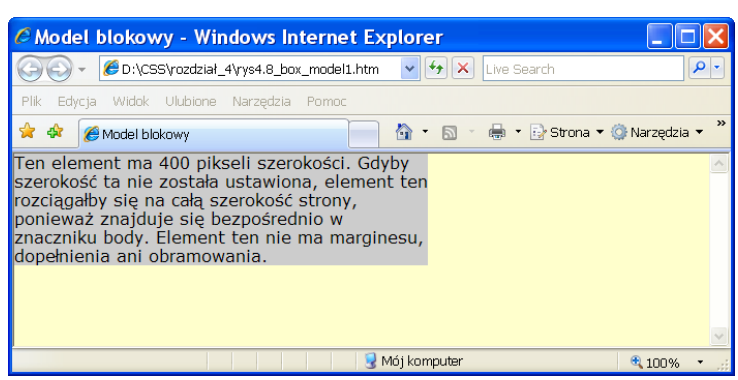
Szerokość elementu blokowego (dalej zwanego blokiem) ustawia własność `width`:

```
p {width:400px;}
```

Aby zobaczyć blok, nie zmieniając ustalonego powyżej rozmiaru, można pokolorować jego tło:

```
p {width:400px; background-color:#EEE;}
```

Rysunek 4.8 przedstawia element o szerokości 400 pikseli z pokolorowanym tłem.



RYSUNEK 4.8. Dzięki ustawieniu własności width element nie zajmuje całego dostępnego miejsca. W tym przypadku element znajduje się w elemencie body, czyli rozciągałby się od lewej do prawej krawędzi okna przeglądarki

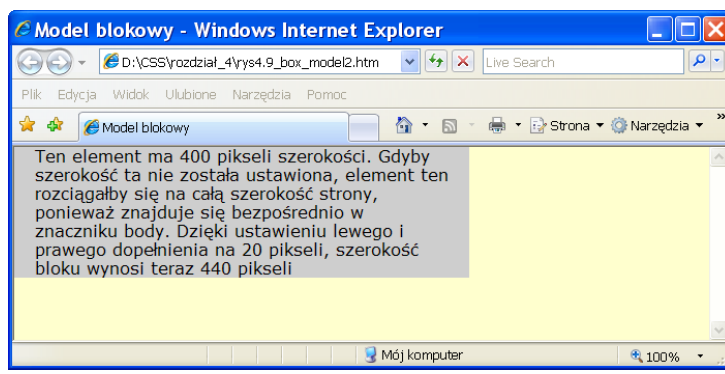
Ponieważ nie ustawiono dopełnienia, treść niniejszego elementu również ma szerokość 400 pikseli i styka się z krawędziami swojego

kontenera. Jest to jak najbardziej zrozumiałe, ale po zastosowaniu dopełnienia i obramowania coś zaczyna szwankować. Po lewej i prawej stronie elementu ustawimy 20-pikselowe dopełnienie:

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

Można by się spodziewać, że po zastosowaniu 40 pikselowego dopełnienia w bloku o szerokości 400 pikseli na treść pozostanie 360 pikseli. Jest jednak inaczej. W zamian cały blok zostaje powiększony o 40 pikseli (rysunek 4.9).

RYSUNEK 4.9. Dopełnienie rozszerza blok

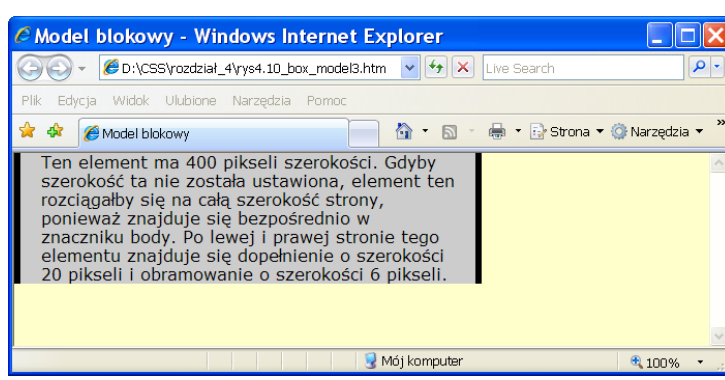


Jeśli po lewej i prawej stronie elementu ustawimy obramowanie o grubości 6 pikseli:

```
p {width:400px; margin: 0; padding:0 20px; border:#000 solid;
border-width: 0 6px 0 6px; background-color:#CCC;}
```

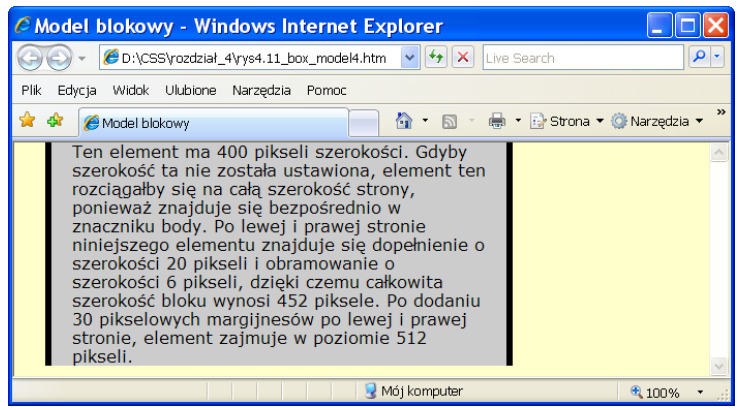
blok poszerzy się o kolejne 12 pikseli (rysunek 4.10). Teraz całkowita szerokość bloku wynosi 452 piksele (6+20+400+20+6).

RYSUNEK 4.10. Obramowanie rozszerza blok jeszcze bardziej



Dodamy jeszcze lewy i prawy margines, aby utworzyć pustą przestrzeń wokół elementu (rysunek 4.11):

```
p {width:400px; margin: 0 30px; padding:0 20px; border:#000
solid; border-width: 0 6px 0 6px; background-color:#CCC;}
```



RYSUNEK 4.11. Marginesy tworzą wokół elementu pustą przestrzeń

Dodanie marginesów, w tym przypadku po 30 pikseli z lewej i prawej strony, zwiększa zajmowaną przez element powierzchnię, ponieważ znajdują się one na zewnątrz bloku. Mimo iż można by oczekwać, że obramowanie i dopełnienie, które znajdują się wewnątrz bloku, nie zwiększają jego szerokości, prawda jest inna.

Uwaga nr 1 na temat modelu blokowego: Bloki o zdefiniowanych rozmiarach rozszerzają się w poziomie w wyniku dodawania do nich marginesów, dopełnienia i obramowania. Własność `width` *ustawia szerokość treści bloku, a nie szerokość samego bloku.*

Może to mieć duże znaczenie w przypadku układów kolumnowych, w których kolumny muszą mieć określoną szerokość. Układ złożony z elementów pływających (sposoby tworzenia takich układów opiszę w kolejnym rozdziale) może zostać zburzony, jeśli szerokość kolumny zostanie nieuważnie zmieniona przez zmianę szerokości dopełnienia, marginesu lub obramowania. Kolumnę zazwyczaj tworzy się za pomocą elementu `div` o odpowiednich rozmiarach. W elemencie tym następnie zagnieżdżane są elementy treści kolumny, jak nagłówki, akapity, listy nawigacyjne itd.

Tworzenie pojedynczej kolumny

Jako ilustrację podstaw techniki tworzenia układów kolumnowych przedstawiam poniższy element `div` o szerokości 170 pikseli, zawierający nagłówek i akapit:

```
<div id="column">
    <h4>Nagłówek h4</h4>
    <p>Nagłówek ten i akapit...</p>
 </div>
```

Reguła CSS dla tego elementu `div`:

```
div#column {width:170px;}
```

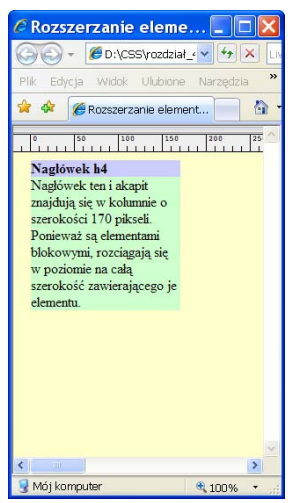
Wzdłuż górnej krawędzi okna dodałem linijkę, aby było widać, jak zmienia się szerokość wraz ze zmianami wprowadzanymi w CSS (rysunek 4.12).

