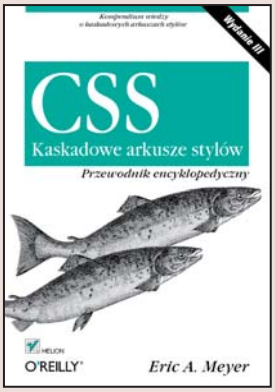
CSS. Kaskadowe arkusze stylów. Przewodnik encyklopedyczny. Wydanie III



Autor: Eric A. Meyer
Tłumaczenie: Anna Trojan
ISBN: 978-83-246-0956-7
Tytuł oryginału: CSS: The Definitive Guide
Format: B5, stron: 528

Kompendium wiedzy o kaskadowych arkuszach stylów

- Opis struktury arkusza stylów
- Sposoby definiowania charakterystyk czcionek
- Tworzenie efektownych interfejsów użytkownika
- Pozycjonowanie elementów

Kaskadowe arkusze stylów (CSS), to technologia, która zrewolucjonizowała proces tworzenia witryn internetowych. Projektanci stron WWW dostali do ręki potężne narzędzie pozwalające na definiowanie wyglądu tekstu, tabel, ramek, łączy i innych elementów dokumentu za pomocą prostych parametrów zamieszczonych w odrębnym pliku lub wydzielonym miejscu kodu strony. Dzięki arkuszom stylów możliwe jest całkowite rozdzielenie procesu edycji treści strony od określania jej aspektów typograficznych i kolorystycznych, co niezwykle ułatwia dostosowanie serwisu WWW do przeglądania na różnych urządzeniach – monitorach komputerów, wyświetlaczach telefonów komórkowych i terminali Blackberry.

Książka „CSS. Kaskadowe arkusze stylów. Przewodnik encyklopedyczny. Wydanie III" to kompleksowe źródło informacji o wszystkich aspektach stosowania kaskadowych arkuszy stylów zgodnie z najnowszymi specyfikacjami konsorcjum W3C – CSS2 oraz CSS2.1. Czytając ją, dowiesz się, jak definiować style dla tekstu, formatować tabele i tworzyć funkcjonalne interfejsy użytkownika. Poznasz różnice pomiędzy elementami pływającymi i pozycjonowanymi, zasady stosowania jednostek miar w definicjach stylów oraz metody przystosowywania treści i układu strony do druku czy odczytywania przez oprogramowanie ułatwiające korzystanie z sieci osobom niepełnosprawnym. Znajdziesz tu także informacje o tym, które elementy specyfikacji CSS są obsługiwane przez przeglądarki obecne dziś na rynku.

- Łączenie stylów z dokumentami XHTML
- Stosowanie selektorów
- Struktura definicji stylów
- Jednostki miary stosowane w definicjach stylów
- Korzystanie z różnych krojów czcionek
- Formatowanie tekstu
- Marginesy i obramowania
- Kolory
- Pozycjonowanie elementów
- Korzystanie z tabel
- Definiowanie stylów dla mediów nieekranowych

Wykorzystaj doświadczenie Erica Meyera i poznaj wszystkie aspekty stosowania CSS

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

Spis treści

Przedmowa **9**

1. CSS i dokumenty **15**
Upadek Internetu 15
CSS przychodzi na ratunek 17
Elementy 21
Łączenie CSS i XHTML 25
Podsumowanie 35

2. Selektory **37**
Podstawowe reguły 37
Grupowanie 41
Selektory klas oraz identyfikatorów 45
Selektory atrybutów 51
Wykorzystywanie struktury dokumentu 56
Pseudoklasy oraz pseudoelementy 63
Podsumowanie 72

3. Struktura oraz kaskada **73**
Specyficzność 73
Dziedziczenie 78
Kaskada 81
Podsumowanie 86

4. Wartości oraz jednostki **87**
Wartości liczbowe 87
Wartości procentowe 87
Kolory 88
Jednostki długości 93
Adresy URL 99

Jednostki CSS2 101
Podsumowanie 102

5. Czcionki ... 103
Rodziny czcionek 104
Waga czcionki 108
Rozmiar czcionki 114
Style i warianty 121
Rozciąganie i dostosowywanie czcionki 125
Właściwość font 127
Dobieranie czcionek 132
Podsumowanie 134

6. Właściwości tekstu ... 137
Wcięcia oraz wyrównanie poziome 137
Wyrównanie w pionie 142
Odstępy pomiędzy słowami oraz literami 151
Transformacja tekstu 154
Dekoracja tekstu 156
Cień tekstu 160
Obsługa białych znaków 161
Kierunek tekstu 163
Podsumowanie 165

7. Podstawowe formatowanie wizualne ... 167
Podstawowe pojemniki 167
Elementy blokowe 170
Elementy wewnętrzne 188
Zmiana wyświetlania elementu 207
Podsumowanie 214

8. Dopełnienie, obramowanie oraz marginesy ... 215
Podstawowe pojemniki elementów 215
Marginesy 219
Obramowanie 231
Dopełnienie 245
Podsumowanie 251

9. Kolory i tła ... 253
Kolory 253
Kolory pierwszego planu 254
Tło 260
Podsumowanie 287

10. Pływanie oraz pozycjonowanie 289
Pływanie 289
Pozycjonowanie 307
Podsumowanie 342

11. Układ tabel 343
Formatowanie tabel 343
Obramowanie komórek tabeli 355
Rozmiar tabeli 363
Podsumowanie 372

12. Listy oraz zawartość generowana 373
Listy 373
Zawartość generowana 381
Podsumowanie 395

13. Style interfejsu użytkownika 397
Czcionki oraz kolory systemowe 397
Kursory 402
Obrysy 406
Podsumowanie 412

14. Media nieekranowe 413
Przydzielanie arkuszy stylów dla określonego medium 413
Media stronicowe 414
Style dźwiękowe 431
Podsumowanie 448

A Przewodnik po właściwościach 449

B Selektory, pseudoklasy oraz pseudoelementy 493

C Przykładowy arkusz stylów HTML 4 501

Skorowidz 505

ROZDZIAŁ 10.

Pływanie oraz pozycjonowanie

CSS rzeczywiście umożliwia atrakcyjny wygląd zawartości dokumentu dzięki zmianom czcionek, tła i pozostałych elementów, jednak jak można osiągnąć podstawowe możliwości wpływające na układ elementów? Dzieje się to dzięki *pływaniu* (ang. *floating*) oraz *pozycjonowaniu* (ang. *positioning*). Są to narzędzia, które pozwalają uzyskać układ kolumnowy, nakładać jeden fragment dokumentu na drugi i, ogólnie mówiąc, otrzymać wszystko to, do czego przez długie lata wykorzystywane były ogromne ilości tabel.

Idea przyświecająca pozycjonowaniu jest względnie prosta. Pozycjonowanie pozwala dokładnie określić, w którym miejscu powinien znajdować się pojemnik elementu w stosunku do miejsca, w jakim normalnie by się pojawił — lub w stosunku do elementu rodzica, innego elementu albo nawet względem samego okna przeglądarki. Możliwości tej cechy są jednocześnie i oczywiste, i zaskakujące. Nie będzie zaskakującą wiadomością to, że przeglądarki implementują ten element CSS2 o wiele lepiej od innych.

Z kolei pływanie pojawiło się już w CSS1 — w oparciu o możliwość dodaną przez Netscape we wczesnych latach istnienia Internetu. Pływanie nie jest dokładnie pozycjonowaniem, jednak z pewnością nie należy również do standardowego, normalnego układu. W dalszej części niniejszego rozdziału zostanie wytłumaczone, na czym ono dokładnie polega.

Pływanie

Koncepcja elementów pływających (ang. *floated elements*) jest z pewnością znana. Już od czasu Netscape 1 możliwe było zadeklarowanie — na przykład — `<img src="b5.gif" align="right">` w celu uzyskania pływających obrazków. Powodowało to, że obrazek „spływał" na prawą stronę i pozwalał na to, by inna treść (taka jak tekst) „opływała" go. Nazwa „pływanie" pochodzi zresztą z dokumentu *Extensions to HTML 2.0*, w którym napisano:

> Dodatki do opcji `ALIGN` wymagają dużej dozy wyjaśnień. Na początek — wartości „left" oraz „right". Obrazki z takim wyrównaniem stanowią zupełnie nowy, *pływający* typ obrazków.

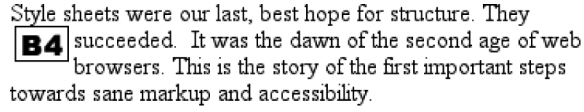
W przeszłości możliwe było jedynie uzyskanie pływających obrazków oraz w niektórych przeglądarkach również tabel. CSS z kolei pozwala na to w przypadku dowolnego elementu, od obrazków i akapitów aż po listy. W CSS takie zachowanie można uzyskać dzięki właściwości `float`.

float	
Wartości:	`left	right	none	inherit`
Wartość początkowa:	`none`
Stosuje się do:	Wszystkich elementów
Dziedziczona:	Nie
Wartość wyliczona:	Jak określono

Żeby na przykład uzyskać obrazek pływający do lewa, można wykorzystać poniższy kod:

```
<img src="b4.gif" style="float: left;" alt="b4">
```

Jak wyraźnie widać na rysunku 10.1, obrazek „spływa" do lewej strony okna przeglądarki, natomiast tekst go opływa. Dokładnie tego można się było spodziewać.



Rysunek 10.1. Pływający obrazek

W przypadku elementów pływających w CSS pojawiają się pewne interesujące kwestie.

Elementy pływające

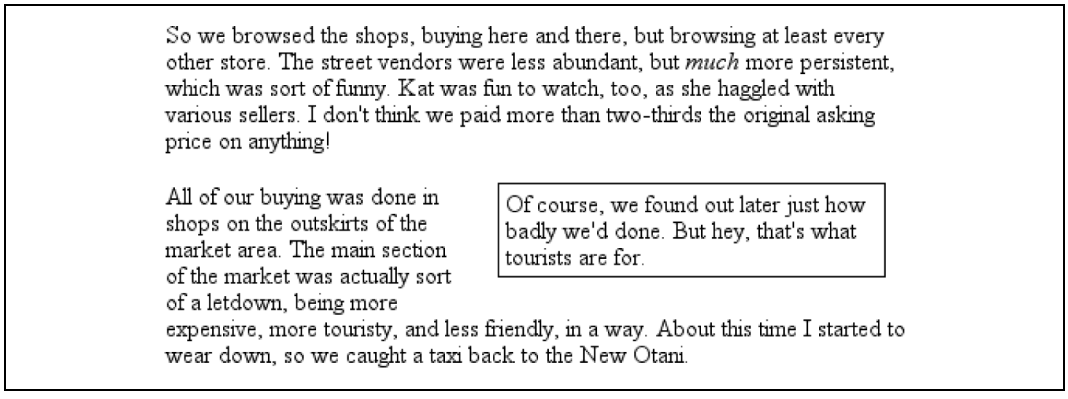
Kiedy ma się do czynienia z elementami pływającymi, należy pamiętać o kilku sprawach. Po pierwsze, element pływający jest w pewnym sensie wyłączany z normalnego układu dokumentu, choć nadal na niego wpływa. W sposób całkowicie unikalny dla CSS elementy pływające istnieją jakby na swojej własnej płaszczyźnie, jednak w dalszym ciągu wpływają na resztę dokumentu.

Ten wpływ bierze się z faktu, iż kiedy element jest pływający, pozostała treść dokumentu opływa go. Jest to zachowanie znane z pływających obrazków, jednak tak samo dzieje się na przykład w przypadku pływającego akapitu. Na rysunku 10.2 widać ten efekt dość wyraźnie dzięki marginesowi dodanemu do pływającego akapitu:

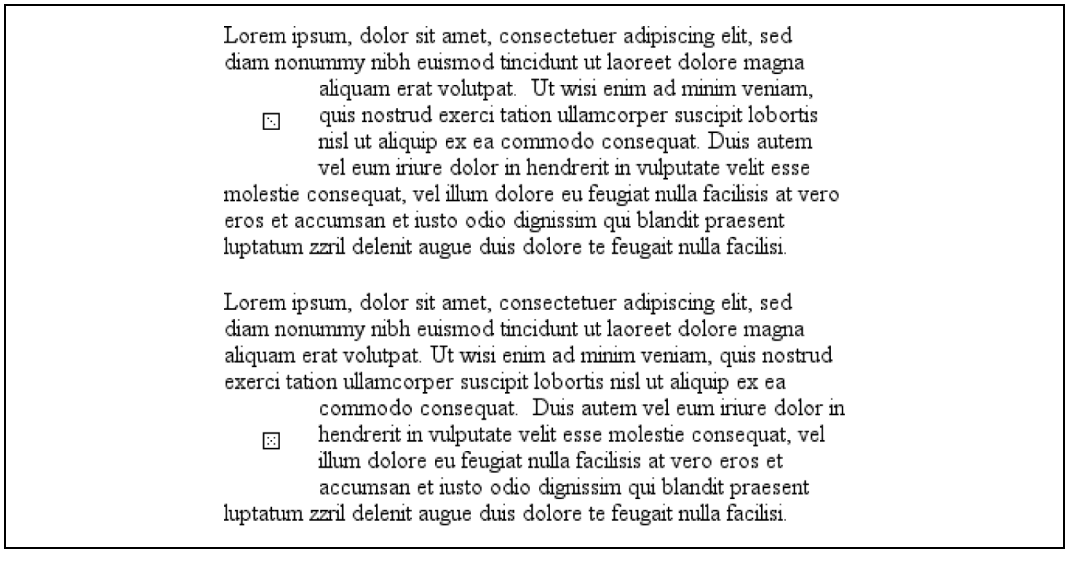
```
p.aside {float: right; width: 15em; margin: 0 1em 1em; padding: 0.25em; border: 1px
solid;}
```

Jedną z pierwszych interesujących kwestii, na które warto zwrócić uwagę, jest fakt, iż marginesy wokół elementów pływających nie ulegają składaniu. Jeśli elementem pływającym jest obrazek z marginesami o wartości dwudziestu pikseli, wokół tego obrazka znajdzie się pusta przestrzeń o szerokości co najmniej dwudziestu pikseli. Jeśli inne elementy przylegające do tego obrazka — zarówno w poziomie, jak i w pionie — również mają marginesy, te marginesy nie będą się składać z marginesami pływającego obrazka, jak widać na rysunku 10.3:

```
p img {float: left; margin: 25px;}
```



Rysunek 10.2. Pływający akapit



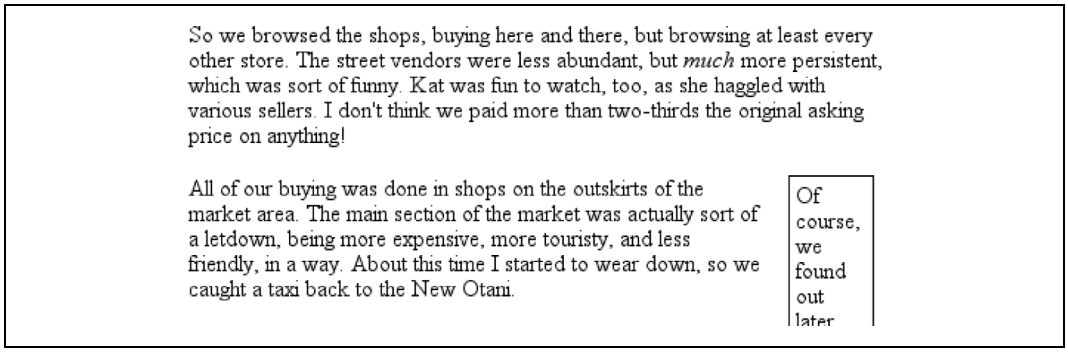
Rysunek 10.3. Pływające obrazki z marginesami

Wracając do analogii z papierem i plastikiem z rozdziału 7. — plastikowe marginesy wokół obrazka *nigdy* nie będą się nakładać na plastik otaczający inne elementy pływające.

Jeśli element pływający jest elementem niezastępowanym, należy zadeklarować wartość jego właściwości `width`. W przeciwnym razie, zgodnie ze specyfikacją CSS, szerokość tego elementu będzie dążyła do zera. W ten sposób szerokość pływającego akapitu może wynosić dosłownie jeden znak — przy założeniu, że jest to minimalna wartość właściwości `width` w interpretacji przeglądarki. Jeśli nie zadeklaruje się wartości `width` dla elementów pływających, można otrzymać coś podobnego do efektu zaprezentowanego na rysunku 10.4 (jest to faktycznie mało prawdopodobne, ale jednak możliwe).

Brak pływania

Istnieje jeszcze jedna wartość właściwości `float` poza `left` oraz `right`. Wartość `float: none` wykorzystywana jest do wykluczenia możliwości pływania dla danego elementu.



Rysunek 10.4. Pływający tekst bez jawnie zadeklarowanej szerokości

Może się to wydawać nieco niedorzeczne, skoro najprostszym sposobem wykluczenia możliwości pływania wydaje się po prostu uniknięcie deklarowania właściwości `float`, prawda? Cóż, przede wszystkim domyślną wartością właściwości `float` jest `none`. Innymi słowy, wartość ta musi istnieć, żeby możliwe było normalne, niepływające zachowanie. Bez niej wszystkie elementy musiałyby w jakiś sposób pływać.

Po drugie, można chcieć nadpisać pewien styl z importowanego arkusza stylów. Można sobie wyobrazić, że dla całego serwera wykorzystuje się jeden arkusz stylów, w którym obrazki są pływające. Na jednej wybranej stronie nie chce się jednak, by obrazki były pływające. Zamiast pisać zupełnie nowy arkusz stylów, wystarczy umieścić regułę `img {float: none;}` w osadzonym arkuszu stylów dokumentu. Poza tymi okolicznościami nie istnieje jednak raczej inne zastosowanie dla deklarowania `float: none`.

Pływanie — szczegóły

Przed przejściem do szczegółów związanych z pływaniem warto wyjaśnić koncepcję *bloku zawierającego element* (ang. *containing block*). Blokiem zawierającym element pływający jest najbliższy element blokowy będący przodkiem tego elementu. Dlatego też w poniższym fragmencie kodu blokiem zawierającym element pływający jest element akapitu, który go zawiera:

```
<h1>Test</h1>
<p>
   To jest tekst akapitu, co widać wyraźnie. Wewnątrz zawartości tego akapitu znajduje
   się pływający obrazek.
   <img src="testy.gif" style="float: right;">
   Blokiem zawierającym pływający obrazek jest akapit.
</p>
```

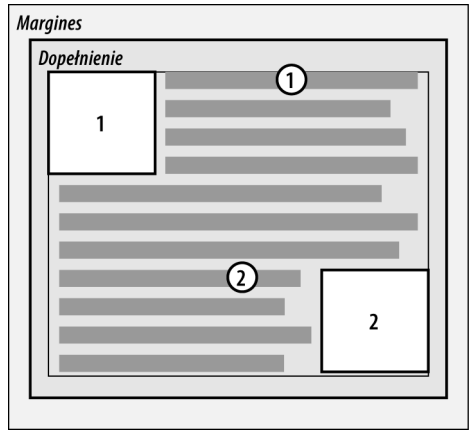
Do koncepcji bloków zawierających element wraca się w dalszej części niniejszego rozdziału, poświęconej pozycjonowaniu.

Elementy pływające generują ponadto pojemnik blokowy bez względu na typ samego elementu. Jeśli więc odnośnikowi (traktowanemu zazwyczaj jako element wewnętrzny i tworzącemu pojemnik wewnętrzny) przypisze się wartość właściwości `float`, to jako element pływający wygeneruje on pojemnik blokowy. Jego układ i zachowanie będą podobne do — na przykład — elementu `div`. Nierzadko spotyka się deklarowanie `display: block` dla elementu pływającego, jednak nie jest to konieczne.

Umieszczeniem elementu pływającego rządzi pewien zbiór ścisłych zasad, które warto omówić przed przejściem do konkretnych przykładów zachowań. Są one nieco podobne do zasad, które zajmują się wyliczaniem marginesów i szerokości, gdyż mają ten sam zdroworozsądkowy punkt wyjścia. A zasady są następujące:

1. Lewy (lub prawy) zewnętrzny brzeg elementu pływającego nie może znajdować się po lewej (lub prawej) stronie wewnętrznego brzegu elementu rodzica.

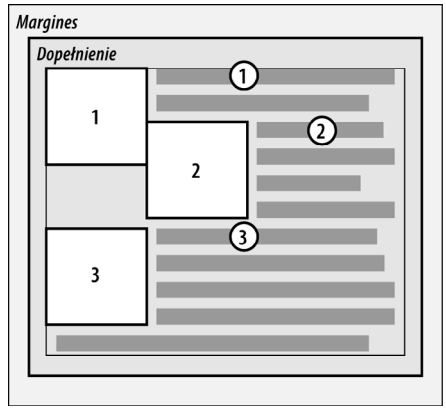
   Jest to wystarczająco proste. Wysunięta najdalej w lewo pozycja, w jakiej może zostać umieszczony lewy zewnętrzny brzeg pływającego po lewej stronie elementu, znajduje się na wewnętrznym lewym brzegu elementu rodzica; podobnie wysunięta najdalej w prawo pozycja, w której może zostać umieszczony prawy zewnętrzny brzeg pływającego po prawej stronie elementu, znajduje się na wewnętrznym prawym brzegu elementu rodzica. Przedstawiono to na rysunku 10.5. Na tym i na następnych rysunkach liczby w kółkach pokazują pozycję, w której faktycznie znajduje się element kodu względem źródła, natomiast ponumerowane prostokąty pokazują widziane pozycje oraz rozmiary elementów pływających.



Rysunek 10.5. Pływanie do lewa (bądź prawa)

2. Lewy (lub prawy) zewnętrzny brzeg elementu pływającego musi znajdować się po prawej (lewej) stronie prawego (lewego) zewnętrznego brzegu pływającego z lewej (prawej) elementu, który pojawia się wcześniej w źródle dokumentu, chyba że górny brzeg późniejszego elementu znajduje się poniżej dolnego brzegu pierwszego.

   Zasada ta zapobiega zachodzeniu pływających elementów na siebie. Jeśli element ma pływać po lewej stronie, ale znajduje się tam już inny pływający element ze względu na jego wcześniejsze umieszczenie w źródle dokumentu, to późniejszy element umieszczany jest obok zewnętrznego prawego brzegu wcześniejszego elementu pływającego. Jeśli jednak góra pływającego elementu znajduje się poniżej dołu wcześniejszego elementu pływającego, to może on płynąć aż do wewnętrznego lewego brzegu elementu rodzica. Przykłady tego zachowania pokazane są na rysunku 10.6.

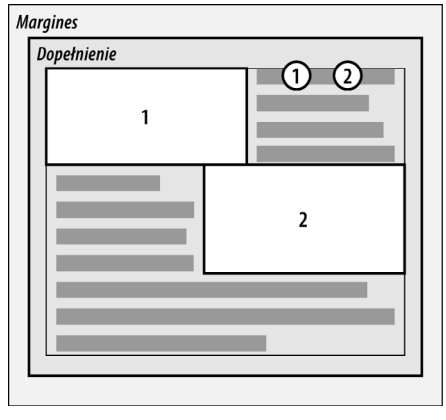


Rysunek 10.6. Zapobieganie nakładaniu się elementów pływających

Istnieje pewna zaleta tej zasady — skoro nie trzeba martwić się o przysłanianie jednego pływającego elementu drugim, to można mieć pewność, że wszystkie pływające elementy dokumentu będą widoczne. Dzięki temu pływanie jest dosyć bezpiecznym efektem. Sytuacja jest wyraźnie inna niż przy zastosowaniu pozycjonowania, w którym bardzo łatwo spowodować zasłonięcie jednego elementu drugim.

3. Prawy zewnętrzny brzeg pływającego z lewej elementu nie może znajdować się po prawej stronie lewego zewnętrznego brzegu żadnego pływającego w prawo elementu, który umieszczony jest po jego prawej stronie. Lewy zewnętrzny brzeg pływającego z prawej elementu nie może znajdować się po lewej stronie prawego zewnętrznego brzegu żadnego pływającego w lewo elementu umieszczonego z jego lewej strony.

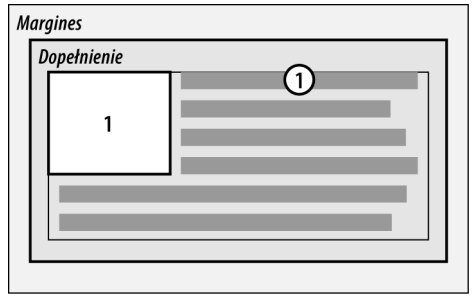
   Zasada ta zapobiega zachodzeniu pływających elementów na siebie. Załóżmy, że element `body` ma pięćset pikseli szerokości, a jego jedyną zawartością są dwa obrazki o szerokości trzystu pikseli. Pierwszy pływa z lewej, drugi z prawej strony. Powyższa zasada uniemożliwi drugiemu obrazkowi zasłonięcie stu pikseli pierwszego obrazka. Drugi obrazek zostanie zepchnięty w dół tak, że jego górny brzeg znajdzie się poniżej dołu pływającego z lewej strony obrazka, jak przedstawiono na rysunku 10.7.



Rysunek 10.7. Dalsze zapobieganie nakładaniu się elementów pływających

4. Górny brzeg elementu pływającego nie może znajdować się wyżej niż wewnętrzny górny brzeg rodzica. Jeśli element pływający znajduje się pomiędzy dwoma składającymi się marginesami, umieszczany jest w taki sposób, jakby posiadał on element blokowy rodzica znajdujący się pomiędzy tymi dwoma elementami.

   Pierwsza część tej zasady jest stosunkowo prosta i powstrzymuje elementy pływające przed przedostaniem się na samą górę dokumentu. Poprawne zachowanie przedstawione jest na rysunku 10.8. Druga część tej zasady dostosowuje wyrównanie w sytuacjach, w których na przykład środkowy z trzech akapitów jest pływający. W takim przypadku akapit pływający pływa tak, jakby miał on rodzica będącego elementem blokowym (takim jak `div`). Zapobiega to przesuwaniu się pływającego akapitu na górę elementu rodzica, który dzielą ze sobą wszystkie trzy akapity.



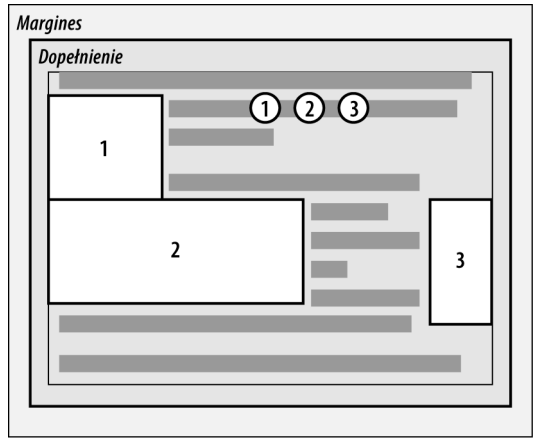
Rysunek 10.8. W przeciwieństwie do balonów elementy pływające nie mogą odpłynąć do góry

5. Górny brzeg elementu pływającego nie może znaleźć się wyżej niż górny brzeg innego wcześniejszego elementu pływającego lub blokowego.

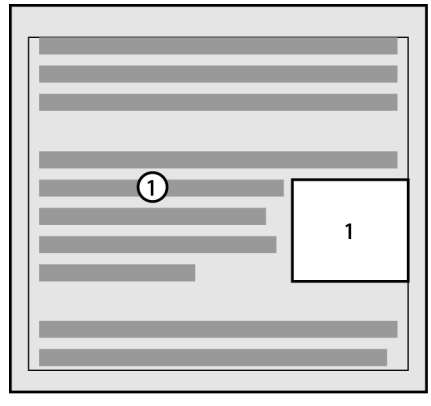
   Podobnie jak w przypadku zasady 4., pozwala to uniknąć wypłynięcia elementów pływających na samą górę elementów rodziców. Jest także niemożliwe, aby górny brzeg elementu pływającego znalazł się powyżej górnego brzegu elementu pływającego, który w kodzie źródłowym dokumentu pojawił się wcześniej. Rysunek 10.9 jest tego przykładem; widać na nim, że ponieważ drugi element pływający został zmuszony do przesunięcia się poniżej pierwszego, góra trzeciego elementu pływającego jest równa z górą drugiego elementu, a nie pierwszego.

6. Górny brzeg elementu pływającego nie może być wyświetlony wyżej niż góra jakiegokolwiek pojemnika wiersza zawierającego pojemnik wygenerowany przez element, który poprzedza element pływający w źródle dokumentu.

   Podobnie jak zasady 4. i 5., zasada ta dalej ogranicza wypływanie elementu w górę poprzez uniemożliwienie wyświetlenia go powyżej górnego brzegu wiersza zawierającego treść poprzedzającą element pływający. Powiedzmy, że w samym środku akapitu znajduje się pływający obrazek. Najwyższe miejsce, w którym może się znaleźć górny brzeg tego obrazka, to góra pojemnika wiersza, z którego wywodzi się obrazek. Jak widać na rysunku 10.10, powstrzyma to obrazek przed uniesieniem się zbyt wysoko.



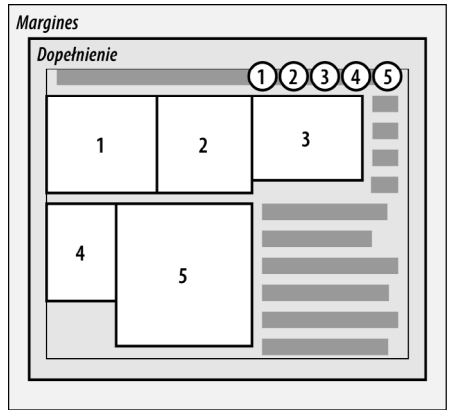
Rysunek 10.9. Utrzymywanie elementów pływających poniżej ich poprzedników



Rysunek 10.10. Utrzymanie elementów pływających na wysokości ich otoczenia

7. Element pływający z lewej (prawej) strony mający inny pływający element po swojej lewej (prawej) stronie nie może mieć swojego prawego (lewego) zewnętrznego brzegu po prawej (lewej) stronie prawego (lewego) brzegu zawierającego go bloku.

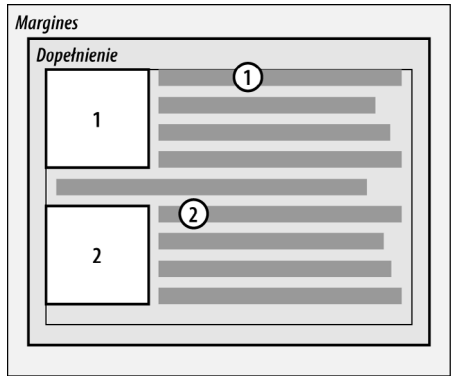
   Innymi słowy — element pływający nie może wystawać poza brzeg elementu go zawierającego, chyba że sam jest zbyt szeroki. Zapobiega to sytuacji, w której seria następujących po sobie elementów pływających mogłaby się pojawić w jednej linii poziomej i wychodziłaby zbyt daleko poza brzegi bloku je zawierającego. Zamiast tego — jeśli element pływający miałby wystawać poza zawierający go blok przez to, że występuje obok innych elementów — zostanie przesunięty niżej (pod poprzednie elementy pływające), jak przedstawia to rysunek 10.11 (na którym elementy pływające zaczynają się w następnym wierszu w celu wyraźniejszego zilustrowania działania zasady). Zasada pojawiła się po raz pierwszy w CSS2 i zniwelowała niedociągnięcia z CSS1.



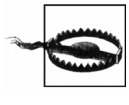
Rysunek 10.11. Jeśli nie ma miejsca, elementy pływające spychane są do nowego „wiersza"

8. Element pływający musi zostać umieszczony najwyżej jak to tylko możliwe.

   Zasada ta podlega oczywiście ograniczeniom przedstawionym w poprzednich siedmiu zasadach. Historycznie rzecz biorąc, przeglądarki wyrównywały górę elementu pływającego z górą pojemnika wiersza znajdującego się za tym, w którym pojawiał się znacznik elementu. Jednak zasada 8. sugeruje, że jego górny brzeg powinien być na równi z górą pojemnika tego wiersza, w którym znajduje się jego znacznik (przy założeniu, że jest wystarczająco dużo miejsca na wykonanie takiej operacji). Teoretycznie poprawne zachowania pokazane zostały na rysunku 10.12.



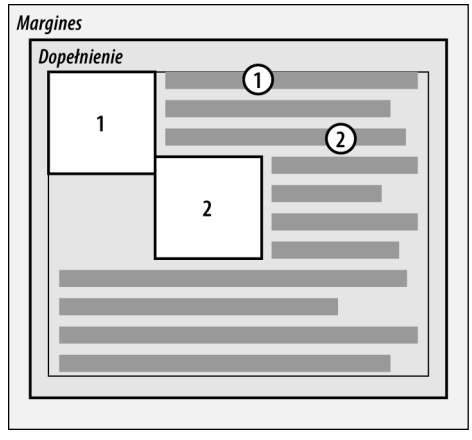
Rysunek 10.12. Uwzględniając pozostałe ograniczenia, należy umieścić element pływający tak wysoko, jak jest to możliwe



Jako że nie ma precyzyjnej definicji określenia „najwyżej, jak to jest możliwe" (co mogłoby oznaczać „najwyżej, jak jest to dogodnie możliwe"), nie można polegać na konsekwentnym zachowaniu przeglądarek (nawet uważanych za zgodne z CSS). Niektóre przeglądarki będą stosować się do historycznego zwyczaju i wyświetlać pływający obrazek w następnym wierszu, podczas gdy inne wyświetlą go w danym wierszu, jeśli tylko będzie na to miejsce.

9. Pływający z lewej element musi być umieszczony najbardziej z lewej strony jak tylko to możliwe, a pływający z prawej element — najbardziej z prawej strony jak tylko to możliwe. Preferowane jest raczej przesunięcie w górę niż przesunięcie w lewo lub prawo.

   Również ta zasada podlega ograniczeniom wynikającym z poprzednich zasad. Istnieją podobne zastrzeżenia, jak w przypadku zasady 8., chociaż nie są one tak mętne. Jak widać na rysunku 10.13, stosunkowo łatwo można określić, czy element posunął się najdalej, jak to było możliwe, w lewo lub w prawo.



Rysunek 10.13. Uwzględniając pozostałe ograniczenia, należy umieścić element pływający tak wysoko, jak jest to możliwe

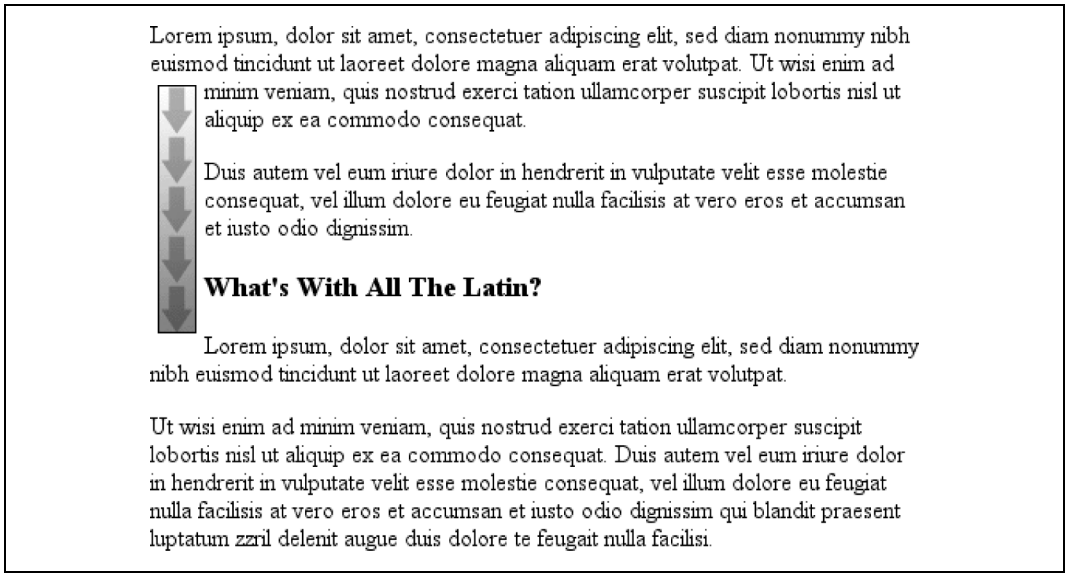
Rzeczywiste zachowanie

Z powyższych zasad wynika kilka ciekawych konsekwencji, które wiążą się z tym, co mówią oraz czego nie dopowiadają owe zasady. Najpierw należy powiedzieć, co dzieje się, gdy element pływający jest wyższy od elementu rodzica.