Uwaga nr 2 na temat modelu blokowego: Elementy, które nie mają zdefiniowanej szerokości, rozciągają się na całą szerokość zawierającego je elementu. Dlatego dodanie poziomego marginesu, dopełnienia i obramowania powoduje zmianę szerokości treści.

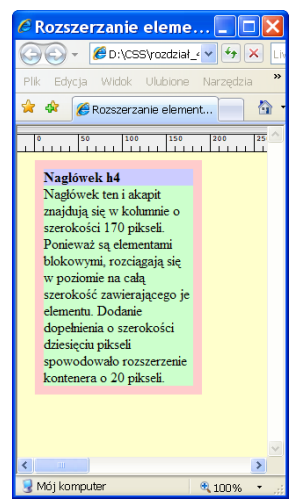
Pokolorowałem tło nagłówka i akapitu, aby pokazać, że całkowicie wypełniają kolumnę w poziomie. Domyślny rozmiar elementów blokowych to `auto`, co oznacza: najwięcej, jak to możliwe. To prowadzi do sformułowania drugiej uwagi.

Widząc tekst stykający się z krawędziami kontenera, jak na powyższym rysunku, powinniśmy natychmiast poczuć chęć ustawienia dopełnienia dla elementu `div`, aby dodać nieco przestrzeni (rysunek 4.13).

```
div#column {width:170px; padding:10px;}
```



RYSUNEK 4.12. Jeśli kontener nie ma ustawionego dopełnienia, elementy blokowe — nagłówek i akapit — rozciągają się od lewej do prawej krawędzi zawierającego je elementu



RYSUNEK 4.13. Dodanie dopełnienia do kontenera powoduje jego rozszerzenie. Teraz kontener ma 192 piksele szerokości. Dzięki pokolorowaniu tła elementu div na różowo wyraźniej widać dopełnienie

Jak widać na linijce u góry, dopełnienie o szerokości 10 pikseli spowodowało rozszerzenie elementu do szerokości 190 pikseli. Dzięki temu treść elementu została odsunięta od krawędzi, jednak aby zachować jego początkową szerokość 170 pikseli, musimy odpowiednio zmniejszyć wartość własności `width` elementu `div`. Ustawimy ją na 150 pikseli. Takie modyfikowanie szerokości kolumn po każdej zmianie szerokości dopełnienia bywa męczące. Zwłaszcza kiedy mamy kilka kolumn.

Ewentualnie można zastosować jednakowe marginesy do wszystkich elementów w kolumnie, ale to oznaczałoby konieczność dostosowania wielu elementów, gdybyśmy zdecydowali się zmienić odległość kolumny od treści.

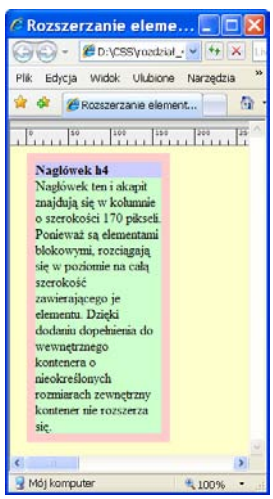
Proste rozwiązanie polega na dodaniu jeszcze jednego elementu `div` wewnątrz elementu `div` tworzącego kolumnę.

```
<div id="column">
  <div id="column_inner">
    <h4>Nagłówek h4</h4>
    <p>Nagłówek ten i akapit...</p>
  </div>
</div>
```

Dopełnienie ustawiamy dla wewnętrznego elementu `div`:

```
div#column {width:170px; padding:10px;}
div#inner_column {padding:10px;}
```

Dzięki temu można kontrolować dopełnienie kolumny za pomocą jednego stylu, unikając przy tym problemów ze zmianą szerokości kolumny (rysunek 4.14).



RYSUNEK 4.14. Dzięki zastosowaniu dopełnienia dla wewnętrznego elementu div szerokość kolumny zdefiniowanej przez zewnętrzny element div nie ulega zmianie

Wewnętrzny element `div` nie ma określonego rozmiaru, a więc uwaga nr 2 na temat modelu blokowego jest prawdziwa. Treść została ściśnięta. Teraz zmieniając jedno ustawienie marginesu, możemy odsunąć wszystkie elementy kolumny od jej krawędzi. Rozmiar kolumny pozostanie bez zmian. Techniki dwóch elementów `div` będę używał w wielu układach stron prezentowanych w kolejnym rozdziale. Dlatego przed przejściem do dalszej części należy dobrze zrozumieć jej ideę.

Wniosek z dotychczasowych rozważań jest następujący: we wszystkich zgodnych ze standardami przeglądarkach własność CSS `width` ustawia nie szerokość elementu, a znajdującej się w jego wnętrzu treści. Dopełnienie, marginesy i obramowanie są dodawane do całkowitego rozmiaru elementu, który ma zdefiniowaną szerokość.

Teraz przyjrzymy się dwóm pozostałym kluczowym technikom w zakresie projektowania układów stron za pomocą CSS. Są to elementy pływające (własność `float`) i własność `clear`.

Po wszystkich moich wywodach na temat zalet unikania prezentacyjnego kodu XHTML może wydać się dziwne, że byłem skłonny użyć dodatkowego znacznika, aby uzyskać efekt wizualny. Jednak elementy div, w przeciwieństwie do tabel, nie zmieniają w żaden sposób efektu wizualnego, chyba że zostaną celowo sformatowane. Dlatego wydaje mi się, że jest to wart uwagi kompromis, zwłaszcza dla początkujących użytkowników, którzy i tak muszą pamiętać o wielu innych rzeczach. Dzięki temu nie trzeba przy każdej zmianie dopełnienia lub marginesu wyciągać kalkulatora i na nowo obliczać szerokości kolumn.

Elementy pływające i ich czyszczenie

Kolejną niezwykle przydatną techniką pomagającą w organizacji układu strony jest użycie elementów pływających w połączeniu z własnością `clear`. Element pływający pozostaje poza standardowym układem elementów na stronie. Elementy znajdujące się za elementem pływającym przesuwają się do góry, układając się obok niego, jeśli jest tam wystarczająco dużo miejsca. Własność `clear` pozwala określić, czy elementy znajdujące się za elementem pływającym mają przesuwać się do góry czy nie. Jeśli na przykład mamy dwa akapity i chcemy, aby tylko pierwszy z nich pojawił się obok elementu pływającego, możemy ten drugi zatrzymać pod elementem pływającym za pomocą własności `clear`. Przyjrzymy się uważniej obu tym własnościom.

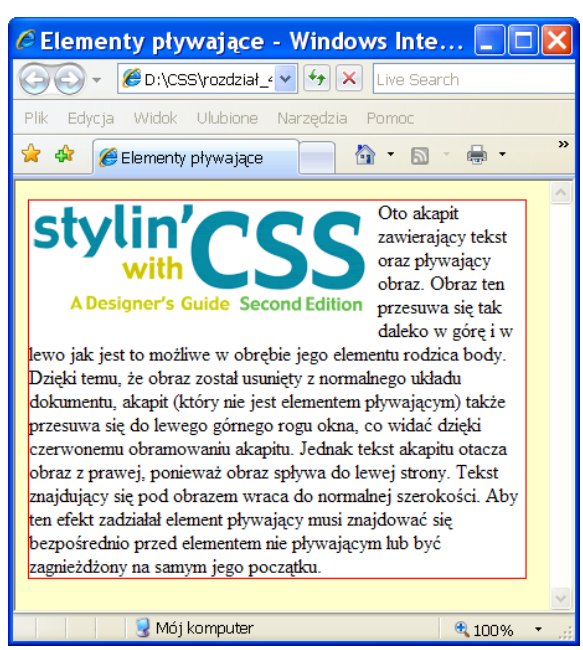
Własność float

Jednym z zastosowań własności `float` jest otaczanie obrazów tekstem. Należy ona jednak także do fundamentów układów wielokolumnowych.