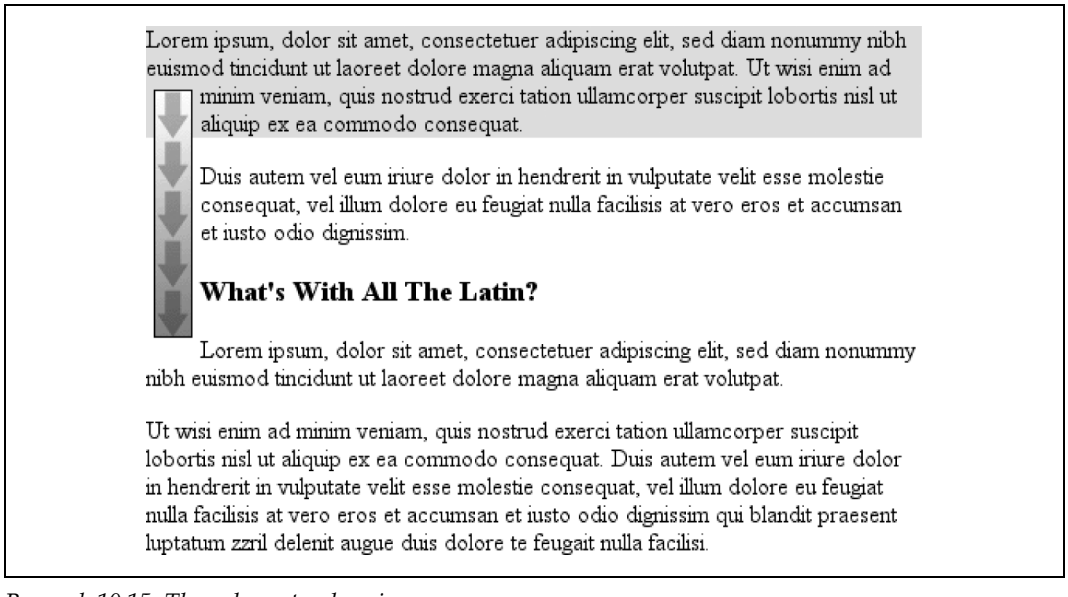
Właściwie zdarza się tak dość często. Weźmy za przykład dokument składający się z nie więcej niż kilku akapitów i elementów `h3`, w którym pierwszy akapit zawiera pływający obrazek. Dalej okazuje się, że pływający obrazek ma margines wielkości pięciu pikseli (`5px`). Można się spodziewać, że dokument zostanie wyświetlony tak, jak na rysunku 10.14.

Nie ma w tym oczywiście nic niezwykłego, ale rysunek 10.15 pokazuje, co się dzieje, jeśli do pierwszego akapitu doda się tło.

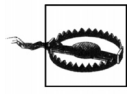
Drugi przykład nie różni się od pierwszego niczym innym niż to, że widoczne jest tło. Jak widać, element pływający wystaje poza dolny brzeg elementu rodzica. Oczywiście tak samo wystawał w pierwszym przykładzie, ale ponieważ nie widać było tła, nie było to aż takie oczywiste. Omówione wcześniej reguły pływania dotyczą jedynie lewych, prawych oraz górnych brzegów elementów pływających i ich rodziców. Zamierzone pominięcie dolnych brzegów w zasadach pływania wiąże się z zachowaniem przedstawionym na rysunku 10.15.



Rysunek 10.14. Oczekiwane zachowanie pływania



Rysunek 10.15. Tło a elementy pływające



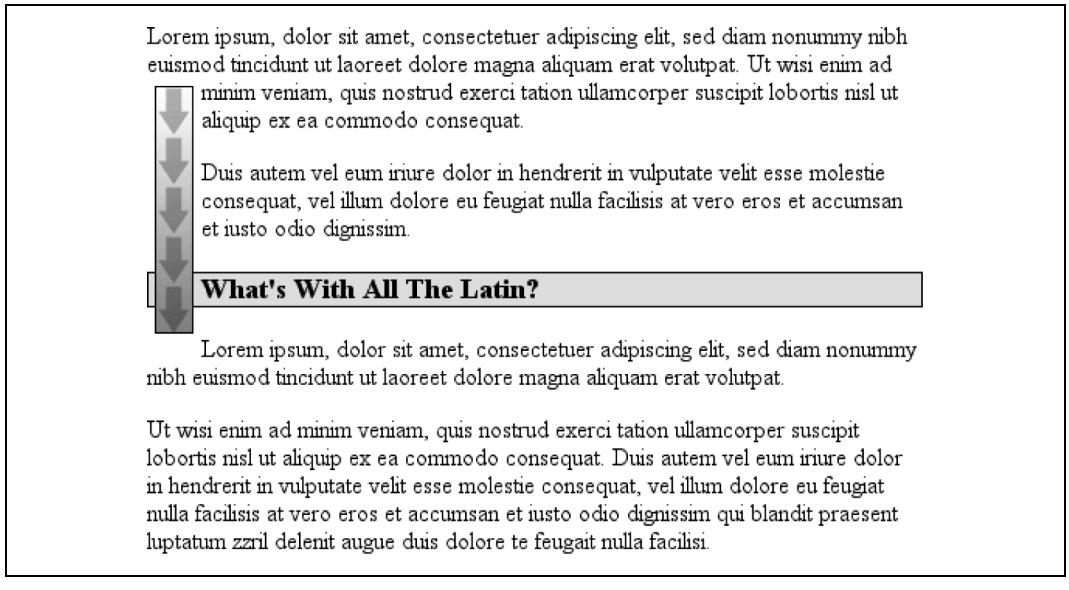
W praktyce niektóre przeglądarki nie zachowują się poprawnie i zwiększają wysokość elementu rodzica tak, by zmieścił się w nim element pływający (nawet jeśli spowoduje to wyświetlenie olbrzymiej ilości pustej przestrzeni wewnątrz elementu rodzica).

Specyfikacja CSS2.1 wyjaśniła jeden z aspektów zachowania elementów pływających: element pływający rozszerza się tak, by zmieścić wszystkie pływające elementy będące jego potomkami. W poprzednich wersjach specyfikacji CSS nie było jasne, co powinno się stać w takiej sytuacji.

W ten sposób można umieścić element pływający wewnątrz jego elementu rodzica poprzez uczynienie rodzica pływającym, jak w poniższym przykładzie:

```
<div style="float: left; width: 100%;">
   <img src="hay.gif" style="float: left;">
   Element 'div' będzie się rozszerzał wokół elementu pływającego, ponieważ sam jest
   elementem pływającym.
</div>
```

Pokrewnym tematem jest zagadnienie dotyczące tła i jego związku z elementami pływającymi, które pojawiają się wcześniej w strukturze dokumentu. Zostało to zobrazowane na rysunku 10.16.



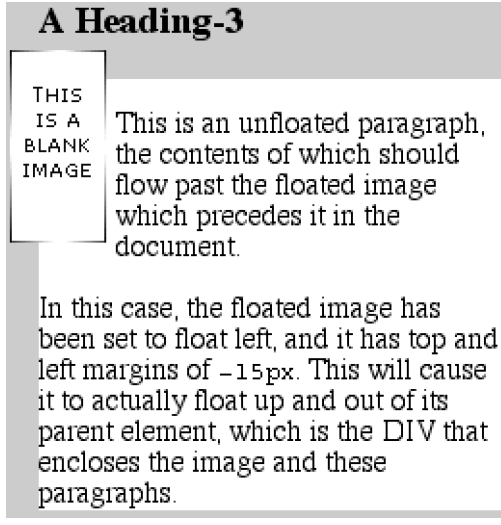
Rysunek 10.16. Tło elementów „wślizguje się" pod elementy pływające

Ponieważ element pływający znajduje się jednocześnie w układzie zawartości dokumentu i poza nim, takie rzeczy muszą się zdarzać. Co się dzieje? Zawartość nagłówka jest „wypierana" przez element pływający. Szerokość elementu nagłówka pozostaje jednak równa szerokości elementu rodzica, dlatego obszar zawartości elementu rozciąga się na całą szerokość obszaru rodzica (a z nim także i tło). By uniknąć zasłonięcia przez element pływający, rzeczywista zawartość elementu nie płynie przez całą szerokość swojego obszaru zawartości.

Marginesy ujemne

Marginesy ujemne mogą spowodować, że element pływający zostanie wysunięty poza obszar elementu rodzica. Wydaje się, że jest to sprzeczne z wyjaśnionymi wcześniej zasadami, ale tak nie jest. W taki sam sposób, w jaki elementy mogą przy użyciu marginesów ujemnych okazać się szersze od swoich rodziców, elementy pływające mogą wystawać na zewnątrz obszaru rodziców.

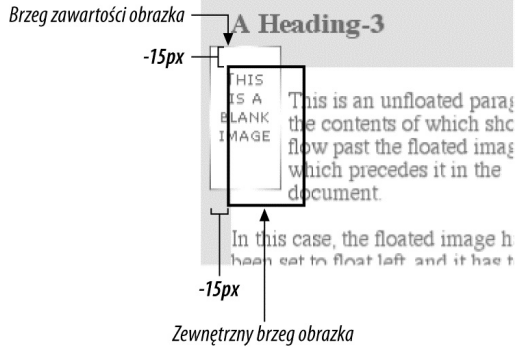
Raz jeszcze można rozważyć przypadek obrazka pływającego z lewej strony, który ma górny i lewy margines o wartości `-15px`. Obrazek umieszczony jest wewnątrz elementu `div`, który nie posiada dopełnienia, obramowania ani marginesów. Rezultat zaprezentowano na rysunku 10.17.



Rysunek 10.17. Pływanie z marginesami ujemnymi

Wbrew pozorom z technicznego punktu widzenia nie narusza to ograniczeń związanych z umieszczeniem elementów pływających poza obszarem elementu rodzica.

Dokładne wczytanie się w zasady wymienione w poprzednim podrozdziale pozwoli odkryć, skąd bierze się takie zachowanie: zewnętrzne brzegi elementu pływającego muszą znajdować się wewnątrz elementu rodzica. Jednak marginesy ujemne mogą przemieścić zawartość elementu pływającego w taki sposób, że zachodzić ona będzie na jego zewnętrzny brzeg, jak przedstawiono to na rysunku 10.18.



Rysunek 10.18. Szczegóły pływania do góry i w lewo dzięki marginesom ujemnym

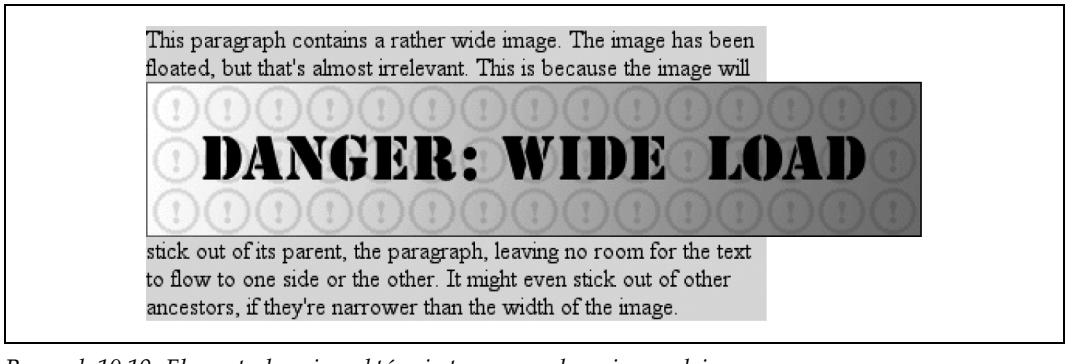
Rachunki w tej sytuacji wyglądają następująco: załóżmy, że górny wewnętrzny brzeg elementu `div` znajduje się w pozycji stu pikseli. Przeglądarka — aby dowiedzieć się, gdzie powinien znajdować się górny wewnętrzny brzeg elementu pływającego — wykona następujące obliczenia: `100px + (-15px) marginesu + 0 dopełnienia = 85px`. Zatem górny wewnętrzny brzeg pływającego elementu powinien znajdować się w pozycji osiemdziesięciu pięciu pikseli.

Pomimo że znajduje się ona powyżej górnego wewnętrznego brzegu rodzica elementu, matematyka rozwiązuje to tak, że reguły specyfikacji nie zostają naruszone. Podobny sposób rozumowania wyjaśnia, w jaki sposób lewy wewnętrzny brzeg elementu pływającego może zostać umieszczony na lewo od lewego wewnętrznego brzegu rodzica.

W tej chwili wiele osób może mieć nieodpartą ochotę, żeby krzyknąć: „Faul!". Osobiście nie obwiniam ich za to. Przyzwolenie na umieszczenie górnego brzegu wewnętrznego nad górnym brzegiem zewnętrznym wydaje się całkowicie błędne, ale przy ujemnym górnym marginesie jest dokładnie tym, co się otrzymuje. Używając marginesów ujemnych dla normalnych elementów niepływających, można sprawić, że będą one wizualnie szersze niż ich przodkowie. To samo sprawdza się w przypadku wszystkich czterech boków pojemnika elementu pływającego. Wystarczy, że ustawi się ujemną wartość marginesów, a zawartość wysunie się poza zewnętrzny brzeg, formalnie nie naruszając przy tym specyfikacji.

W tym miejscu pojawia się jedno ważne pytanie — co się stanie z wyglądem dokumentu, jeśli dzięki użyciu marginesów ujemnych element wypłynie poza obszar elementu rodzica? Obrazek może na przykład wypłynąć tak daleko, że przykryje akapit, który został już wcześniej wyświetlony przez przeglądarkę. W takim przypadku to od przeglądarki zależy, czy dokument zostanie ponownie przerysowany. Specyfikacje CSS1 oraz CSS2 wyraźnie stwierdzają, że przeglądarki nie muszą przystosowywać wyświetlonej wcześniej zawartości do tego, co później dzieje się w dokumencie. Innymi słowy — jeśli obrazek wpłynie na obszar poprzedniego akapitu, to może po prostu zasłonić to, co już się tam znajduje. Z drugiej strony przeglądarka może inaczej rozwiązać taki problem (na przykład poprzez „oblanie" elementu zawartością, mimo że takie zachowanie nie jest wymagane). Raczej nie należy liczyć na szczególne zachowanie przeglądarki, co sprawia, że przy pływaniu przydatność marginesów ujemnych jest ograniczona. Zawieszone (czyli wysunięte poza obrys akapitu) elementy pływające są prawdopodobnie stosunkowo bezpieczne, ale próby wypchnięcia elementu w górę strony są, ogólnie rzecz biorąc, złym pomysłem.

Istnieje jeszcze jeden sposób, w jaki element pływający może przekroczyć wewnętrzny lewy lub prawy brzeg elementu rodzica — dzieje się tak wtedy, gdy element pływający jest szerszy niż rodzic. W takim przypadku element pływający— aby jak najlepiej wyświetlić swoją zawartość — po prostu będzie wystawał poza prawy lub lewy wewnętrzny brzeg (w zależności od tego, z której strony pływa). Doprowadzi to do rezultatu przedstawionego na rysunku 10.19.



Rysunek 10.19. Element pływający, który jest szerszy od swojego rodzica

Pływanie, zawartość i nakładanie się

Jeszcze bardziej interesującą kwestią jest to, co się stanie, kiedy element pływający nałoży się na zawartość w normalnym układzie dokumentu. Może się tak zdarzyć na przykład wtedy, gdy element pływający ma ujemny margines po stronie, którą opływa zawartość (czyli na przykład w przypadku ujemnego marginesu lewego w elemencie pływającym z prawej). Pokazane zostało już, co stanie się w przypadku obramowania oraz tła elementów blokowych. Jednak co z elementami wewnętrznymi?

Ze specyfikacji CSS1 oraz CSS2 nie wynika zbyt jasno, co w takim przypadku będzie zachowaniem oczekiwanym. Specyfikacja CSS2.1 wyjaśniła to zagadnienie za pomocą następujących reguł:

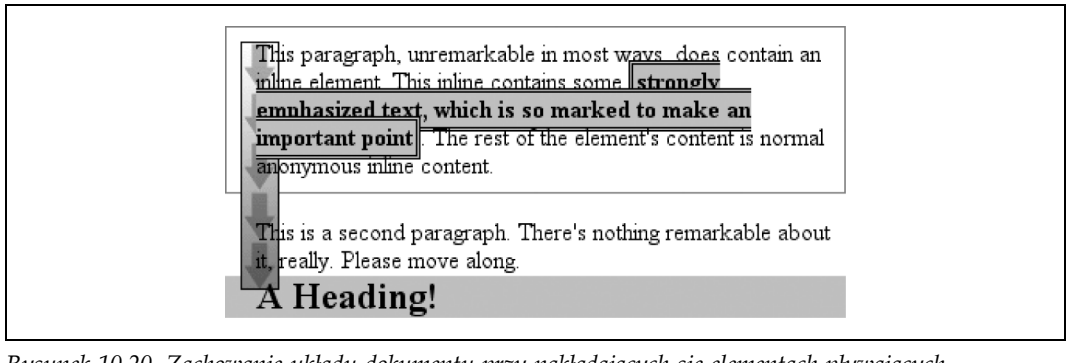
- Obramowanie, tło oraz zawartość pojemnika wewnętrznego, który nakłada się na element pływający, wyświetlane są „na wierzchu" elementu pływającego.
- Obramowanie oraz tło pojemnika blokowego, który nakłada się na element pływający, wyświetlane są „pod" elementem pływającym, natomiast jego zawartość wyświetlana jest „na wierzchu" tego elementu.

W celu zilustrowania tych reguł można rozważyć następującą sytuację:

```
<img src="testy.gif" class="sideline">
<p class="box">
   This paragraph, unremarkable in most ways, does contain an inline element. This
   inline contains some <strong>strongly emphasized text, which is so marked to make
   an important point</strong>.  The rest of the element's content is normal
   anonymous inline content.
</p>
<p>
   This is a second paragraph.  There's nothing remarkable about it, really. Please
   move along.
</p>
<h2 id="jump-up">A Heading!</h2>
```

Kiedy do powyższego kodu zastosuje się poniższe style, otrzyma się rezultat widoczny na rysunku 10.20:

```
img.sideline {float: left; margin: 10px -15px 10px 10px;}
p.box {border: 1px solid gray; padding: 0.5em;}
p.box strong {border: 3px double black; background: silver; padding: 2px;}
h2#jump-up {margin-top: -15px; background: silver;}
```



Rysunek 10.20. Zachowanie układu dokumentu przy nakładających się elementach pływających

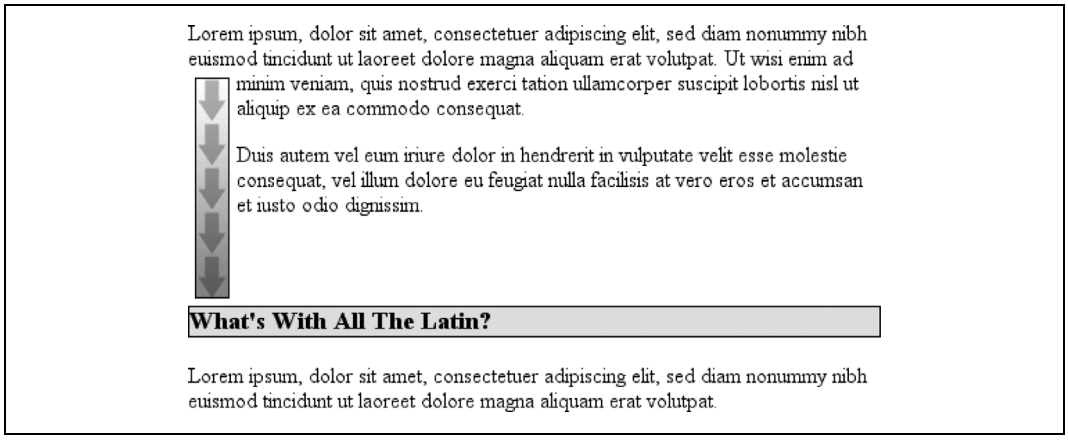
Element wewnętrzny (`strong`) całkowicie nakłada się na pływający obrazek — wraz z tłem, obramowaniem i zawartością. Z kolei w przypadku elementów blokowych na wierzchu elementu

pływającego wyświetlana jest jedynie zawartość. Ich tło oraz obramowanie umieszczane są pod elementem pływającym.

Opisane tutaj zachowanie polegające na nakładaniu się elementów jest niezależne od ich kolejności w źródle dokumentu. Nie ma znaczenia, czy element pojawia się przed czy po elemencie pływającym; zawsze pojawia się to samo zachowanie.

Właściwość clear

Pływanie zostało już omówione dość szczegółowo, jednak przed przejściem do pozycjonowania należy wspomnieć o jeszcze jednej kwestii. Nie zawsze będzie się chciało, by zawartość opływała element pływający — w niektórych przypadkach specjalnie będzie się chciało temu zapobiec. Jeśli mamy dokument podzielony na części, może się zdarzyć, że nie chcemy, by elementy pływające z jednej części nakładały się na kolejne części dokumentu. W takim przypadku trzeba by ustawić pierwszy element każdej z części w taki sposób, by zabronić elementom pływającym pojawiania się obok niego. Gdyby pierwszy element w jakiś sposób miał zostać umieszczony obok elementu pływającego, zostałby on zepchnięty na dół, do momentu aż pojawi się poniżej elementu pływającego, a cała następująca po nim treść pojawi się dopiero po nim, jak pokazano na rysunku 10.21.



Rysunek 10.21. Wyświetlanie elementu z zastosowaniem właściwości clear

Można to uzyskać dzięki właściwości `clear`.

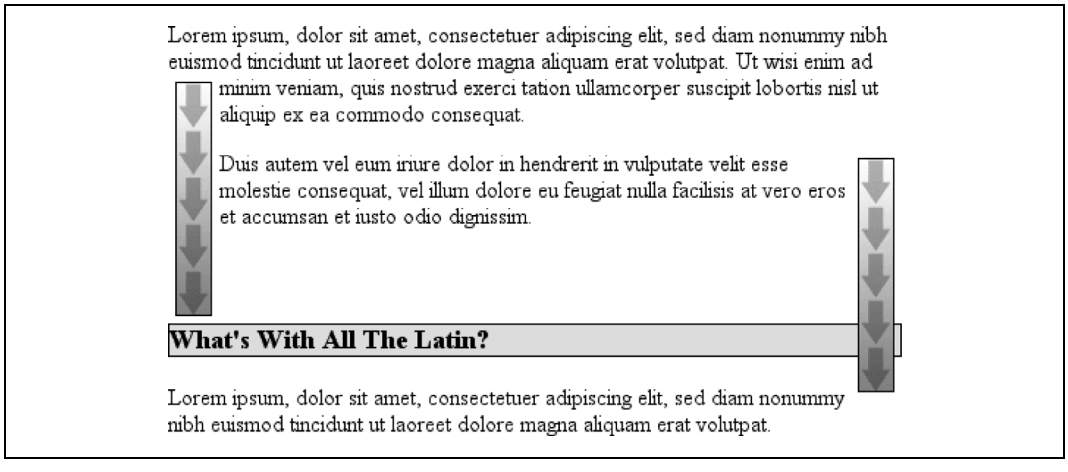
clear

Wartości:	`left` | `right` | `both` | `none` | `inherit`
Wartość początkowa:	`none`
Stosuje się do:	Elementów blokowych
Dziedziczona:	Nie
Wartość wyliczona:	Jak określono

Żeby na przykład upewnić się, iż żaden element h3 nie zostanie umieszczony na prawo od elementów pływających w lewo, należy zadeklarować h3 {clear: left;}. Regułę tę można przetłumaczyć w następujący sposób: „Upewnij się, że lewa strona elementu h3 jest wolna od pływających obrazków"; ma ona efekt podobny do konstrukcji HTML <br clear="left">. Ironicznym zbiegiem okoliczności jest to, że zachowaniem domyślnym w większości przeglądarek jest zmuszenie elementów br do generowania pojemników wewnętrznych, dlatego clear nie stosuje się do nich, dopóki nie zmieni się ich właściwości display! Poniższa reguła wykorzystuje właściwość clear w celu zapobieżenia opływaniu elementów pływających po lewej stronie:

```
h3 {clear: left;}
```

Choć reguła ta zepchnie elementy h3 poniżej elementów pływających w lewo, pozwoli na pojawienie się elementów pływających po prawej stronie elementów h3, jak widać na rysunku 10.22.



Rysunek 10.22. Właściwość clear ustawiona na left — po prawej stronie mogą pojawić się elementy pływające

By zapobiec tego rodzaju zachowaniu i upewnić się, że elementy h3 nie występują w jednej linii z żadnymi elementami pływającymi, należy skorzystać z wartości both:

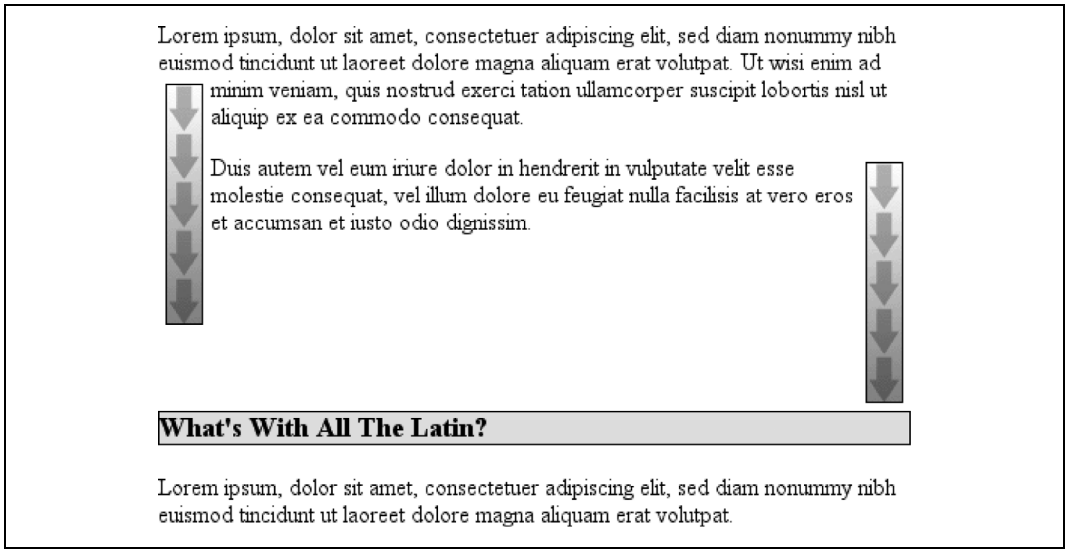
```
h3 {clear: both;}
```

Co jest zrozumiałe, wartość ta zapobiega pojawieniu się elementów pływających po obu stronach elementu h3, jak zademonstrowano na rysunku 10.23.

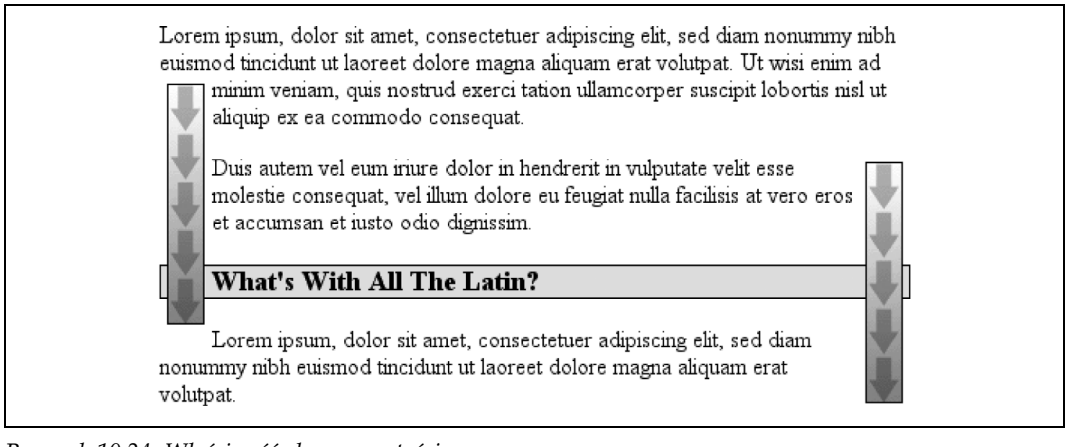
Gdyby wskazane było jedynie spychanie elementów h3 w dół przez elementy pływające znajdujące się po ich prawej stronie, należałoby zadeklarować h3 {clear: right;}.

Wreszcie istnieje wartość clear: none, która pozwala elementom na pływanie po obu stronach elementu. Tak jak w przypadku float: none, wartość ta istnieje głównie po to, by pozwolić na normalne zachowanie w dokumentach, w których elementy zezwalają na pojawienie się elementów pływających po obu ich stronach. Wartość none może oczywiście zostać wykorzystana do nadpisania innych stylów, jak widać na rysunku 10.24. Pomimo istnienia obowiązującej w całym dokumencie reguły zabraniającej pojawienia się elementów pływających po obu stronach nagłówka h3, jeden z takich nagłówków został ustawiony w taki sposób, by na takie zachowanie zezwolić:

```
h3 {clear: both;}
<h3 style="clear: none;">What's With All The Latin?</h3>
```



Rysunek 10.23. Właściwość clear ustawiona na both



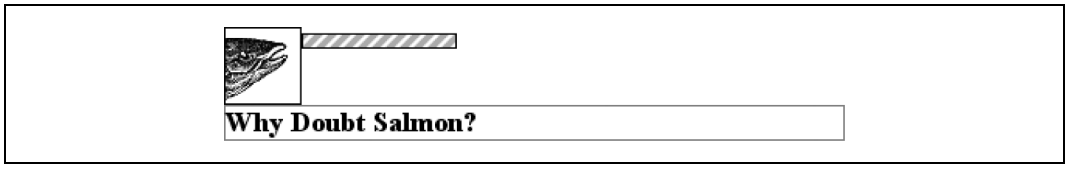
Rysunek 10.24. Właściwość clear o wartości none

W CSS1 oraz CSS2 właściwość `clear` działała poprzez zwiększenie marginesu górnego elementu w taki sposób, by element ten znalazł się poniżej elementu pływającego, w praktyce ignorując szerokość marginesu ustawioną dla góry takiego elementu. W ten sposób zamiast na przykład zadeklarowanej wartości `1.5em`, szerokość marginesu mogła być zwiększona do `10em`, `25px` czy `7.133in` — czy do innej wielkości, która była wymagana do przesunięcia elementu w dół na tyle, by jego obszar zawartości znalazł się poniżej dolnego brzegu elementu pływającego.

W CSS2.1 wprowadzono pojęcie prześwitu (ang. *clearance*). *Prześwit* jest dodatkowym odstępem dodawanym ponad marginesem górnym elementu, który ma na celu zepchnięcie go poniżej wszystkich elementów pływających. Oznacza to, że margines górny takiego elementu w tej sytuacji nie zmienia się. Przesunięcie elementu w dół jest natomiast spowodowane przez prześwit. Warto zwrócić uwagę na umieszczenie obramowania nagłówka na rysunku 10.25, które wynika z następujących reguł:

```
img.sider {float: left; margin: 0;}
h3 {border: 1px solid gray; clear: left; margin-top: 15px;}

<img src="boxer.gif" class="sider" height="50" width="50">
<img src="stripe.gif" height="10" width="100">
<h3>Why Doubt Salmon?</h3>
```



Rysunek 10.25. Właściwość clear i jej wpływ na marginesy

Obramowanie górne nagłówka `h3` oraz obramowanie dolne pływającego obrazka nie są od siebie oddzielone, ponieważ prześwit o wielkości dwudziestu pięciu pikseli został dodany powyżej marginesu górnego o wartości piętnastu pikseli w celu zepchnięcia krawędzi obramowania górnego elementu `h3` tuż poniżej dolnego brzegu elementu pływającego. Będzie tak, dopóki margines górny `h3` nie będzie miał wielkości czterdziestu pikseli bądź więcej — w tej sytuacji `h3` w sposób naturalny zostanie umieszczony poniżej elementu pływającego, a wartość `clear` będzie nieistotna.

W większości przypadków nie wiadomo jednak, jak bardzo przesunięty musi zostać element. Sposobem na zapewnienie tego, by element taki miał jakiś odstęp pomiędzy swoim górnym brzegiem a dołem elementu pływającego, jest dodanie marginesu dolnego do samego elementu pływającego. Jeśli zatem chce się uzyskać przestrzeń o wysokości przynajmniej piętnastu pikseli poniżej elementu pływającego z poprzedniego przykładu, należy zmienić arkusz stylów CSS w następujący sposób:

```
img.sider {float: left; margin: 0 0 15px;}
h3 {border: 1px solid gray; clear: left;}
```

Margines dolny elementu pływającego zwiększa rozmiar jego pojemnika, a tym samym również punkt, poniżej którego zepchnięte muszą zostać przesuwane w dół elementy. Dzieje się tak oczywiście dlatego, że — jak widać było wcześniej — brzegi marginesów elementu pływającego definiują krawędzie pojemnika tego elementu.

Pozycjonowanie

Idea przyświecająca pozycjonowaniu jest względnie prosta. Pozycjonowanie pozwala dokładnie określić, w którym miejscu powinny znajdować się pojemniki elementów w stosunku do miejsca, w jakim normalnie by się pojawiły — lub w stosunku do elementu rodzica, innego elementu albo nawet względem samego okna przeglądarki.

Podstawowe koncepcje

Przed zagłębieniem się w różne rodzaje pozycjonowania dobrym pomysłem będzie przyjrzenie się istniejącym typom pozycjonowania oraz różnicom pomiędzy nimi. Konieczne będzie również zdefiniowanie pewnych podstawowych koncepcji, które są kluczowe dla zrozumienia sposobu działania pozycjonowania.

Typy pozycjonowania

Poprzez użycie właściwości `position` można wybrać jeden z czterech różnych typów pozycjonowania, które wpływają na sposób generowania pojemnika elementu.

position

Wartości:	`static` | `relative` | `absolute` | `fixed` | `inherit`
Wartość początkowa:	`static`
Stosuje się do:	Wszystkich elementów
Dziedziczona:	Nie
Wartość wyliczona:	Jak określono

Wartości właściwości `position` mają następujące znaczenia:

`static`
: Pojemnik elementu tworzony jest w zwykły sposób. Elementy blokowe tworzą pojemnik prostokątny, który jest częścią układu dokumentu, natomiast elementy wewnętrzne powodują utworzenie jednego bądź większej liczby pojemników wiersza, które są umieszczane wewnątrz ich elementów rodzica.