Zaczniemy od otaczania obrazów tekstem.

```
img {float:left; margin:0 4px 4px 0;}
```

Powyższa reguła spycha obraz na lewo, dzięki czemu tekst będzie znajdował się z jego prawej strony (rysunek 4.15).



RYSUNEK 4.15. Obraz pływający znajduje się poza standardowym układem strony. Jeśli za nim znajduje się element tekstowy, tworzący go tekst będzie otaczał ten obraz

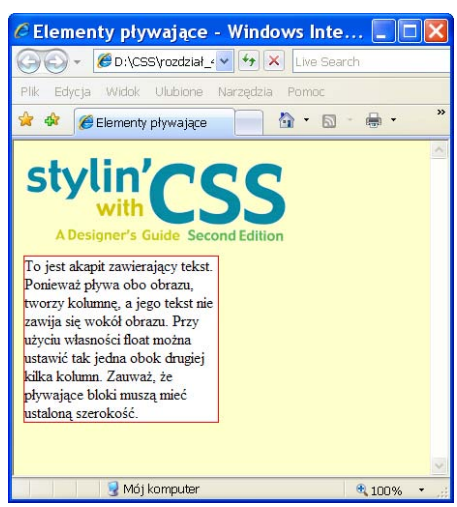
Aby własność `float` zadziałała poprawnie, kod XHTML musi wyglądać następująco:

```
<img .../>

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

Innymi słowy, używając własności `float`, żądamy, aby element został przesunięty jak najdalej w lewo (lub prawo w przypadku deklaracji `float: right`) w obrębie zawierającego go elementu — w tym przypadku `body`. Akapit (na powyższym rysunku ma czerwoną ramkę) nie traktuje elementu pływającego jako bloku znajdującego się przed nim i dlatego również przesuwa się do lewego górnego rogu swojego rodzica. Jednak jego treść (tekst) zawija się wokół pływającego obrazu.

Od tego już tylko krok do tworzenia pływających kolumn (rysunek 4.16).

```
p {float:left; width:200px; margin:0;}

img {float:left; margin:0 4px 4px 0;}
```



RYSUNEK 4.16. Kiedy akapit o stałej szerokości zostaje ustawiony za pomocą własności float obok pływającego obrazu, tworzy kolumnę, a jego tekst nie otacza już obrazu

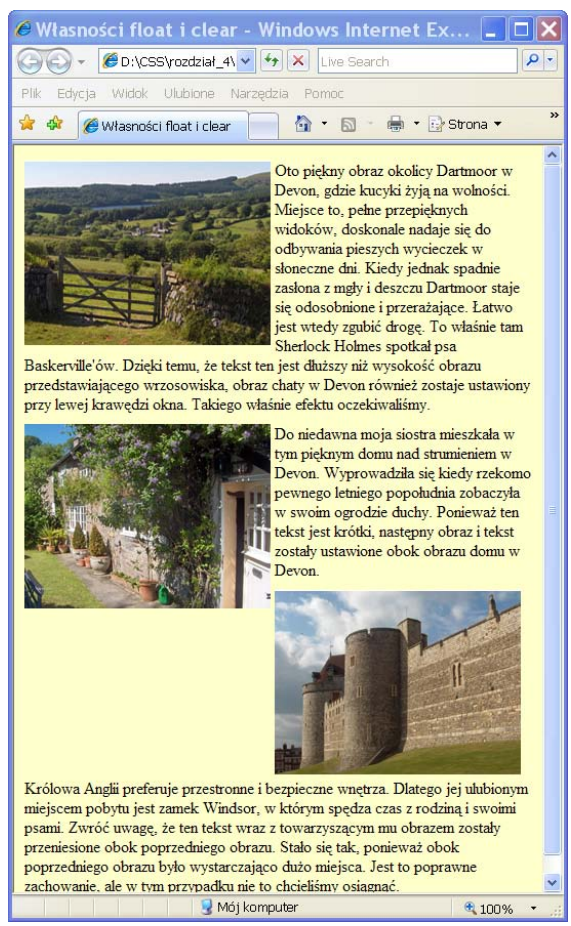
Zasad dotyczących elementów pływających jest wiele więcej. Można o nich przeczytać w książce Erica Meyera pod tytułem Cascading Style Sheets 2.0 Programmer's Reference (McGraw-Hill Osborne Media, 2001). Cytując za tym autorem, „Kiedy do elementu zastosuje się własność `float`*, zasady te powodują wyniesienie tego elementu tak daleko w lewo i do góry, jak to tylko możliwe". Mimo iż od wydania tej książki minęło kilka lat, jest ona doskonałym źródłem wiedzy na temat wewnętrznych mechanizmów rządzących CSS, której znalezienie gdzie indziej graniczy z cudem.*

Zastosowanie własności `float` zarówno do obrazu, jak i akapitu (o ustalonych szerokościach) powoduje, że tekst przestaje owijać się wokół obrazu. Jest to jedna z głównych zasad tworzenia układów kolumnowych przy użyciu elementów pływających. Elementy ustawiają się obok siebie jak kolumny, jeśli mają ustaloną szerokość i jest dla nich wystarczająco miejsca (obrazy zawsze mają określony rozmiar i nie trzeba ustawiać im rozmiaru w CSS, aby pływały). Jeśli ustalimy szerokość trzech elementów `div` i zastosujemy do nich własność `float`, otrzymamy trzy kontenery, do których można wstawiać inne elementy (które także mogą pływać). Praktyczne przykłady opisywanych technik przedstawione zostały w rozdziale 5.

Własność clear

Z własnością `float` zazwyczaj współwystępuje własność `clear`. Jeśli jeden element jest pływający, inny — jeśli starczy dla niego miejsca — ustawi się obok niego. Czasami jednak nie chcemy, aby tak się stało. Wolimy, aby ten drugi element pozostał pod elementem pływającym. Rysunek 4.17 przedstawia stronę zbudowaną z elementów złożonych z obrazu i tekstu otaczającego go z prawej strony. Efekt ten uzyskano dzięki zastosowaniu pływających obrazów. Jest to identyczna sytuacja, jak na rysunku 4.16, tylko powtórzona trzy razy.

RYSUNEK 4.17. Dzięki temu, że obok drugiego obrazu jest miejsce, trzeci obraz pływa obok niego — efekt ten nie był zamierzony



Oto kod XHTML powyższej strony (nieco skrócony dla oszczędzenia miejsca)

```
<img src="../images/dartmoor-view.jpg" />
<p>Oto piękny obraz okolicy Dartmoor... </p>
<img src="../images/english-cottage.jpg" />
```

```
<p>Do niedawna moja siostra mieszkała w tym pięknym domu...</p>
<img src="../images/windsor-castle_walls.jpg" />
<p>Królowa Anglii...</p>
```

Kod CSS:

```
p {margin:0 0 10px 0;}
img {float:left; margin:0 4px 4px 0;}
```

Każdy obraz powinien pływać po lewej stronie opisującego go tekstu. Jeśli jednak tekst jest zbyt krótki i nie zajmuje całego miejsca dostępnego po prawej stronie obrazu, jak w przypadku drugiego akapitu na rysunku 4.17, następna para obraz – tekst zostanie podciągnięta do góry, w to wolne miejsce.

Układ ten jest poprawnie interpretowany przez przeglądarkę. Trzeci element ma wystarczająco miejsca, aby przenieść się obok drugiego, więc to robi. **Nie** jest to oczywiście zamierzony przez nasz efekt wizualny. Rozwiązaniem w tym przypadku jest dodanie niepływającego elementu do kodu XHTML i ustawienie jego własności `clear` w celu zatrzymania ostatniego elementu na dole. Poniżej znajduje się potrzebny kod XHTML z dodanym elementem `div` oraz odpowiedni kod CSS:

```
<img src="../images/dartmoor-view.jpg" />
<p>Oto piękny obraz okolicy Dartmoor... </p>
<img src="../images/english-cottage.jpg" />
<p>Do niedawna moja siostra mieszkała w tym pięknym domu...</p>
<div class="clearthefloats"></div>
<img src="../images/windsor-castle_walls.jpg" />
<p>Królowa Anglii...</p>
```

W kodzie CSS musimy tylko dodać klasę `clearthefloats`:

```
p {margin:0 0 10px 0;}
img {float:left; margin:0 4px 4px 0;}
.clearthefloats {clear:both;}
```

Po wstawieniu dodatkowego elementu XHTML i klasy czyszczącej w CSS (która czyści elementy pływające z obu stron), strona wygląda tak, jak oczekiwaliśmy (rysunek 4.18).

RYSUNEK 4.18. Dzięki zastosowaniu dodatkowego elementu układ strony prezentuje się bez zarzutów

Nowy element dodany między drugim a trzecim akapitem znajduje się teraz (mimo że go nie widać, ponieważ nie zawiera żadnej treści) pod drugim obrazem. Dzięki temu, że trzeci obraz i akapit znajdują się za tym elementem w kodzie XHTML, są ustawione pod nim i pożądany efekt został osiągnięty.

Wartość `both` *własności* `clear` *powoduje, że element zostaje przeniesiony pod elementy pływające i przesunięty zarówno do prawej, jak i do lewej. Można było w tym przypadku zastosować wartość* `left`*, ale dzięki wartości* `both` *strona będzie dobrze wyglądać także wtedy, gdy obrazy będą pływać do prawej.*

Prawidłowe posługiwanie się własnością `clear` jest bardzo ważną umiejętnością przy tworzeniu układów kolumnowych. W ramce „Metoda czyszczenia Asletta" opisałem technikę czyszczenia elementów pływających przy użyciu tylko kodu CSS i jednej klasy w kodzie XHTML. Elementami pływającymi i własnością `clear` zajmiemy się później. Zaprezentowane do tej pory informacje wystarczą początkującemu twórcy układów stron opartych na elementach pływających. Teraz przejdziemy do własności `position`.

Metoda czyszczenia Asletta

Nazwa tej metody pochodzi od jej twórcy, Tony'ego Asletta (*www.csscreator.com*). Pozwala ona zmusić kontener, na przykład `div`, aby otaczał zagnieżdżoną w nim pływającą treść, czego normalnie nie robi. Technika ta wykorzystuje pseudoelement CSS `:after` do wstawiania ukrytego fragmentu niepływającej treści (kropki z wysokością równą zero) za pozostałą treścią znajdującą się w kontenerze. Dodatkowo do tej wstawianej treści zastosowana jest własność `clear`, przez co kontener musi ją otaczać. Oto cały kod:

```
.clearfix:after {
  content: ".";
  display: block;
  height: 0;
  clear: both;
  visibility: hidden;
}
.clearfix {display: inline-table;}
/* Lewy ukośnik ukrywa kod przed przeglądarką IE dla komputerów Mac \*/
* html .clearfix {height: 1%;}
.clearfix {display: block;}
/* koniec hacka */
```

Można ten kod dodać na końcu arkusza stylów, aby był dostępny na każdej stronie (wstawiłem go na końcu pliku *text_n_colors.css*, który znajdziesz na FTP). Od tej pory, aby utworzyć kontener zamykający w sobie pływającą treść, wystarczy zastosować do niego klasę `clearfix`, na przykład:

```
<div class="clearfix">
```

Niektóre z zastosowań tej techniki:

1. Utrzymanie stopki pod pływającymi kolumnami (układy złożone z pływających kolumn opisuję w rozdziale 5). Klasę `clearfix` należy zastosować do elementu `div` zawierającego kolumny, dzięki czemu kontener ten zawsze będzie się rozszerzał w pionie na długość wystarczającą do pomieszczenia wszystkich kolumn. Element `div` reprezentujący stopkę znajdujący się za elementem zawierającym kolumny będzie dzięki temu znajdował się zawsze pod najdłuższą kolumną.
2. Dodanie obramowania wokół kilku pływających elementów. Elementy pływające należy umieścić w jednym kontenerze `div` i zastosować do tego kontenera klasę `clearfix`. Dzięki temu elementy pływające pozostaną w kontenerze i wystarczy nadać odpowiedni styl jego ramce.

Jest to jedna z tych technik, o których nie wiadomo, do czego mogą się przydać, dopóki się ich nie potrzebuje. Z pewnością jest to lepsze rozwiązanie niż wstawianie dodatkowych elementów `div` do źródła strony.

Należy pamiętać, że w przeglądarce IE 6 elementy pływające są niesłusznie zamykane w elementach `div`. Jest to jeszcze jeden powód, aby najpierw sprawdzać stronę w przeglądarkach zgodnych ze standardami, a dopiero potem testować w IE 6.

Więcej informacji na temat opisywanej metody można znaleźć pod adresem *www.positioniseverything.net/easycle*
↪aring.html.

Własność position

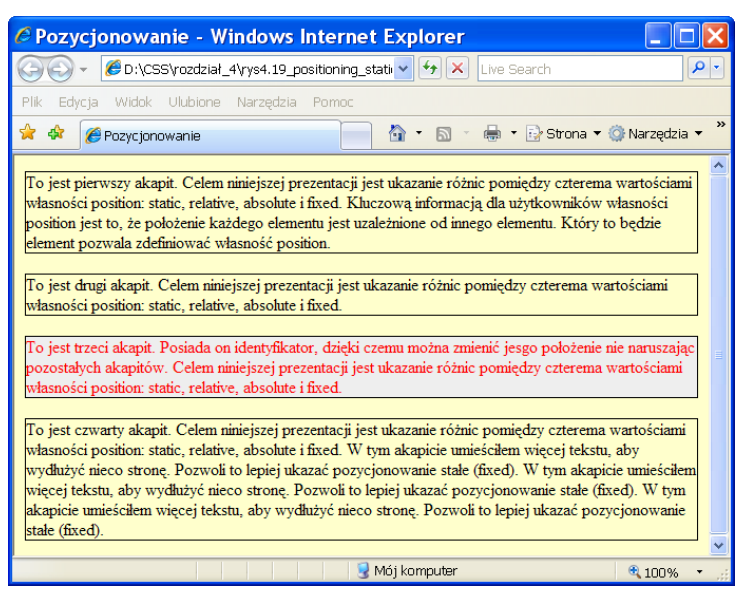
Sercem układu kolumnowego w CSS jest własność `position`. Pozwala ona zdefiniować punkt odniesienia, względem którego element ma być pozycjonowany na stronie.

Zobaczmy, co to oznacza.

Własność `position` może przyjmować jedną z czterech wartości: `static`, `absolute`, `fixed` i `relative`. Domyślna jest pierwsza z wymienionych. Miałem nieco problemów ze zrozumieniem działania tych wartości, kiedy po raz pierwszy natknąłem się na nie. Aby pomóc czytelnikowi uniknąć takich samych rozterek jak moje, przyjrzymy się tym czterem wartościom na czterech przykładach dokumentu złożonego z czterech akapitów. Zmianie będzie ulegać tylko wartość własności `position` trzeciego akapitu. Pozostałe akapity cały czas będą miały tę własność ustawioną na wartość domyślną. Aby ułatwić sobie zadanie, trzeciemu akapitowi nadałem identyfikator o nazwie `specialpara`.

Pozycjonowanie statyczne

Najpierw przyjrzymy się naszym akapitom z własnością `position` ustawioną na `static` (rysunek 4.19).



RYSUNEK 4.19. Jeśli wszystkie cztery akapity mają zdefiniowaną własność position jako static, układają się jeden pod drugim zgodnie ze standardowym rozkładem elementów na stronie

Pozycjonowanie statyczne polega na ułożeniu elementów jeden pod drugim. Odległość między nimi jest równa ich domyślnym marginesom.

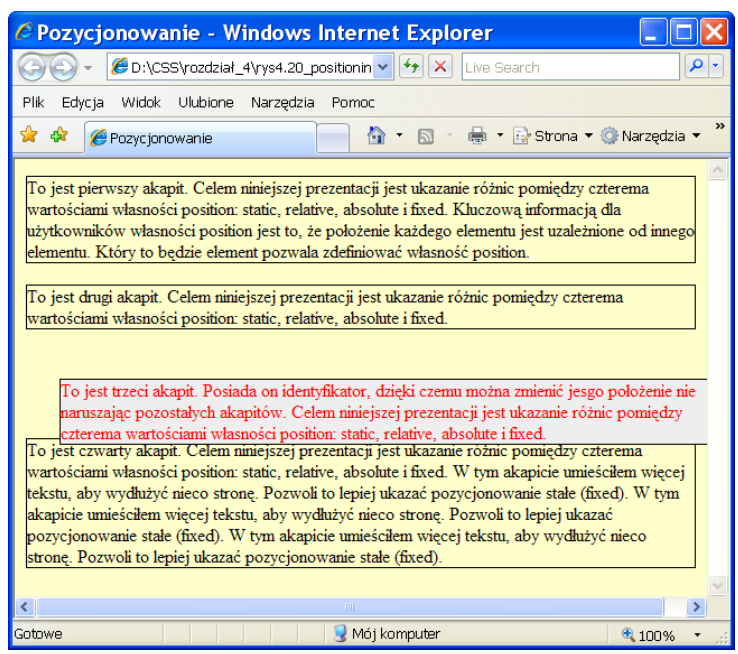
Aby zmienić ten sekwencyjny układ elementów pozycjonowania statycznego na stronie, trzeba użyć jednej z pozostałych trzech wartości własności `position`.

Pozycjonowanie względne

Ustawiamy własność `position` trzeciego akapitu na wartość `relative`. Dzięki temu możemy przesuwać ten element względem jego domyślnego położenia za pomocą własności `top`, `right`, `bottom` i `left`. Zazwyczaj wystarczą tylko wartości `top` i `left`. Poniższa reguła stylistyczna:

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

wywoła efekt widoczny na rysunku 4.20.



RYSUNEK 4.20. Element pozycjonowany względnie można przesuwać w stosunku do jego domyślnego położenia za pomocą własności `top`, `right`, `bottom` i `left`

Lewy górny róg akapitu został przesunięty o 30 pikseli w dół i 20 pikseli w prawo. Jak widać, mimo że trzeci akapit został przesunięty, nic więcej na stronie się nie zmieniło. Miejsce zajmowane przez ten akapit, kiedy był pozycjonowany statycznie, nie zostało zwolnione. Podobnie pozostałe elementy — nadal są na swoich pierwotnych miejscach.

Aby przesunąć element do góry i w lewo, należy użyć wartości ujemnych.

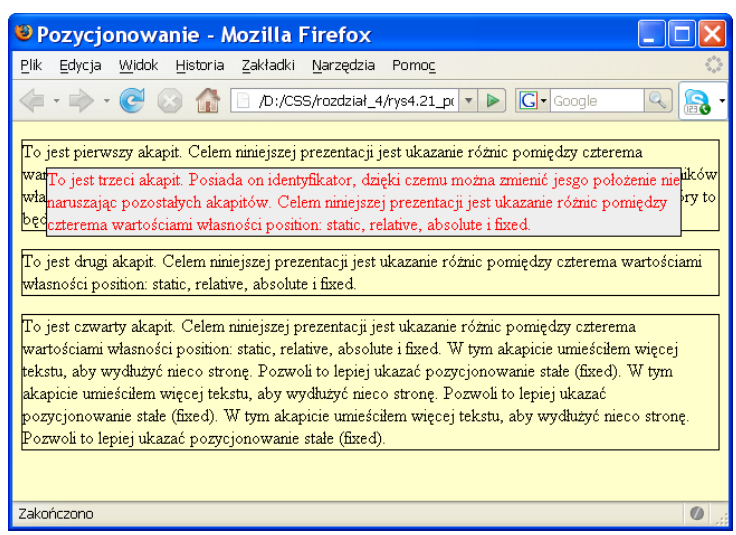
Należy pamiętać, że przesuwając element w ten sposób, trzeba wcześniej wygospodarować dla niego miejsce. W dokumencie zaprezentowanym na rysunku 4.19 należało przesunąć czwarty akapit w dół o 30 pikseli lub więcej. To pozwoliłoby uniknąć nałożenia się na niego trzeciego akapitu po zmianie jego położenia.

Pozycjonowanie bezwzględne

Pozycjonowanie bezwzględne to zupełnie inna bajka niż pozycjonowanie statyczne i względne. Pozwala całkowicie wytrącić element z normalnego układu dokumentu. Zmodyfikujemy kod z poprzedniego przykładu, zmieniając tylko wartość własności `position` z `relative` na `absolute`.

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

Rezultat tego działania przedstawia rysunek 4.21.



RYSUNEK 4.21. Pozycjonowanie bezwzględne polega na całkowitym wytrąceniu elementu z normalnego układu strony i położeniu go względem innego elementu — w tym przypadku elementu body

Jak widać na rysunku 4.21, miejsce wcześniej zajmowane przez trzeci akapit zostało zajęte przez kolejny. Element pozycjonowany bezwzględnie jest całkowicie niezależny od innych otaczających go w kodzie XHTML elementów. Jego położenie jest obliczane względem elementu najwyższego poziomu, czyli `body`. W ten sposób dobrnęliśmy do etapu, kiedy musimy zapoznać się z pojęciem **kontekstu pozycjonowania**, które będzie jeszcze wielokrotnie powracać w tym rozdziale.

Zacznijmy od tego, że ***domyślnym* kontekstem pozycjonowania elementu pozycjonowanego bezwzględnie jest element `body`**. Jak widać na rysunku 4.21, własności `top` i `left` przesunęły aka-

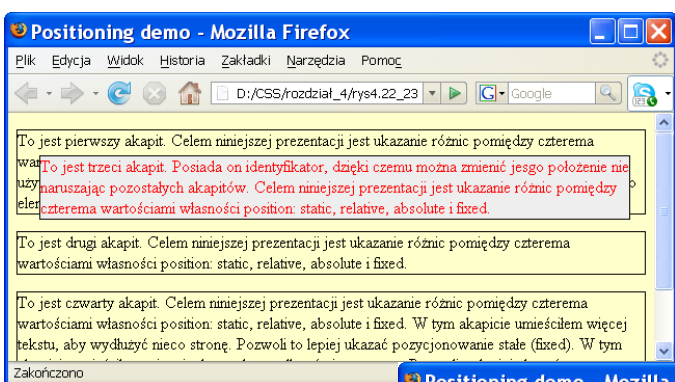
pit względem elementu `body` — elementu najwyższego poziomu w naszym dokumencie, zamiast względem jego domyślnej pozycji w dokumencie.

Ponieważ kontekstem pozycjonowania elementu pozycjonowanego bezwzględnie jest element `body`, pozycjonowany element przesuwa się w miarę przewijania strony, aby cały czas pozostać w tym samym miejscu. Element `body` także przesuwa się w miarę przewijania strony.

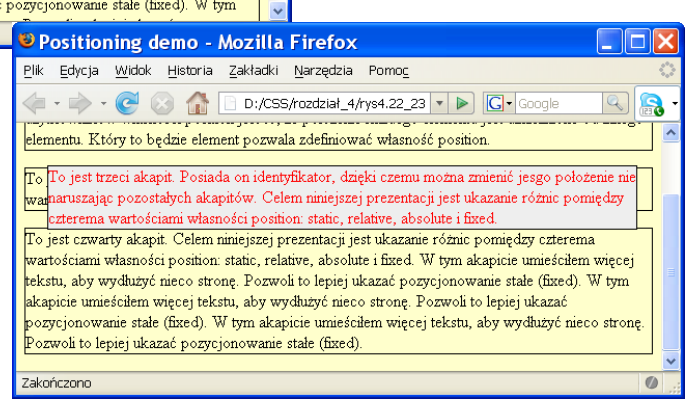
Zanim nauczymy się pozycjonować bezwzględnie elementy w odniesieniu do innych niż `body` elementów, przyjrzymy się ostatniej z wartości własności `position`.

Pozycjonowanie stałe

Pozycjonowanie stałe (`fixed`) jest podobne do bezwzględnego. Różnica polega na tym, że element jest pozycjonowany w odniesieniu do okna przeglądarki. Dzięki temu dany element nie przesuwa się w miarę przewijania strony. Rysunki 4.22 i 4.23 przedstawiają efekt pozycjonowania stałego.



RYSUNEK 4.22. Pozycjonowanie stałe jest podobne do pozycjonowania bezwzględnego



RYSUNEK 4.23. Różnica między pozycjonowaniem stałym a bezwzględnym staje się widoczna po przewinięciu strony

Dzięki „przykuciu" elementu do okna przeglądarki można symulować odradzane ramki. Można na przykład utworzyć element nawigacyjny, który będzie cały czas w tym samym miejscu, unikając jednocześnie problemów z wieloma dokumentami składającymi się na jedną stronę. Pozycjonowanie stałe nie działa jednak w przeglądarce IE 6, ale działa w IE 7. Elegancki sposób obejścia problemu z pozycjonowaniem stałym w przeglądarce IE 6 został opisany na stronie *http://tagsoup.com/cookbook/css/fixed/*.

Kontekst pozycjonowania

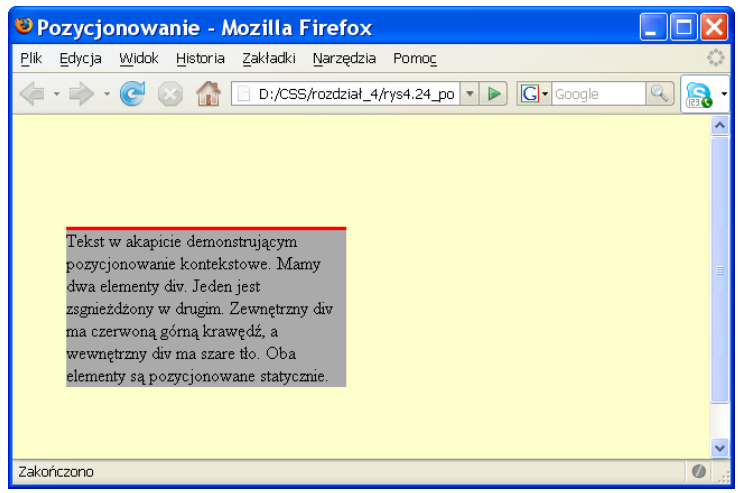
Ponieważ kontekst pozycjonowania jest bardzo ważny dla tych, którzy chcą zrezygnować z tabel jako szkieletu strony, omówimy to zagadnienie szerzej. Zasada jest prosta — **pozycjonowanie kontekstowe** polega na przesuwaniu danego elementu względem innego za pomocą własności `top`, `right`, `left` i `bottom`. Ten drugi element jest właśnie **kontekstem pozycjonowania**. Jak wiemy z podrozdziału „Pozycjonowanie bezwzględne", kontekstem pozycjonowania elementów pozycjonowanych bezwzględnie jest element `body` — chyba że go zmienimy. Na przykład element `body` zawiera wszystkie inne elementy strony. Kontekstem pozycjonowania może być jednak dowolny element będący przodkiem innego elementu, jeśli jego własności `position` nada się wartość `relative`.

Przyjrzyjmy się poniższemu kodowi:

```
<body>
<div id="outer">Zewnętrzny element div
<div id="inner">Trochę tekstu...</div>
</div>
</body>
```

Kod CSS:

```
div#outer_div {width:250px; margin:50px 40px; border-top:3px
solid red;}
div#inner_div {top:10px; left:20px; background:#AAA;}
```



RYSUNEK 4.24. Dwa zagnieżdżone akapity. Górna krawędź zewnętrznego elementu div jest czerwona, a wewnętrzny element div jest szary. Ponieważ wewnętrzny div jest statyczny, własności top i left są ignorowane

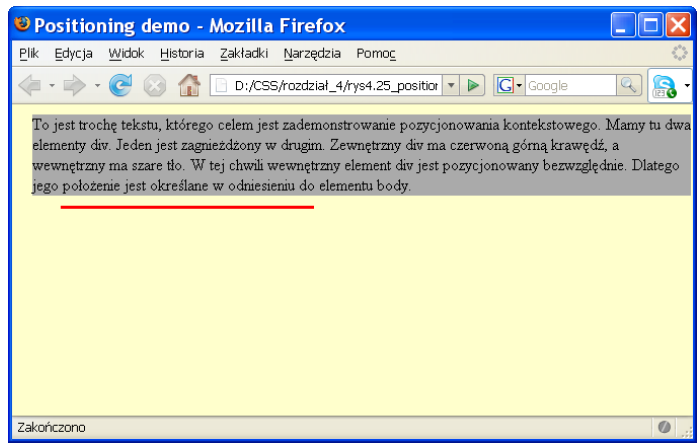
Aby spełnić wymagania dotyczące składni dokumentów, tekst powinien znajdować się w znaczniku `p`. *Dla uproszczenia jednak użyłem znacznika* `div`.

Po dokładniejszym przeanalizowaniu tego kodu nasuwa się pytanie: dlaczego wewnętrzny element `div` nie jest odsunięty o 10 pikseli w dół i 20 pikseli w lewo względem zewnętrznego elementu `div`? Zamiast tego górne lewe rogi obu tych elementów znajdują się w tym samym punkcie. Powodem jest pozycjonowanie statyczne obu elementów. Oznacza to, że elementy te wpasowują się w domyślny rozkład elementów na stronie. Ponieważ zewnętrzny element nie ma żadnej treści, element wewnętrzny zaczyna się w tym samym miejscu. Własności `top`, `right`, `bottom` i `left` działają tylko, jeśli element jest pozycjonowany względnie, bezwzględnie lub w sposób stały. Zobaczmy, co się stanie, kiedy zmienimy pozycjonowanie na bezwzględne (`absolute`).