`relative`
: Pojemnik elementu zostaje przesunięty o pewną odległość. Element zachowuje taki kształt, jaki by miał, gdyby nie był pozycjonowany, a ponadto zachowana zostaje przestrzeń, którą normalnie zajmowałby element.

`absolute`
: Pojemnik elementu jest całkowicie usuwany z układu dokumentu i jest pozycjonowany w stosunku do bloku go zawierającego, którym może być inny element w dokumencie bądź też początkowy blok zawierający (opisany w kolejnym podrozdziale). Usuwana jest wszelka przestrzeń zajmowana normalnie przez ten element, tak jakby element nie istniał. Każdy element pozycjonowany bezwzględnie generuje pojemnik blokowy bez względu na to, jaki typ pojemnika by wygenerował, gdyby pozostał w układzie normalnym dokumentu.

`fixed`
: Pojemnik elementu jest pozycjonowany tak, jakby był on pozycjonowany bezwzględnie, ale blokiem go zawierającym jest cały widoczny obszar dokumentu.

Nie należy się teraz przejmować szczegółami, gdyż każdy rodzaj pozycjonowania omówiony zostanie w dalszej części rozdziału. Najpierw jednak należy przyjrzeć się blokom zawierającym elementy.

Blok zawierający element

Bloki zawierające elementy omówione zostały już wcześniej w kontekście elementów pływających. W takim przypadku blokiem zawierającym element pływający był najbliższy element blokowy będący przodkiem elementu. W przypadku pozycjonowania sytuacja nie jest aż tak prosta. W CSS2.1 zdefiniowano następujące zasady:

- Blok zawierający element główny (zwany także *początkowym blokiem zawierającym*) tworzony jest przez przeglądarkę. W języku HTML elementem głównym jest element `html`, chociaż niektóre przeglądarki uważają za element główny element `body`. W większości przeglądarek początkowym blokiem zawierającym jest prostokąt o wielkości widocznego obszaru dokumentu.
- Dla elementów niebędących elementem głównym, których wartość właściwości `position` równa jest `relative` bądź `static`, blok je zawierający tworzony jest przez brzeg obszaru zawartości najbliższego pojemnika przodka, który jest elementem blokowym, komórką tabeli bądź elementem typu `inline-block`.
- Dla elementów niebędących elementem głównym, których wartość właściwości `position` równa jest `absolute`, blokiem je zawierającym jest najbliższy przodek (dowolnego rodzaju), który ma wartość właściwości `position` inną niż `static`. Odbywa się to w następujący sposób:
  - Jeśli przodkiem jest element blokowy, to blokiem zawierającym zostaje brzeg dopełnienia tego elementu (innymi słowy — obszar, który zostałby ograniczony obramowaniem).
  - Jeśli przodkiem jest element wewnętrzny, to blokiem zawierającym staje się brzeg obszaru zawartości tego elementu. W językach pisanych od lewa do prawa górny oraz lewy brzeg bloku zawierającego są górnym i lewym brzegiem zawartości pierwszego pojemnika elementu przodka, a dolny oraz prawy brzeg są jednocześnie dolnym i prawym brzegiem zawartości ostatniego pojemnika przodka. W językach pisanych od prawej do lewej prawy brzeg bloku zawierającego pokrywa się z prawym brzegiem zawartości pierwszego pojemnika, a lewy brzeg pobierany jest z ostatniego pojemnika. Góra i dół pozostają takie same.
  - Jeśli nie występuje element będący przodkiem, to do ustalenia bloku zawierającego wykorzystywany jest brzeg początkowego bloku zawierającego.

Ważną informacją dotyczącą bloków zawierających elementy jest fakt, że elementy mogą być pozycjonowane na zewnątrz bloku je zawierającego. Przypomina to sposób, w którym przy użyciu marginesów ujemnych można umieścić elementy pływające poza obszarem zawartości rodzica. Z tego powodu może się wydawać, iż określenie „blok zawierający" powinno raczej nazywać się „kontekst pozycjonowania". Ponieważ jednak specyfikacja używa wyrażenia „blok zawierający", również w niniejszej książce zostanie on tak nazwany (naprawdę staram się zminimalizować zamieszanie!).

Właściwości przesunięcia

Trzy z przedstawionych w poprzednim podrozdziale typów pozycjonowania — `relative`, `absolute` oraz `fixed` — wykorzystują cztery różne właściwości w celu opisania przesunięcia brzegów elementu pozycjonowanego względem brzegów bloku go zawierającego. Te cztery właściwości, które nazywa się *właściwościami przesunięcia* (ang. *offset properties*), składają się na spory zakres tego, co umożliwia działanie pozycjonowania (ramka z opisem znajduje się na następnej stronie).

Właściwości te opisują przesunięcie od najbliższego boku bloku zawierającego element (stąd określenie *przesunięcie*). Na przykład właściwość `top` opisuje odległość brzegu marginesu górnego pozycjonowanego elementu od górnej krawędzi bloku go zawierającego. W przypadku `top` wartości dodatnie przesuwają krawędź marginesu górnego pozycjonowanego elementu *w dół*, podczas gdy wartości ujemne przesuwają go *ponad* górny brzeg bloku go

top, right, bottom, left

Wartości:	`<length>` | `<percentage>` | `auto` | `inherit`
Wartość początkowa:	`auto`
Stosuje się do:	Elementów pozycjonowanych (czyli elementów, w których właściwość `position` ma wartość inną od `static`)
Dziedziczona:	Nie
Wartość procentowa:	Odnosi się do wysokości bloku zawierającego element dla `top` oraz `bottom` i do szerokości bloku zawierającego dla `right` oraz `left`
Wartość wyliczona:	Dla elementów pozycjonowanych względnie — należy zobaczyć część „Uwaga"; dla elementów o wartości `static` — `auto`; dla wartości długości — odpowiednia długość bezwzględna; dla wartości procentowych — określona wartość. W innym przypadku — `auto`
Uwaga:	Wartości wyliczone uzależnione są od różnych czynników; więcej przykładów na ten temat we wpisach z Dodatku A

zawierającego. Podobnie `left` opisuje, jak daleko z prawej (przy wartościach dodatnich) lub z lewej (przy wartościach ujemnych) strony lewego brzegu bloku zawierającego znajdzie się brzeg marginesu lewego pozycjonowanego elementu. Wartości dodatnie powodują przesunięcie krawędzi marginesu elementu pozycjonowanego w prawo, natomiast wartości ujemne — w lewo.

Działanie wszystkich czterech właściwości można przedstawić tak: dodatnie wartości powodują przesunięcie do wnętrza, przesuwając brzegi w kierunku środka bloku zawierającego element, natomiast wartości ujemne powodują odsunięcie brzegu na zewnątrz.

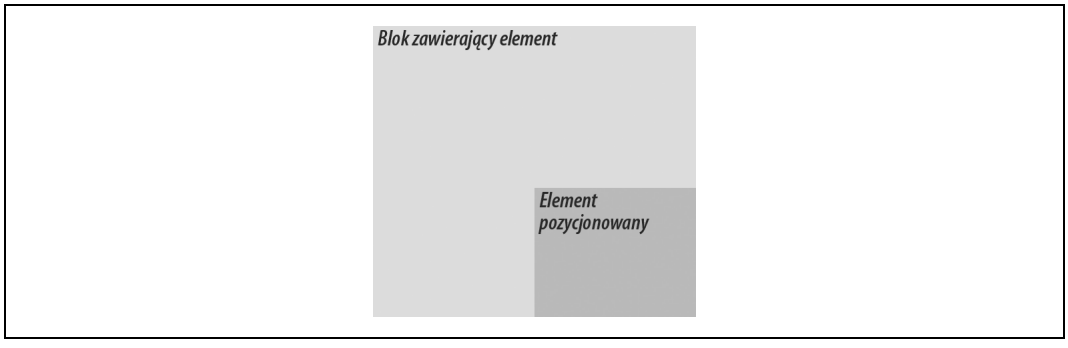
Oryginalna specyfikacja CSS2 w rzeczywistości mówiła, że odsuwane są brzegi obszaru zawartości elementu, a nie krawędzie marginesów, jednak było to niespójne z innymi częściami CSS2. Błąd ten został poprawiony w opublikowanej później erracie oraz w CSS2.1. Brzegi marginesów wykorzystywane są do obliczania przesunięcia we wszystkich obecnie rozwijanych przeglądarkach (w momencie pisania niniejszej książki).

Wynikiem przesunięcia krawędzi marginesów pozycjonowanego elementu jest to, że w procesie pozycjonowania przesuwany jest cały element — marginesy, obramowanie, dopełnienie oraz zawartość. Innymi słowy — elementowi pozycjonowanemu można dodać marginesy, obramowanie oraz dopełnienie. Zostaną one zachowane i utrzymane wraz z pozycjonowanym elementem oraz będą zawarte wewnątrz obszaru zdefiniowanego przez właściwości przesunięcia.

Należy pamiętać, że właściwości przesunięcia definiują je od analogicznego brzegu bloku zawierającego element (na przykład `left` określa przesunięcie od lewego brzegu), a nie od lewego górnego rogu bloku zawierającego. Dlatego na przykład jednym ze sposobów na wypełnienie prawego dolnego rogu bloku zawierającego element jest użycie następujących wartości:

```
top: 50%; bottom: 0; left: 50%; right: 0;
```

W analizowanym przykładzie brzeg marginesu lewego pozycjonowanego elementu umieszczony jest w połowie szerokości bloku go zawierającego. Jest to przesunięcie od lewego brzegu bloku zawierającego. Brzeg marginesu prawego elementu pozycjonowanego nie jest jednak przesunięty od prawego brzegu bloku zawierającego element, a więc oba brzegi są zbieżne. Podobne rozumowanie sprawdza się dla górnego i dolnego brzegu elementu pozycjonowanego — brzeg marginesu górnego umieszczony jest w połowie wysokości bloku zawierającego element, ale brzeg marginesu dolnego nie jest przesunięty od dolnego brzegu bloku zawierającego. Prowadzi to do efektu pokazanego na rysunku 10.26.



Rysunek 10.26. Wypełnienie prawej dolnej ćwiartki bloku zawierającego element

To, co przedstawione jest na rysunku 10.26, jak również w większości przykładów z niniejszego rozdziału, oparte jest na pozycjonowaniu bezwzględnym. Ponieważ pozycjonowanie bezwzględne jest najprostszym typem, na podstawie którego można zademonstrować działanie właściwości `top`, `right`, `bottom` oraz `left`, na razie przykłady ograniczą się do niego.

Warto zwrócić uwagę, że element pozycjonowany ma nieco inny kolor tła. Patrząc na rysunek 10.26, można zauważyć, że pozycjonowany element nie posiada marginesów. Gdyby jednak je posiadał, to pomiędzy obramowaniem a brzegami przesunięcia pojawiłby się pasek pustej przestrzeni. Wyglądałoby to tak, jakby element nie wypełniał dokładnie prawej dolnej ćwiartki bloku go zawierającego. W rzeczywistości wypełniałby ją, ale nie byłoby to natychmiast widoczne dla oka obserwatora. Innymi słowy — poniższe dwa zestawy deklaracji stylów dałyby mniej więcej taki sam efekt wizualny przy założeniu, że blok zawierający element ma `100em` wysokości i `100em` szerokości:

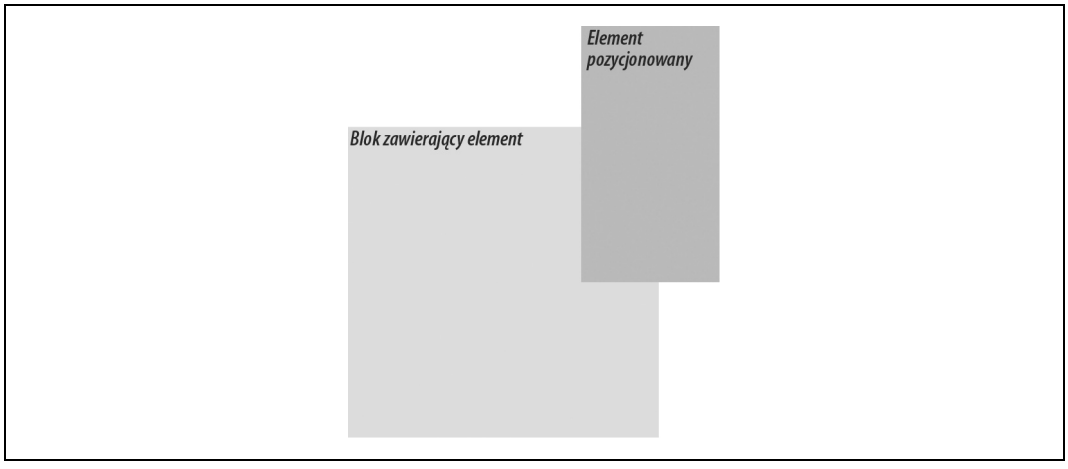
```
top: 50%; bottom: 0; left: 50%; right: 0; margin: 10em;
top: 60%; bottom: 10%; left: 60%; right: 10%; margin: 0;
```

Podobieństwo będzie znowu tylko wizualne.

Element można umieścić na zewnątrz bloku go zawierającego poprzez użycie wartości ujemnych. Na przykład poniższe wartości doprowadzą do rezultatów zaprezentowanych na rysunku 10.27:

```
top: -5em; bottom: 50%; left: 75%; right: -3em;
```

Poza wartościami długości oraz wartościami procentowymi właściwości przesunięcia mogą również zostać ustawione na `auto`, co jest wartością domyślną. Wartość ta nie zachowuje się jednak tak samo w każdym przypadku; zmienia się to w zależności od wykorzystywanego typu pozycjonowania. Sposób działania wartości `auto` zostanie omówiony w dalszej części rozdziału, wraz z omawianiem kolejnych typów pozycjonowania.



Rysunek 10.27. Pozycjonowanie elementu poza blokiem go zawierającym

Szerokość oraz wysokość

Zapewne w wielu przypadkach (po określeniu tego, gdzie element ma zostać umieszczony) będziemy chcieli pójść jeszcze dalej i zadeklarować wysokość oraz szerokość tego elementu. Prawdopodobnie dodatkowo pojawią się okoliczności, w których będziemy chcieli ograniczyć to, jak wysoki lub szeroki może stać się element pozycjonowany, nie wspominając już o przypadkach, w których pragnie się, aby przeglądarka automatycznie obliczyła szerokość lub wysokość, a może nawet obydwie wartości.

Ustawianie szerokości oraz wysokości

Jeśli chce się nadać elementom pozycjonowanym określoną szerokość, to oczywiste jest, że najlepszą właściwością, którą należy wykorzystać, będzie właściwość `width`. Podobnie właściwość `height` pozwoli zadeklarować dokładną wysokość pozycjonowanego elementu.

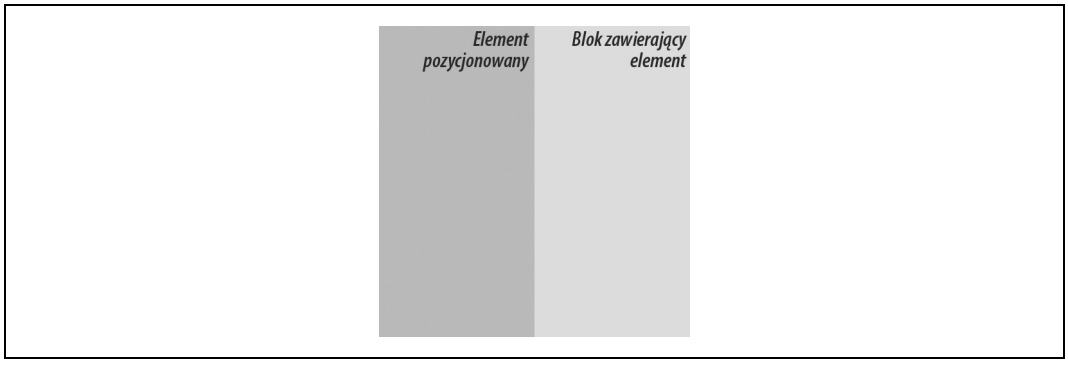
Mimo że czasami ważne jest ustawienie szerokości i wysokości elementu, przy pozycjonowaniu elementów nie zawsze jest to konieczne. Jeśli na przykład umieszczenie wszystkich czterech boków opisane jest przy użyciu właściwości `top`, `right`, `bottom` oraz `left`, to wysokość i szerokość elementu wynika w sposób niejawny z rozmieszczenia jego boków. Zakładając, że chce się, by element pozycjonowany bezwzględnie wypełnił od góry do dołu lewą połowę bloku go zawierającego, można użyć poniższych stylów, aby otrzymać rezultat przedstawiony na rysunku 10.28:

```
top: 0; bottom: 0; left: 0; right: 50%;
```

Ponieważ domyślna wartość właściwości `width` i `height` wynosi `auto`, rezultat przestawiony na rysunku 10.28 jest dokładnie taki sam, jaki by był, gdyby użyto stylów wypisanych poniżej:

```
top: 0; bottom: 0; left: 0; right: 50%; width: 50%; height: 100%;
```

Obecność `width` oraz `height` w powyższym przykładzie nie dodaje nic do umieszczenia elementu.

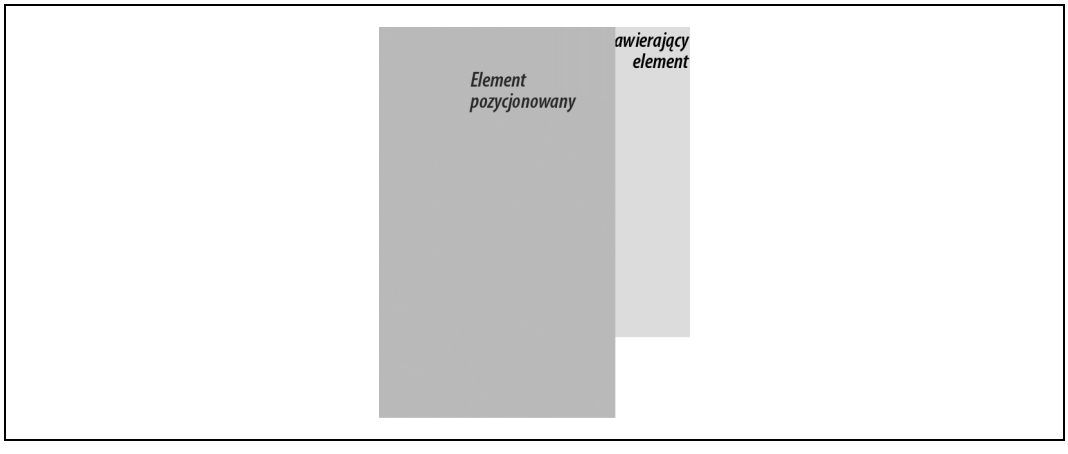


Rysunek 10.28. Ustawienie pozycjonowania oraz rozmiaru elementu za pomocą samych właściwości przesunięcia

Oczywiście gdyby należało dodać do elementu dopełnienie, obramowanie bądź marginesy, obecność jawnie zadeklarowanych wartości właściwości `height` oraz `width` dużo by zmieniła:

```
top: 0; bottom: 0; left: 0; right: 50%; width: 50%; height: 100%; padding: 2em;
```

Dzięki takiej deklaracji otrzyma się element pozycjonowany, który rozszerza się poza blok go zawierający, jak widać na rysunku 10.29:



Rysunek 10.29. Pozycjonowanie elementu częściowo poza blokiem go zawierającym

Dzieje się tak, ponieważ — jak widać było w poprzednich rozdziałach — dopełnienie dodawane jest do obszaru zawartości elementu, natomiast rozmiar obszaru zawartości elementu zależy od wartości właściwości `height` oraz `width`. Aby otrzymać pożądane dopełnienie i mimo to zmieścić element w bloku go zawierającym, należy albo usunąć deklaracje `height` oraz `width`, albo w jawny sposób ustawić je na `auto`.

Ograniczanie szerokości oraz wysokości

Jeśli będzie to konieczne lub pożądane, można ustanowić limit szerokości lub wysokości elementu, używając następujących właściwości CSS2, które będą w niniejszej książce określane mianem *właściwości min-max*. Minimalny rozmiar obszaru zawartości elementu można zdefiniować za pomocą właściwości `min-width` oraz `min-height`.

min-width, min-height

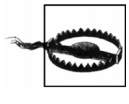
Wartości:	*<length>* | *<percentage>* | inherit
Wartość początkowa:	0
Stosuje się do:	Wszystkich elementów z wyjątkiem niezastępowanych elementów wewnętrznych oraz elementów tabeli
Dziedziczona:	Nie
Wartość procentowa:	Odnosi się do wysokości (min-height) oraz szerokości (min-width) bloku zawierającego element
Wartość wyliczona:	Dla wartości procentowych — jak określono; dla wartości długości — długość bezwzględna; w innym przypadku — none

W podobny sposób rozmiar elementu może być ograniczany dzięki właściwościom max-width oraz max-height.

max-width, max-height

Wartości:	*<length>* | *<percentage>* | none | inherit
Wartość początkowa:	none
Stosuje się do:	Wszystkich elementów z wyjątkiem niezastępowanych elementów wewnętrznych oraz elementów tabeli
Dziedziczona:	Nie
Wartość procentowa:	Odnosi się do wysokości (max-height) oraz szerokości (max-width) bloku zawierającego element
Wartość wyliczona:	Dla wartości procentowych — jak określono; dla wartości długości — długość bezwzględna; w innym przypadku — none

Nazwy tych właściwości mówią same za siebie. Co może na pierwszy rzut oka wydawać się nieco mniej oczywiste, jednak po chwili staje się jasne, to to, że żadna z wartości tych właściwości nie może być ujemna.



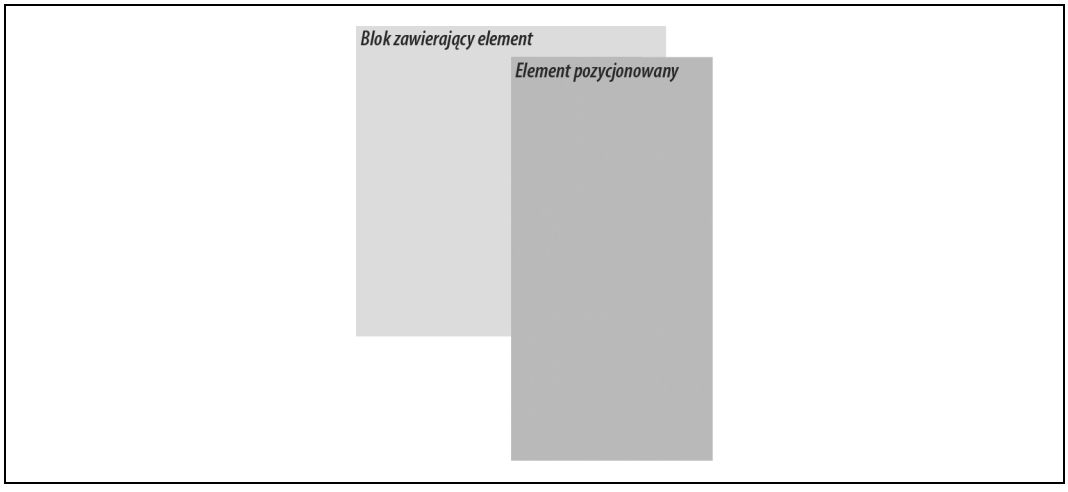
Właściwości min-height, min-width, max-height oraz max-width nie były obsługiwane przez przeglądarkę Internet Explorer dla systemu Windows aż do wersji IE7.

Poniższe style wymuszą, by element pozycjonowany miał przynajmniej 10em szerokości i 20em wysokości, jak widać na rysunku 10.30:

```
top: 10%; bottom: 20%; left: 50%; right: 10%; min-width: 10em; min-height: 20em;
```

Nie jest to zbyt ciekawe rozwiązanie, ponieważ zmusza ono element do tego, by miał co najmniej pewien rozmiar bez względu na wielkość bloku go zawierającego. Poniżej znajduje się lepsze rozwiązanie:

```
top: 10%; bottom: auto; left: 50%; right: 10%; height: auto; min-width: 15em;
```



Rysunek 10.30. Ustawienie minimalnej wysokości oraz szerokości dla elementu pozycjonowanego

Oto przypadek, w którym element powinien mieć przynajmniej `40%` szerokości bloku go zawierającego, ale nigdy nie może mieć jej mniej niż `15em`. Zmieniono również wartości `bottom` i `height` tak, by były one ustalane automatycznie. Takie ustawienia pozwolą na to, by element był tak wysoki, jak jest to konieczne do wyświetlenia zawartości (bez względu na to, jak wąski się stanie; oczywiście nie może być węższy niż `15em`).

W kolejnym podrozdziale omówiona zostanie rola, jaką w wysokości oraz szerokości elementu pozycjonowanego odgrywa wartość `auto`.

Można jeszcze odwrócić sytuację i zabezpieczyć element przed zbyt dużym rozciągnięciem wszerz lub wzdłuż, używając do tego kolejnych dwóch właściwości: `max-width` oraz `max-height`. Zastanówmy się nad sytuacją, w której z pewnych powodów chce się, by szerokość elementu wynosiła trzy czwarte szerokości bloku go zawierającego, ale nie stała się większa niż czterysta pikseli. Style odpowiednie do osiągnięcia tego efektu są następujące:

```
left: 0%; right: auto; width: 75%; max-width: 400px;
```

Ogromną zaletą właściwości min-max jest to, że pozwalają one względnie bezpieczne łączyć różne jednostki. Można ustawiać rozmiary oparte na wartościach procentowych, używając jednocześnie ograniczeń bazujących na miarach długości i odwrotnie.

Warto wspomnieć, że właściwości min-max mogą być bardzo użyteczne również w połączeniu z elementami pływającymi. Można na przykład pozwolić, aby szerokość elementu pływającego była określona względem szerokości elementu rodzica (który jest blokiem go zawierającym), jednocześnie zapewniając, że szerokość elementu pływającego nigdy nie spadnie poniżej `10em`. Odwrotne podejście także jest możliwe:

```
p.aside {float: left; width: 40em; max-width: 40%;}
```

Ustawi to szerokość elementu pływającego na `40em`, chyba że byłoby to więcej niż `40%` szerokości bloku zawierającego element — w takim przypadku element pływający zostanie zwężony.

Do zagadnienia nadawania rozmiarów elementom powrócimy przy omawianiu poszczególnych rodzajów pozycjonowania.

Wypływanie oraz przycinanie zawartości

Jeśli zawartość elementu będzie zbyt duża w stosunku do jego wielkości, może to zagrozić wypłynięciem jej z elementu. W takich sytuacjach istnieje kilka rozwiązań, a CSS2 pozwala autorom wybrać jedno z nich. Pozwala także określić obszar elementu, na zewnątrz którego zaczną działać właściwości przycinania i wypływania.

Wypływanie

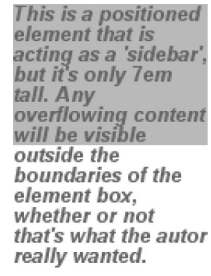
Powiedzmy, że mamy element, który z jakiegoś powodu został ograniczony do pewnej wielkości, a jego zawartość nie mieści się w określonym obszarze. Można przejąć kontrolę nad taką sytuacją, stosując właściwość `overflow`.

overflow

Wartości:	`visible | hidden | scroll | auto | inherit`
Wartość początkowa:	`Visible`
Stosuje się do:	Elementów blokowych oraz zastępowanych
Dziedziczona:	Nie
Wartość wyliczona:	Jak określono

Domyślna wartość `visible` oznacza, że zawartość elementu może być widoczna na zewnątrz jego pojemnika. Zazwyczaj prowadzi to do wylania się zawartości na zewnątrz własnego pojemnika elementu, ale nie zmienia kształtu pojemnika. Poniższe style mają efekt taki, jak przedstawiono na rysunku 10.31:

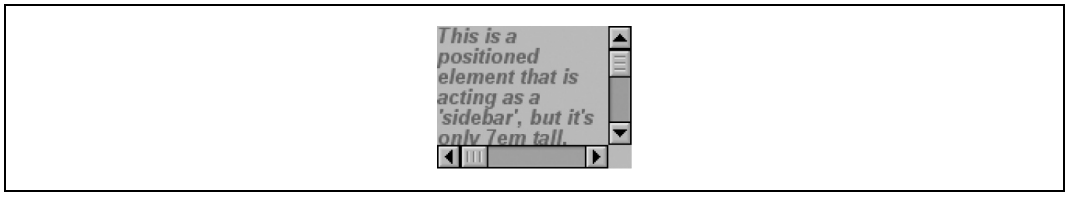
```
div#sidebar {position: absolute; top: 0; left: 0; width: 25%; height: 7em; background:
#BBB; overflow: visible;}
```



Rysunek 10.31. Zawartość elementu w sposób widoczny wypływająca poza pojemnik elementu

Jeśli właściwość `overflow` ma wartość `scroll`, zawartość elementu jest przycinana (to znaczy jej część jest niewidoczna), ale zapewniony jest sposób na to, aby użytkownik miał dostęp do niewidocznej zawartości. W przeglądarce będzie to oznaczało zastosowanie suwaka (bądź zestawu suwaków) lub innej metody dostępu do zawartości bez zmiany kształtu samego elementu. Jedna z możliwości, które mogłyby wyniknąć z poniższych stylów, pokazana została na rysunku 10.32:

```
div#sidebar {position: absolute; top: 0; left: 0; width: 15%; height: 7em; overflow:
scroll;}
```



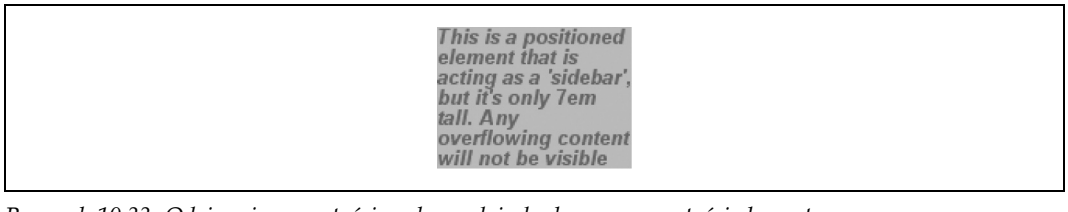
Rysunek 10.32. Wypływająca zawartość dostępna jest dzięki mechanizmowi przewijania

Jeśli użyto wartości `scroll`, to zawsze powinien być wyświetlony mechanizm przewijający (na przykład suwaki). Cytując specyfikację, „pozwala to uniknąć problemów związanych z pojawianiem się lub znikaniem suwaków w środowisku dynamicznym". W ten sposób nawet jeśli element posiada dostateczną ilość miejsca na wyświetlenie swojej zawartości, to suwaki nadal będą widoczne. Ponadto przy drukowaniu strony lub wyświetlaniu dokumentu w innym medium drukowanym zawartość powinna być wyświetlona tak, jakby zadeklarowana była wartość `visible`.

Kiedy wartość właściwości `overflow` ustawiona jest na `hidden`, zawartość elementu jest odcinana na krawędziach pojemnika elementu, ale nie zostanie dostarczony żaden interfejs przewijania, który udostępniłby zawartość znajdującą się poza obszarem przycinania użytkownikowi. Można rozważyć poniższe style:

```
div#sidebar {position: absolute; top: 0; left: 0; width: 15%; height: 7em; overflow:
hidden;}
```

W takim przypadku przycięta zawartość nie będzie dostępna dla użytkownika. Doprowadziłoby to do sytuacji przedstawionej na rysunku 10.33.



Rysunek 10.33. Odcinanie zawartości na krawędziach obszaru zawartości elementu

Istnieje jeszcze deklaracja `overflow: auto`, która pozostawia przeglądarce ustalenie tego, jakie zachowanie będzie najkorzystniejsze. Zaleca się, aby przeglądarka zastosowała mechanizm przewijania wtedy, gdy jest to konieczne. Jest to potencjalnie przydatny sposób użycia właściwości `overflow`, ponieważ przeglądarka może interpretować go jako „dostarcz suwaki jedynie wtedy, gdy będą potrzebne" (mogą nie być potrzebne, ale zapewne mogłyby być, a prawdopodobnie powinny być zastosowane).

Przycinanie zawartości elementu

W sytuacjach, kiedy zawartość elementu pozycjonowanego bezwzględnie wypływa poza jego pojemnik, a wartość właściwości `overflow` jest taka, że zawartość powinna zostać przycięta, można zmienić kształt przycinanego obszaru przy użyciu właściwości `clip`.

clip	
Wartości:	`rect(`*`top`*`, `*`right`*`, `*`bottom`*`, `*`left`*`)` | `auto` | `inherit`
Wartość początkowa:	`auto`
Stosuje się do:	Elementów pozycjonowanych bezwzględnie (w CSS2 `clip` stosuje się do elementów blokowych oraz zastępowanych)
Dziedziczona:	Nie
Wartość wyliczona:	Dla prostokąta — zbiór czterech wyliczonych długości reprezentujących boki prostokąta odcinającego; w innym przypadku — jak określono

Wartość domyślna `auto` oznacza, że zawartość elementu nie powinna być przycinana. Inną możliwością jest zdefiniowanie przycinanego kształtu względem obszaru zawartości elementu. Nie zmienia to kształtu obszaru zawartości elementu, ale zmienia obszar, w którym może być wyświetlona zawartość elementu.

W CSS2 jedynym dostępnym kształtem obszaru przycinania jest prostokąt, specyfikacja uwzględnia jednak możliwość włączenia innych kształtów w przyszłych wersjach.

Jest to osiągane przy użyciu wartości kształtu `rect(`*`top`*`, `*`right`*`, `*`bottom`*`, `*`left`*`)`. Można określić brak zmiany przycinanego obszaru w następujący sposób:

```
clip: rect(0, auto, auto, 0);
```

Składnia wartości `rect` jest interesująca. Z technicznego punktu widzenia może to być `rect(`*`top`*`, `*`right`*`, `*`bottom`*`, `*`left`*`)` — należy zwrócić uwagę na użycie przecinków — jednak należy pamiętać o tym, że specyfikacja CSS2 zawiera także przykłady zapisu bez przecinków oraz definiuje wartość `rect` w taki sposób, że może one przyjmować obydwa zapisy. Tekst niniejszej książki posługuje się głównie zapisem wykorzystującym przecinki, ponieważ jest on bardziej czytelny, a także preferowany w CSS2.1.