```
div#outer_div {width:250px; margin:50px 40px; border-top:3px
solid red;}

div#inner_div {position:absolute; top:10px; left:20px;
background:#AAA;}
```

W odniesieniu do czego jest pozycjonowany ten element? Ponieważ nie ma żadnego innego elementu pozycjonowanego względnie, element ten będzie pozycjonowany bezwzględnie w odniesieniu do elementu `body`.

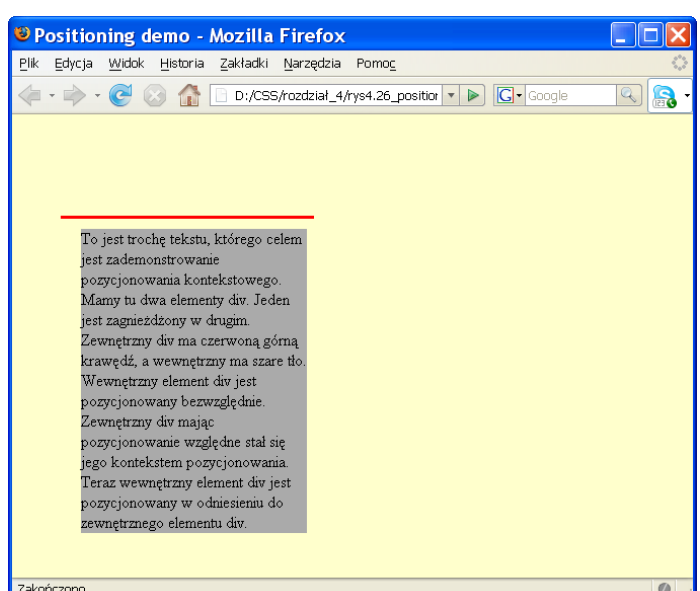
Górna krawędź zewnętrznego elementu `div` jest czerwona, dzięki czemu widać, gdzie się znajduje. Marginesy odsuwają ją 50 pikseli w dół i 40 pikseli w prawo względem górnego rogu okna przeglądarki. Ponieważ wewnętrzny element `div` jest pozycjonowany bezwzględnie, jego położenie jest określane w odniesieniu do elementu `body`, **ponieważ element `body` jest domyślnym kontekstem pozycjonowania**. Element ten ignoruje swojego rodzica (zewnętrzny element `div`) i ustawia się zgodnie z wartościami własności `top` i `left` względem elementu `body`, co widać na rysunku 4.25.



RYSUNEK 4.25. Mimo że wewnętrzny element div (szare tło) znajduje się w kodzie XHTML wewnątrz zewnętrznego elementu div (z czerwoną górną krawędzią), dzięki deklaracji position: absolute ustawia się względem elementu body

Sprawnie posługując się marginesami i dopełnieniem, można w większości przypadków przy projektowaniu układu strony poradzić sobie tylko za pomocą pozycjonowania statycznego. Wielu początkujących użytkowników CSS zmienia rodzaj pozycjonowania prawie wszystkich elementów, przez co trudno jest im nad nimi potem zapanować. Nie należy zmieniać domyślnego pozycjonowania elementu, jeśli nie jest to absolutnie konieczne.

Jeśli teraz zmienimy pozycjonowanie zewnętrznego elementu `div` na względne (`relative`), to kontekstem pozycjonowania pozycjonowanego bezwzględnie wewnętrznego elementu `div` będzie zewnętrzny `div`, co widać na rysunku 4.26. Teraz własności `top` i `left` wewnętrznego elementu `div` ustawiają go w odniesieniu do zewnętrznego elementu `div`. Gdybyśmy obecnie zdefiniowali własności `top` i `left` zewnętrznego elementu `div` na wartości inne niż zero, wewnętrzny element `div` przesunąłby się wraz z nim, ponieważ to jest jego kontekst pozycjonowania. Jasne?



RYSUNEK 4.26. Dzięki nadaniu zewnętrznemu elementowi div pozycjonowania względnego jego pozycjonowani bezwzględnie potomkowie są rozmieszczani względem niego zgodnie z wartościami własności top i left

Teraz przejdziemy do omawiania własności `display`, a później przeanalizujemy praktyczny przykład zastosowania własności `position`.

Własność display

Poza własnością `position` każdy element ma także własność `display`. Mimo iż własność ta może przyjmować kilka wartości, najczęściej używane są dwie: `block` i `inline`. Tym, którzy przespali poprzednie rozdziały, przypominam różnicę między elementami blokowymi (`block`) a liniowymi (`inline`):

- Elementy blokowe, na przykład akapity, nagłówki czy listy, układają się jeden nad drugim w oknie przeglądarki.
- Elementy liniowe, na przykład `a`, `span` czy `img`, układają się jeden obok drugiego w oknie przeglądarki. Przechodzą do nowej linii dopiero wówczas, gdy w aktualnej nie ma dla nich wystarczająco dużo miejsca.

Możliwość zamiany elementów blokowych w liniowe i odwrotnie, jak poniżej:

domyślnie element blokowy

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

domyślnie element liniowy

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

bywa bardzo przydatna, ponieważ pozwala zmusić element liniowy do tego, aby zapełnił zawierający go element. Zastosujemy tę funkcję później do odnośników przy tworzeniu rozwijalnych menu.

Jeszcze jedną wartością własności `display`, o której warto wspomnieć, jest `none`. Powoduje ona, że element i wszystkie zagnieżdżone w nim elementy stają się niewidoczne na stronie. Miejsce normalnie zajmowane przez ten element nie jest wtedy przez niego zajmowane. Wygląda to tak, jakby kod XHTML tego elementu w ogóle nie istniał (jest jeszcze własność `visibility`, której wartość `hidden` powoduje, że element jest niewidoczny, ale nadal zajmuje przeznaczone dla niego miejsce). Dalej nauczymy się przełączać własność `display` elementów pomiędzy wartościami `none` i `block` w odpowiedzi na najechanie na te elementy kursorem myszy lub opuszczenie ich. Dzięki temu będziemy w stanie utworzyć menu rozwijalne. Także język JavaScript pozwala zmieniać wartość tej własności w odpowiedzi na działania użytkownika. Zobaczmy przykładową stronę wykorzystującą poznane ostatnio własności `position` i `display`.

Praktyczne użycie własności position i display

Latem 2007 roku pisałem arkusze stylów dla *icyou.com*, witryny, na której można obejrzeć filmy wideo dotyczące tematyki zdrowotnej. Jej właścicielem jest firma, w której pracuję: *Benefitfocus.com*. Prawie na każdej stronie tej witryny można znaleźć zestaw miniatur, których kliknięcie powoduje odtworzenie filmu. Aby zaoszczędzić nieco miejsca w oknie przeglądarki, opisy filmów zdecydowaliśmy się umieścić w chmurkach, które pojawiają się po najechaniu na miniaturę kursorem (rysunek 4.27).

RYSUNEK 4.27. Po najechaniu kursorem na miniaturę w witrynie icyou.com wyświetla się chmurka z informacją o filmie

Oto jak to zrobiliśmy. Najpierw przedstawiam kod XHTML jednej miniatury:

```
<div class="video_selection">

   <a href="#"><img src="../images/23_diabetes_testimonial-
   1.jpg" alt="Blood pressure video" /></a>
```

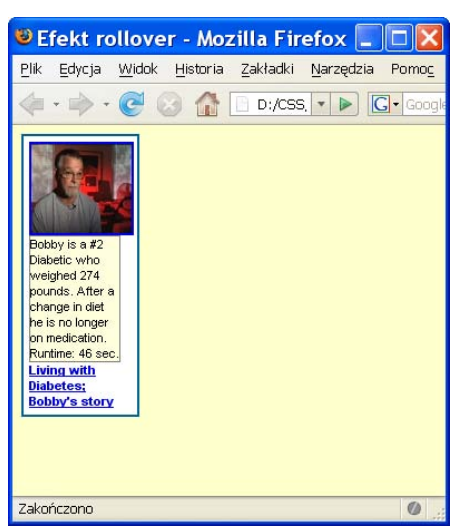
```
    <p> Bobby is a #2 Diabetic who weighed 274 pounds. After a
    change in diet he is no longer on medication. Runtime: 46
    sec.</p>
    <h2><a href="#">Living with Diabetes; Bobby's story</a></h2>
</div>
```

Kod ten pozbawiony reguł stylistycznych daje w przeglądarce następujący efekt (rysunek 4.28).

RYSUNEK 4.28. Wszystkie elementy potrzebne do utworzenia miejsca na miniaturę i związaną z nią chmurkę

Dodajemy podstawowe style dla powyższego kodu XHTML:

```
div {width:92px; border:2px solid #069; padding:5px;}
h2, p {font-size:.7em; font-family:Arial, sans-serif;
margin:0;}
p {width:80px; border:1px solid gray; padding:.3em;
background-color:#FFD;}
```



RYSUNEK 4.29. Pierwszy krok to odpowiednie sformatowanie wyglądu i szerokości elementów

Elementowi `div`, który pełni funkcję kontenera, ustawiliśmy obramowanie oraz dodaliśmy dopełnienie, aby odsunąć tekst od krawędzi (oczywiście zwiększając równocześnie jego pierwotną szerokość). Teraz element ten ma szerokość 106 pikseli (92+2+2+5+5). Dodatkowo zmieniliśmy własności czcionki w elementach `h2` i `p` oraz usunęliśmy ich domyślne marginesy.

Podobnie sformatowaliśmy akapit, ustawiając mu dopełnienie i obramowanie. Element ten będzie naszą chmurką.

Teraz zaczyna się zabawa. Wyjmiemy nasz akapit z normalnego układu elementów na stronie, zmieniając jego własność `position` na `absolute`. Jednocześnie ustawimy tę samą własność elementu `div` na `relative`, aby stał się kontekstem pozycjonowania akapitu. Należy pamiętać, że przodkiem elementu pozycjonowanego bezwzględnie musi być element pozycjonowany względnie. W tym przypadku element `div` jest rodzicem elementu `p`, a więc wszystko jest w porządku (rysunek 4.30).

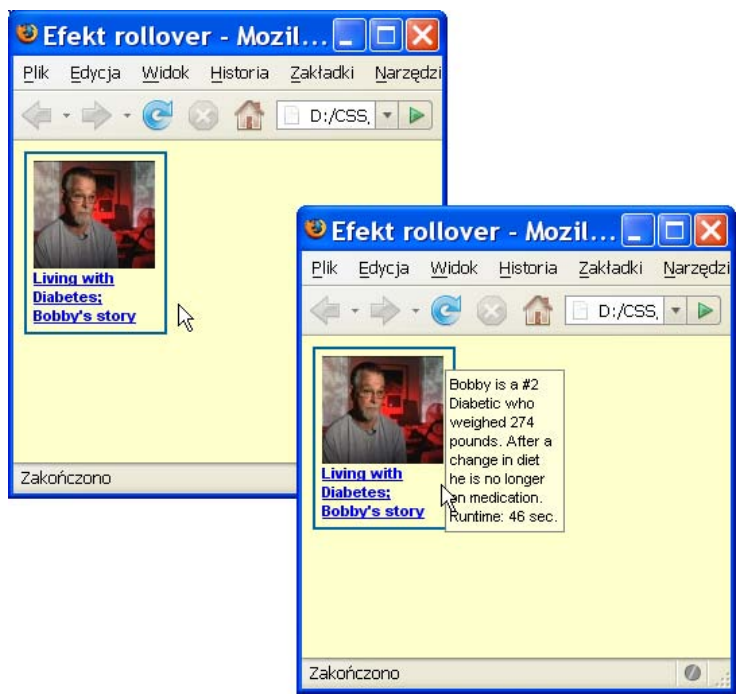
```
div {position:relative; width:92px; border:2px solid #069;
padding:5px;}

h2, p {font-size:.7em; font-family:Arial, sans-serif;}

p {position:absolute; left:96px; top:15px; width:80px;
border:1px solid gray; padding:.3em; background-color:#FFD;}
```

RYSUNEK 4.30. Teraz bezwzględnie pozycjonowany element reprezentujący chmurkę znajduje się względem elementu div w takiej pozycji, jak chcemy

Własności `left` i `top` akapitu przesunęły go nieco w prawo i w dół, dzięki czemu znajduje się on po prawej stronie obrazka.

Ostatni krok to ukrycie chmurki dopóki użytkownik nie najedzie kursorem na element `div`. Do tego celu użyjemy pseudoklasy `:hover` (rysunki 4.31a i b).

```
div {position:relative; width:92px; border:2px solid #069;
padding:5px;}

h2, p {font-size:.7em; font-family:Arial, sans-serif;
margin:0;}

p {position:absolute; display:none; width:80px; left:96px;
top:15px; border:1px solid gray; padding:.3em; background-
color:#FFD;}

div:hover p, p:hover {display:block;}
```



RYSUNKI 4.31A I B. Dzięki użyciu pseudoklasy :hover w połączeniu z własnością display uzyskaliśmy efekt chmurki pojawiającej się w wyniku najechania kursorem na obszar zajmowany przez element div

Teraz akapit z opisem jest ukryty, ponieważ jego własność `display` została ustawiona na wartość `none`. Jednak ostatni wiersz wyróżnionego kodu mówi: jeśli kursor znajdzie się nad elementem `div`, wyświetl akapit; dodatkowo jeśli kursor znajdzie się nad akapitem (co może się zdarzyć, kiedy użytkownik przesunie kursor z elementu `div` na akapit), nie ukrywaj tego akapitu. Gdy użytkownik usunie kursor znad elementu `div` lub wyświetlonego akapitu, reguła przestaje działać i akapit znika.

W ten sposób utworzyliśmy prosty efekt chmurki przy użyciu samego kodu CSS. Jest jednak pewien problem z przeglądarką IE 6. Obsługuje ona pseudoklasę `:hover` tylko dla elementu `a`.

Dlatego aby przeglądarka ta działała zgodnie z naszymi oczekiwaniami, musimy dołączyć plik JavaScript *csshover.htc* (zobacz ramkę „Pseudoklasa :hover w przeglądarce IE 6").

```
body {behavior:url(csshover.htc);}
```

Należy zauważyć, że powyższa reguła przyjmuje, iż plik *csshover.htc* znajduje się w tym samym folderze co plik XHTML. Zazwyczaj plik ten umieszcza się w osobnym folderze, razem z innymi plikami zawierającymi kod JavaScript dołączanymi do strony za pomocą adresów względnych.

Pseudoklasa :hover w przeglądarce IE 6

Przed powstaniem CSS 2 funkcję reagowania na najechanie kursorem miały tylko odnośniki. Pseudoklasa `:hover` pozwala na określenie zachowania dowolnego elementu w chwili najechania na niego kursorem. Na przykład poniższe reguły CSS powodują, że niebieskie tło elementu po najechaniu na niego kursorem zmieni się w czerwone.

```
div#respond {background-color:blue;}
div#respond:hover {background-color:red;}
```

Jest to bardzo przydatna funkcja, która stanowi podstawę menu rozwijanych opartych na CSS. Niestety, IE 6 obsługuje tę pseudoklasę tylko dla elementu `a`. Na szczęście bystry programista Peter Nederlof wpadł na pomysł, jak rozwiązać ten problem. Utworzył plik o nazwie *csshover.htc*, który trzeba dołączyć do arkusza stylów za pomocą specjalnej własności `behavior` (*www.xs4all.nl/~peterned/hovercraft.html*):

```
body {font:1em verdana, arial, sans-serif; behavior:
url(css/csshover.htc);}
```

W tym przypadku, jak wskazuje adres URL podany w regule, utworzyłem nowy folder o nazwie *css* i w nim umieściłem plik *csshover.htc*. Aby umieścić ten plik w innym miejscu, należy zmodyfikować odpowiednio adres URL.

Plik csshover.htc znajduje się wśród plików dotyczących tego rozdziału na FTP oraz w katalogu Stylib CSS w folderze JavaScript.

Po dołączeniu pliku *csshover.htc* do arkusza stylów przeglądarka IE 6 poprawnie obsługuje pseudoklasę `:hover`.

Dzięki temu rozdziałowi czytelnik powinien opanować następujące umiejętności:

- Tworzenie kolumny z wewnętrznym elementem `div` i zagnieżdżonymi elementami.
- Czyszczenie elementów pływających (własność `clear`).
- Pozycjonowanie i wyświetlanie elementów za pomocą własności `position` i `display`.

Techniki te są podstawą tworzenia układów stron w CSS, co jest tematem kolejnego rozdziału.