Szczególnie ważne jest podkreślenie informacji, że wartości podane w `rect(...)` *nie są* wartościami odsunięcia boków. Są one natomiast odległością mierzoną od lewego górnego rogu elementu (lub też od prawego górnego rogu w przypadku języków pisanych od prawa do lewa). Dlatego przycinający prostokąt mieszczący kwadrat o wymiarach dwadzieścia na dwadzieścia pikseli zawarty w lewym górnym rogu elementu zostanie zdefiniowany w następująco:

```
rect(0, 20px, 20px, 0)
```

Jedyne wartości dozwolone przy określaniu przycinanego obszaru w `rect(...)` to wartości długości oraz wartość `auto`, która oznacza to samo co ustawienie przycinanego brzegu na odpowiedni brzeg obszaru zawartości elementu. Poniższe dwie deklaracje mają zatem takie samo działanie:

```
clip: rect(auto, auto, 10px, 1em);
clip: rect(0, 0, 10px, 1em);
```

Ponieważ każde przycinanie w `clip` odbywa się od lewego górnego rogu, a wartości procentowe nie są dozwolone, jest praktycznie niemożliwe, by utworzyć wyśrodkowany obszar przycinania, o ile nie zna się wymiarów samego elementu. Można rozważyć następującą sytuację:

```
div#sidebar {position: absolute; top: 0; bottom: 50%; right: 50%; left: 0; clip:
rect(1em,4em,6em,1em);}
```

Ponieważ nie istnieje żaden sposób, by dowiedzieć się, ile em szerokości czy wysokości będzie miał element, nie ma również możliwości zdefiniowania prostokąta przycinającego — który kończy się jeden em na prawo lub jeden em poniżej obszaru zawartości elementu. Jedynym sposobem, by się tego dowiedzieć jest ustawienie wysokości oraz szerokości samego elementu:

```
div#sidebar {position: absolute; top: 0; left: 0; width: 5em; height: 7em; clip:
rect(1em,4em,6em,1em);}
```

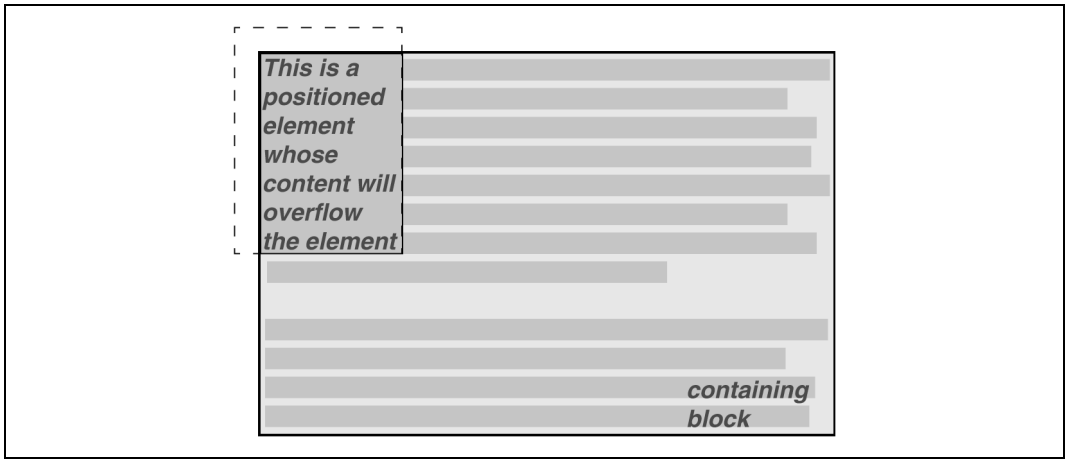
Spowoduje to coś podobnego do sytuacji zaprezentowanej na rysunku 10.34, w której linia przerywana została dodana w celu zilustrowania obszaru przycinającego. Linia ta w rzeczywistości nie pokaże się w przeglądarce próbującej wyświetlić element.

Rysunek 10.34. Ustawienie obszaru przycinającego dla wypływającej zawartości

Możliwe jest jednak używanie wartości ujemnych, które rozciągną przycinający obszar poza pojemnik elementu. Jeśli chce się wypchnąć obszar przycinający o jedną czwartą cala w górę i w lewo, to można zrobić to przy użyciu poniższej deklaracji (przedstawiono to na rysunku 10.35):

```
clip: rect(-0.25in, auto, auto, -0.25in);
```

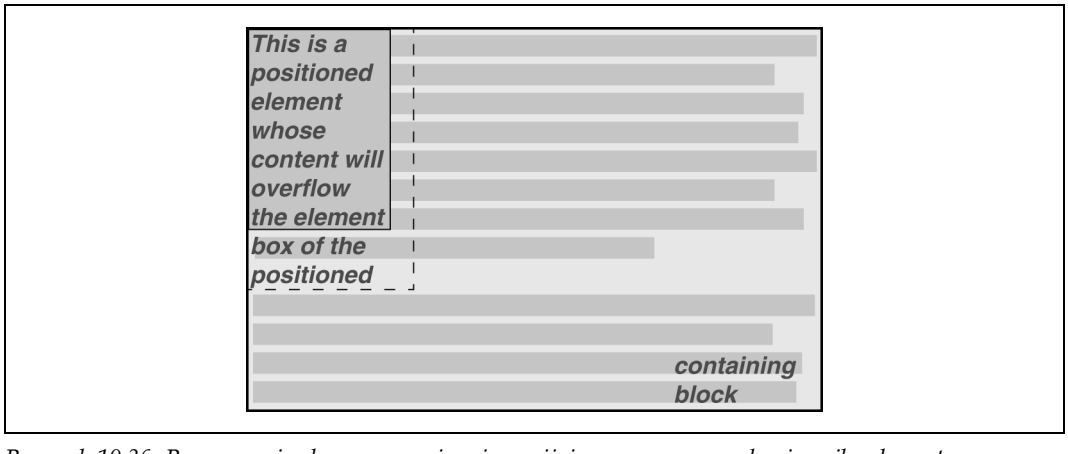
Jak widać, nie przynosi to nic dobrego. Prostokąt przycinający rozciąga się w górę i w lewo, ale ponieważ nie ma tam żadnej zawartości, brak znaczącej różnicy.



Rysunek 10.35. Rozszerzenie obszaru przycinającego poza pojemnik elementu

Z drugiej strony — bardziej przydatne może się okazać wyjście poza dolny i prawy brzeg zamiast poza górny lub lewy. Rysunek 10.36 pokazuje rezultat uzyskany za pomocą poniższej reguły (należy pamiętać, że przerywane linie znajdują się na rysunku jedynie w celach demonstracyjnych):

```
div#sidebar {position: absolute; top: 0; left: 0; width: 5em; height: 7em; clip:
rect(0,6em,9em,0);}
```



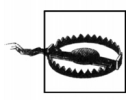
Rysunek 10.36. Rozszerzanie obszaru przycinania poniżej oraz na prawo od pojemnika elementu

Reguła przytoczona powyżej powiększa obszar, w którym zawartość elementu może być widoczna. Nie zmienia jednak układu zawartości dokumentu, więc jedynym widzianym efektem jest uwidocznienie większej ilości zawartości pod elementem. Tekst nie rozlewa się w prawą stronę, ponieważ szerokość pojemników wierszy jest nadal ograniczona przez szerokość pozycjonowanego elementu. Jeśli zawartością elementu byłby rysunek o szerokości większej niż pozycjonowany element lub formatowany tekst z długim wierszem, elementy te mogłyby być widoczne po prawej stronie pozycjonowanego elementu aż do miejsca, w którym kończy się prostokąt przycinający.

Jak już pewnie każdy zdał sobie z tego sprawę, składnia `rect(...)` jest raczej niespotykana w porównaniu z resztą specyfikacji CSS. Oparta jest ona na wczesnych projektach części CSS poświęconej pozycjonowaniu, które używały schematu odsunięcia od lewego górnego rogu. W przeglądarce Internet Explorer zaimplementowano ją, zanim CSS2 stało się pełnym zaleceniem W3C, i w taki sposób zrodził się konflikt, ponieważ w ostatniej chwili dokonano zmiany w składni `rect(...)` — na wartości odsunięcia boków, które obowiązują w całej pozostałej części CSS2. Dokonanie zmiany było rozsądne, ponieważ sprawiło, że reguły CSS stały się wewnętrznie spójne.

Jednak było już za późno. Implementacja pojawiła się na rynku i zamiast zmusić Microsoft do przeprogramowania przeglądarki i spowodować, że potencjalnie istniejące strony będą źle wyświetlane, standard został zmieniony tak, by odpowiadał implementacji. Niestety oznacza to, że nie jest możliwe ustawienie spójnego prostokąta przycinającego, jeśli wysokość i szerokość elementu nie są precyzyjnie zdefiniowane.

Problem ten pogłębia jeszcze bardziej fakt, że `rect(...)` pozwala na użycie jedynie wartości długości oraz `auto`. Dodanie wartości procentowych do zestawu poprawnych wartości dla `rect(...)` poprawiłoby w dużym stopniu jego używalność. Można mieć nadzieję, że przyszłe wersje CSS dodadzą taką możliwość.



Długa i zagmatwana historia właściwości `clip` wpływa na to, że we współczesnych przeglądarkach zachowuje się ona w sposób niespójny i nie można polegać na jej zachowaniu w środowisku o wielu różnych przeglądarkach.

Widoczność elementu

Oprócz przycinania i wypływania można również kontrolować widoczność całego elementu.

visibility

Wartości:	`visible | hidden | collapse | inherit`
Wartość początkowa:	`visible`
Stosuje się do:	Wszystkich elementów
Dziedziczona:	Tak
Wartość wyliczona:	Jak określono

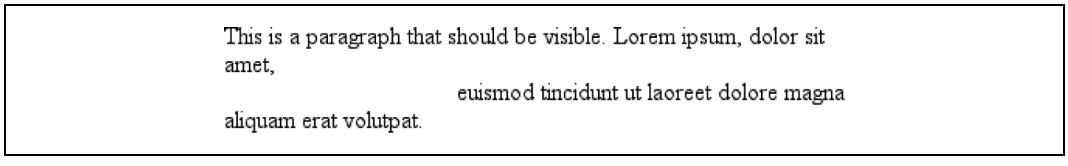
Stosowanie tej właściwości jest stosunkowo proste. Jeśli użyje się wartości `visible` dla właściwości `visibility`, element będzie oczywiście widoczny.

Kiedy zadeklaruje się `visibility: hidden`, to element (używając sformułowania ze specyfikacji) będzie niewidzialny. Element taki — mimo że niewidoczny — nadal ma taki sam wpływ na układ dokumentu, jakby był widoczny. Innymi słowy — nadal znajduje się on na swoim miejscu, tylko że go nie widać. Należy zrozumieć różnicę pomiędzy tą wartością a zadeklarowaniem `display: none`. W takim przypadku element nie tylko nie jest wyświetlony, ale

zostaje także usunięty z dokumentu i nie ma żadnego wpływu na jego układ. Rysunek 10.37 pokazuje dokument, w którym element `em` ma wartość `hidden`, zgodnie z poniższym fragmentem kodu oraz deklaracją stylu:

```
em.trans {visibility: hidden; border: 3px solid gray; background: silver; margin: 2em;
padding: 1em;}

<p>
   This is a paragraph that should be visible. Lorem ipsum, dolor sit amet, <em
   class="trans">consectetuer adipiscing elit, sed diam nonummy nibh </em> euismod
   tincidunt ut laoreet dolore magna aliquam erat volutpat.
</p>
```



Rysunek 10.37. Uczynienie elementów niewidzialnymi poprzez ukrycie ich pojemników

Wszystko, co zwykle jest widoczne w elemencie — zawartość, tło, obramowanie — staje się niewidzialne. Przestrzeń, która normalnie byłaby zajęta przez element, nadal istnieje, ponieważ element pozostaje częścią układu dokumentu. Tylko my po prostu go nie widzimy.

Warto również zauważyć, że możliwe jest ustawienie wartości `visible` dla potomka elementu, który ma `visibility` równą `hidden`. Powoduje to wyświetlenie elementu tam, gdzie normalnie by się znajdował, pomimo tego, że niewidzialny jest jego przodek (i być może także rodzeństwo). Ponieważ widoczność jest właściwością dziedziczoną, w celu przeprowadzenia takiej operacji wystarczy zadeklarować potomkowi wartość `visible`:

```
p.clear {visibility: hidden;}
p.clear em {visibility: visible;}
```

Jeśli chodzi o `visibility: collapse`, to wartość ta wykorzystywana jest przy wyświetlaniu tabel i omówiona zostanie w kolejnym rozdziale. Według specyfikacji CSS2 wartość `collapse` zastosowana do elementów niebędących tabelami działa tak samo jak wartość `hidden`.

Pozycjonowanie bezwzględne

Ponieważ większość przykładów oraz rysunków z poprzednich podrozdziałów ilustrowała pozycjonowanie bezwzględne, Czytelnik jest już w połowie drogi do zrozumienia sposobu działania tej wartości właściwości `position`. Większość pozostałych do omówienia kwestii obejmuje szczegóły tego, co się dzieje, kiedy wywołane jest pozycjonowanie bezwzględne.

Bloki zawierające elementy a elementy pozycjonowane bezwzględnie

Gdy element jest pozycjonowany bezwzględnie, jest całkowicie usuwany z układu dokumentu. Następnie jest pozycjonowany w stosunku do bloku go zawierającego, a jego brzegi rozmieszczane są przy użyciu właściwości przesunięcia (`top`, `left` i tak dalej). Element pozycjonowany nie opływa zawartości innych elementów ani też ich zawartość nie opływa elementu pozycjonowanego. Sugeruje to, że element pozycjonowany bezwzględnie może zachodzić na inne elementy lub inne elementy mogą go zasłaniać (pod koniec rozdziału zostanie wyjaśnione, w jaki sposób można wpłynąć na kolejność zachodzenia elementów na siebie).

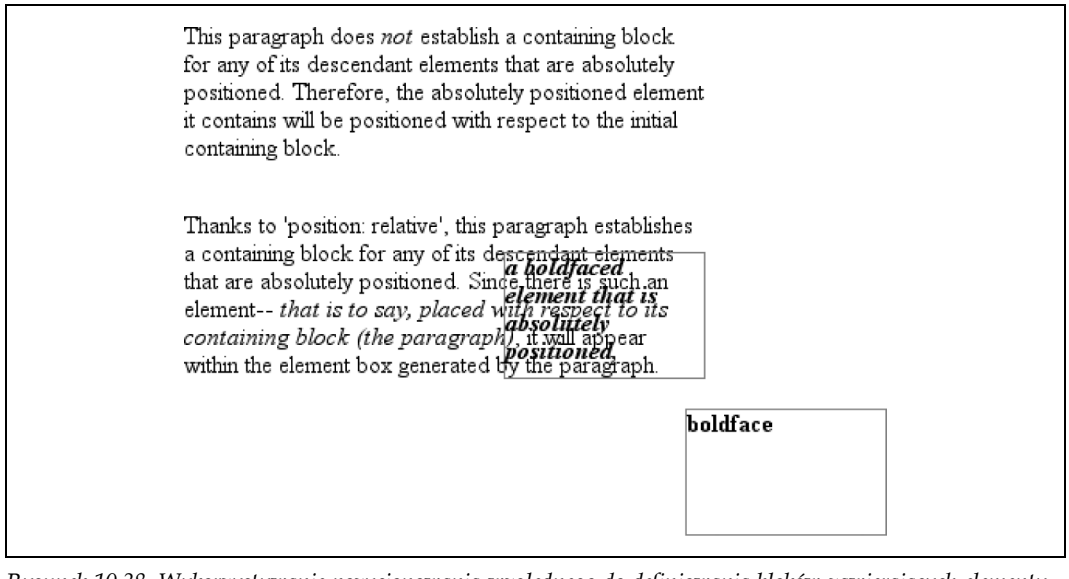
Blokiem zawierającym element pozycjonowany bezwzględnie jest najbliższy element będący jego przodkiem, który ma wartość `position` równą `static`. Często zdarza się, że autor wybiera element, który będzie służył jako blok zawierający elementu pozycjonowanego bezwzględnie, i nadaje mu wartość właściwości `position` równą `relative`, bez żadnych przesunięć:

```
p.contain {position: relative;}
```

Warto zastanowić się nad przykładem znajdującym się na rysunku 10.38, który jest ilustracją następującego kodu:

```
p {margin: 2em;}
p.contain {position: relative;}    /* ustanowienie bloku zawierającego element */
b {position: absolute; top: auto; right: 0; bottom: 0; left: auto; width: 8em; height:
5em; border: 1px solid gray;}

<body>
   <p>
      This paragraph does <em>not</em> establish a containing block for any of its
      descendant elements that are absolutely positioned.  Therefore, the absolutely
      positioned <b>boldface </b> element it contains will be positioned with respect
      to the initial containing block.
   </p>
   <p class="contain">
      Thanks to 'position: relative', this paragraph establishes a containing block
      for any of its descendant elements that are absolutely positioned. Since there
      is such an element-- <em>that is to say, <b>a boldfaced element that is
      absolutely positioned,</b> placed with respect to its containing block (the
      paragraph)</em>, it will appear within the element box generated by the
      paragraph.
   </p>
</body>
```



Rysunek 10.38. Wykorzystywanie pozycjonowania względnego do definiowania bloków zawierających elementy

Elementy `b` w obu akapitach są pozycjonowane bezwzględnie. Różnica polega na bloku zawierającym, który został wykorzystany dla każdego z nich. Element `b` w pierwszym akapicie jest pozycjonowany względem początkowego bloku zawierającego, ponieważ wszystkie elementy

będące jego przodkami mają wartość właściwości `position` równą `static`. Drugi akapit został jednak ustawiony na `position: relative`, dlatego ustanawia on blok zawierający dla swoich potomków.

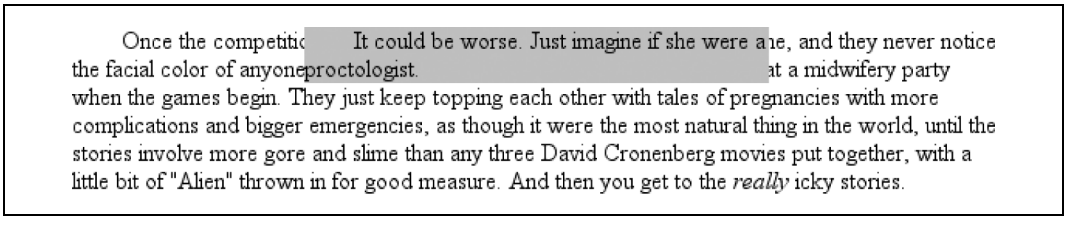
Nietrudno jest zauważyć, że w drugim akapicie pozycjonowany element zachodzi na część zawartości tekstowej akapitu. Nie ma sposobu, aby tego uniknąć, chyba że umieści się pogrubiony tekst poza obszarem akapitu (przy użyciu ujemnych wartości dla `right` bądź jednej z pozostałych właściwości przesunięcia) lub wyznaczy dopełnienie dla akapitu tak, by było dostatecznie szerokie, aby pomieścić element pozycjonowany. Tekst akapitu prześwituje przez element pozycjonowany, ponieważ element `b` ma przezroczyste tło. Jedynym sposobem uniknięcia prześwitywania jest ustawienie tła dla elementu pozycjonowanego bądź też całkowite usunięcie go z akapitu.

Czasami chcemy się upewnić, że to element `body` będzie tworzył blok zawierający dla wszystkich swoich potomków, zamiast pozwalać przeglądarce na wybranie początkowego bloku zawierającego. Wystarczy tylko zadeklarować następującą regułę:

```
body {position: relative;}
```

W takim dokumencie można wstawić akapit pozycjonowany bezwzględnie — jak w poniższym kodzie — i otrzymać rezultat widoczny na rysunku 10.39:

```
<p style="position: absolute; top: 0; right: 25%; left: 25%; bottom: auto; width: 50%;
height: auto; background: silver;">...</p>
```



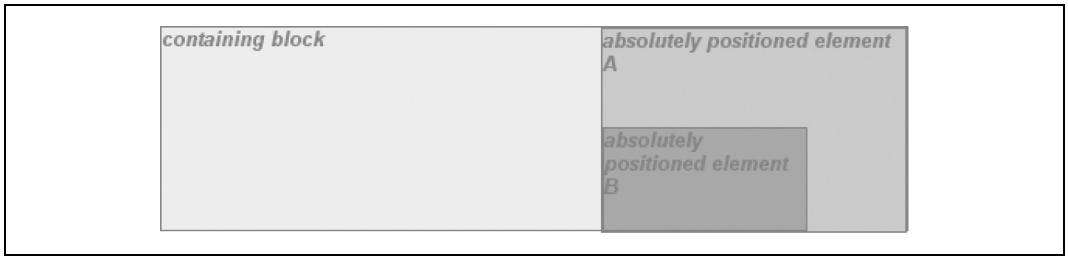
Rysunek 10.39. Pozycjonowanie elementu, którego blokiem zawierającym jest element główny

Akapit jest teraz umieszczony na samym początku dokumentu, ale jego szerokość stanowi połowę szerokości dokumentu i dlatego zasłania on kilka pierwszych elementów.

Ważną kwestią, którą należy podkreślić, jest fakt, iż kiedy element jest pozycjonowany bezwzględnie, ustanawia on również blok zawierający dla swoich elementów potomnych. Można na przykład najpierw pozycjonować element bezwzględnie, a następnie pozycjonować bezwzględnie jednego z jego potomków — jak widać na rysunku 10.40, który został wygenerowany za pomocą następujących stylów oraz kodu:

```
div {position: relative; width: 100%; height: 10em; border: 1px solid; background: #EEE;}
div.a {position: absolute; top: 0; right: 0; width: 15em; height: 100%; margin-left:
auto; background: #CCC;}
div.b {position: absolute; bottom: 0; left: 0; width: 10em; height: 50%; margin-top:
auto; background: #AAA;}

<div>
   <div class="a">absolutely positioned element A
      <div class="b">absolutely positioned element B</div>
   </div>
   containing block
</div>
```



Rysunek 10.40. Elementy pozycjonowane bezwzględnie ustanawiają bloki zawierające

Należy również pamiętać, że — jeśli dokument jest przewijany — elementy pozycjonowane będą się przewijały wraz z nim. Dzieje się tak w przypadku wszystkich elementów pozycjonowanych bezwzględnie, które nie są potomkami elementów o sztywnej pozycji. Wynika to z faktu, że elementy pozycjonowane są względem czegoś, co jest częścią normalnego układu dokumentu. Jeśli na przykład pozycjonuje się bezwzględnie tabelę, której blokiem zawierającym jest początkowy blok zawierający dokumentu, wtedy tabela będzie przewijana, ponieważ początkowy blok zawierający jest częścią normalnego układu dokumentu. Podobnie, nawet jeśli elementy pozycjonowane bezwzględnie zagnieżdżone są na cztery poziomy w dół, najbardziej „zewnętrzny" z nich będzie nadal pozycjonowany względem początkowego bloku zawierającego. W ten sposób będzie on przewijany z początkowym blokiem zawierającym wraz ze wszystkimi swoimi potomkami.

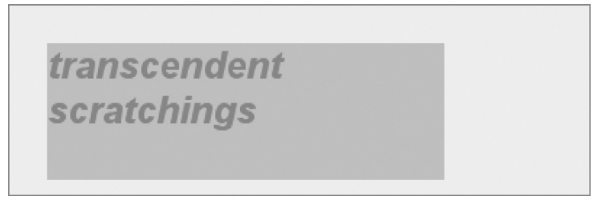
Kolejny podrozdział dostarczy pomocnych informacji tym osobom, które chcą, aby elementy *nie przewijały* się wraz z treścią dokumentu, a ich pozycja została ustalona względem obszaru wyświetlania.

Rozmieszczenie i rozmiar elementów pozycjonowanych bezwzględnie

Połączenie ze sobą koncepcji rozmieszczenia oraz rozmiaru elementów może wydawać się nieco dziwne, jednak jest ono niezbędne w przypadku elementów pozycjonowanych bezwzględnie, ponieważ specyfikacja wiąże je ze sobą bardzo ściśle. Po bliższym przyjrzeniu się tym kwestiom połączenie to okazuje się nie takie znowu dziwne. Można rozważyć, co się stanie, kiedy element pozycjonowany jest z wykorzystaniem wszystkich czterech właściwości przesunięcia, jak poniżej:

```
#masthead h1 {position: absolute; top: 1em; left: 1em; right: 25%; bottom: 10px;
margin: 0; padding: 0; background: silver;}
```

Wysokość oraz szerokość pojemnika elementu `h1` uzależnione są od umieszczenia jego zewnętrznych krawędzi marginesów, jak pokazano na rysunku 10.41.



Rysunek 10.41. Ustalenie wysokości elementu w oparciu o właściwości przesunięcia

Gdyby wysokość bloku zawierającego element została zwiększona, nagłówek h1 również stałby się wyższy; gdyby blok zawierający element został zwężony, element h1 również stałby się węższy. Gdyby do elementu h1 dodano marginesy bądź dopełnienie, miałoby to wpływ na wyliczoną wysokość oraz szerokość h1.

Jednak co stanie się, gdy zrobi się to wszystko, a dodatkowo spróbuje się jawnie ustawić wysokość oraz szerokość:

```
#masthead h1 {position: absolute; top: 0; left: 1em; right: 10%; bottom: 0; margin: 0;
padding: 0; height: 1em; width: 50%; background: silver;}
```

Coś nie będzie pasować, ponieważ jest bardzo mało prawdopodobne, żeby wszystkie te wartości były poprawne. Tak naprawdę, by wszystkie wymienione wartości były prawidłowe, blok zawierający element musiałby być dokładnie dwa i pół razy szerszy niż wyliczona wartość h1 dla font-size. Każda inna szerokość będzie oznaczała, że przynajmniej jedna wartość jest niepoprawna i musi zostać zignorowana. Odgadnięcie, która to z nich, zależy od wielu różnych czynników, a czynniki te zmieniają się w zależności od tego, czy element jest zastępowany, czy niezastępowany.

Można również rozważyć następującą regułę:

```
#masthead h1 {position: absolute; top: auto; left: auto;}
```

Jaki powinien być rezultat? Tak się składa, że odpowiedź na to pytanie *nie* brzmi następująco: „Wartości mają być wyzerowane". Znajduje się ona w kolejnym podrozdziale.

Automatyczne krawędzie

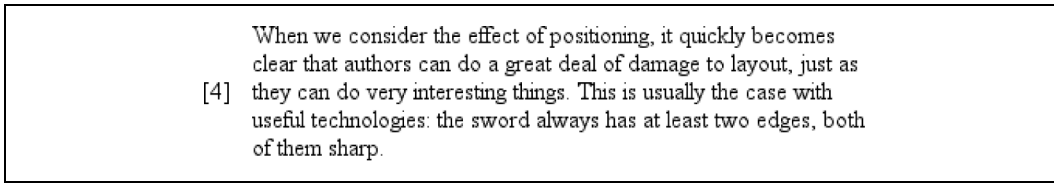
Kiedy element jest pozycjonowany bezwzględnie, specjalne zachowanie występuje wtedy, gdy któraś z właściwości przesunięcia różna od bottom ustawiona jest na auto. Jako przykład można wziąć top. Można rozważyć następującą sytuację:

```
<p>
   When we consider the effect of positioning, it quickly becomes clear that authors
   can do a great deal of damage to layout, just as they can do very interesting
   things.<span style="position: absolute; top: auto; left: 0;">[4]</span> This is
   usually the case with useful technologies: the sword always has at least two edges,
   both of them sharp.
</p>
```

Co powinno się zdarzyć? W przypadku left sprawa jest prosta. Lewa krawędź elementu powinna być umieszczona wzdłuż lewej krawędzi bloku zawierającego element (a można zakładać, że będzie to początkowy blok zawierający). Jednak w przypadku top dzieje się coś o wiele ciekawszego. Góra elementu pozycjonowanego powinna znaleźć się w jednej linii z miejscem, w którym umieszczona byłaby góra elementu, gdyby nie byłby on pozycjonowany. Innymi słowy, należy sobie wyobrazić, w którym miejscu zostałby umieszczony element span, gdyby wartość jego właściwości position była równa static; jest to *pozycja statyczna* — miejsce, w którym powinna znaleźć się jego górna krawędź. CSS2.1 omawia tę kwestię następująco:

> …termin „pozycja statyczna" (elementu) odnosi się mniej więcej do pozycji, jaką ten element miałby w układzie normalnym. Uściślając, pozycja statyczna dla „top" jest odległością pomiędzy górną krawędzią bloku zawierającego elementu a krawędzią marginesu górnego hipotetycznego pojemnika, który byłby pierwszym pojemnikiem elementu, gdyby jego właściwość „position" była ustawiona na „static". Wartość ta jest ujemna, jeśli hipotetyczny pojemnik znajduje się powyżej bloku zawierającego element.

W ten sposób powinno się otrzymać rezultaty widoczne na rysunku 10.42.



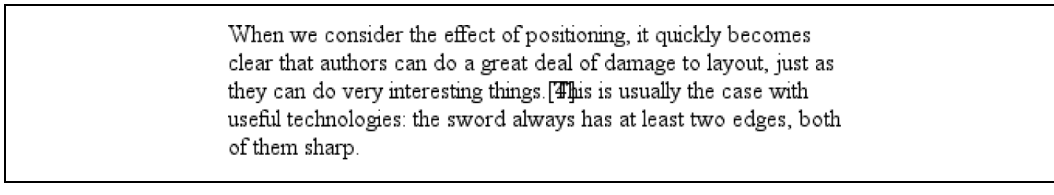
Rysunek 10.42. Pozycjonowanie bezwzględne elementu spójnie z jego pozycją „statyczną"

Element „[4]" znajduje się tuż poza treścią akapitu, ponieważ lewa krawędź początkowego bloku zawierającego element znajduje się na lewo od lewego brzegu akapitu.

Takie same podstawowe reguły obowiązują w przypadku właściwości `left` oraz `right` ustawionych na `auto`. W tych przypadkach lewy (bądź prawy) brzeg elementu pozycjonowanego znajduje się w jednej linii z miejscem, w którym krawędź ta zostałaby umieszczona, gdyby element ten nie był pozycjonowany. Można zmodyfikować powyższy przykład tak, by właściwości `left` oraz `right` zostały ustawione na `auto`:

```
<p>
   When we consider the effect of positioning, it quickly becomes clear that authors
   can do a great deal of damage to layout, just as they can do very interesting
   things.<span style="position: absolute; top: auto; left: auto;">[4]</span> This is
   usually the case with useful technologies: the sword always has at least two edges,
   both of them sharp.
</p>
```

Da to rezultat przedstawiony na rysunku 10.43.



Rysunek 10.43. Pozycjonowanie bezwzględne elementu spójnie z jego pozycją „statyczną"

Teraz element „[4]" znajduje się dokładnie w tym miejscu, w którym by się znalazł, gdyby nie był pozycjonowany. Warto zauważyć, że ponieważ jest jednak pozycjonowany, miejsce, w którym znajdowałby się w układzie normalnym, jest wypełniane. Sprawia to, że element pozycjonowany nakłada się na zawartość akapitu znajdującą się w układzie normalnym.

Należy zauważyć, że specyfikacje CSS2 oraz CSS2.1 podają, iż w takich przypadkach „…przeglądarki mogą same odgadnąć prawdopodobną [statyczną] pozycję". Współczesne przeglądarki dość dobrze radzą sobie z traktowaniem wartości `auto` dla `left` oraz `right` w sposób zamierzony i z umieszczaniem elementu spójnie z miejscem, które zajmowałby on w układzie normalnym.

Takie autorozmieszczanie działa tylko w pewnych sytuacjach — ogólnie rzecz biorąc wtedy, gdy nie istnieje zbyt wiele ograniczeń, jeśli chodzi o inne wymiary elementu pozycjonowanego. Omówiony przypadek mógł być tego przykładem, ponieważ nie miał żadnych ograniczeń, jeśli chodzi o wysokość bądź szerokość elementu ani też umieszczenie jego krawędzi dolnych oraz prawych. A co stanie się, kiedy takie ograniczenia istnieją? Można rozważyć następującą sytuację:

```
<p>
   When we consider the effect of positioning, it quickly becomes clear that authors
   can do a great deal of damage to layout, just as they can do very interesting
   things.<span style="position: absolute; top: auto; left: auto; right: 0; bottom: 0;
   height: 2em; width: 5em;">[4]</span>  This is usually the case with useful
   technologies: the sword always has at least two edges, both of them sharp.
</p>
```

Nie jest możliwe połączenie wszystkich tych wartości. Ustalenie tego, co się stanie w takiej sytuacji, jest tematem kolejnego podrozdziału.

Rozmieszczenie oraz rozmiar elementów niezastępowanych

Ogólnie rzecz biorąc, rozmiar oraz rozmieszczenie elementu uzależnione są od bloku go zawierającego. Wartości różnych właściwości (`width`, `right`, `padding-left` i tak dalej) wpływają na sytuację, jednak podstawą jest blok zawierający element.

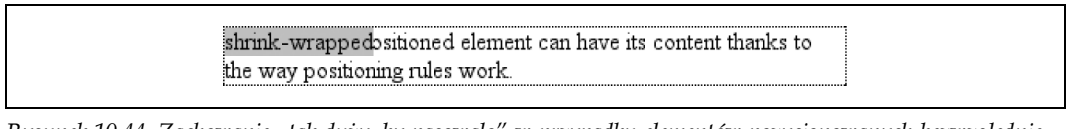
Można rozważyć szerokość oraz umieszczenie w poziomie elementu pozycjonowanego. Może to zostać przedstawione jako równanie mówiące, że `left + margin-left + border-left-width + padding-left + width + padding-right + border-right-width + margin-right + right` = szerokość bloku zawierającego element.

Takie równanie ma sens. Jest to podstawowe równanie wykorzystywane przy obliczaniu rozmiaru elementów blokowych w układzie normalnym, jedynie z dodanymi właściwościami `left` oraz `right`. W jaki sposób działają one razem? Istnieje kilka reguł, z którymi należy się zapoznać.

Po pierwsze, jeśli właściwości `left`, `width` oraz `right` ustawione są na `auto`, otrzyma się rezultaty przedstawione w poprzednim podrozdziale: lewy brzeg umieszczany jest w swojej pozycji statycznej, zakładając, że pracuje się w języku pisanym od lewa do prawa. W językach pisanych od prawa do lewa to prawy brzeg umieszczany jest w swojej pozycji statycznej. Właściwość `width` elementu ustawiana jest na „tak dużą, by pasowało", co oznacza, że obszar zawartości elementu będzie szeroki jedynie na tyle, by zmieścić jego zawartość. Przypomina to sposób, w jaki zachowują się komórki tabeli. Właściwość, która nie ma pozycji statycznej (`right` w językach pisanych od lewa do prawa i `left` w językach pisanych od prawa do lewa), ustawiana jest tak, by zająć pozostałą odległość. Na przykład:

```
<div style="position: relative; width: 25em; border: 1px dotted;">
   An absolutely positioned element can have its content
   <span style="position: absolute; top: 0; left: 0; right: auto; width: auto;
   background: silver;">shrink-wrapped</span>
   thanks to the way positioning rules work.
</div>
```

Da to rezultat widoczny na rysunku 10.44.



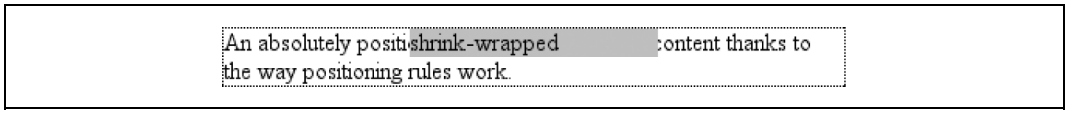
Rysunek 10.44. Zachowanie „tak duży, by pasowało" w przypadku elementów pozycjonowanych bezwzględnie

Góra elementu umieszczana jest wzdłuż góry bloku zawierającego element (w tym przypadku elementu `div`), natomiast szerokość elementu jest wystarczająca, by zmieścić jego zawartość. Pozostała odległość od prawego brzegu elementu od prawej krawędzi bloku go zawierającego staje się wyliczoną wartością właściwości `right`.

Teraz można założyć, że wartość `auto` mają jedynie marginesy (lewy i prawy), a nie `left`, `width` oraz `right` — jak w poniższym przykładzie:

```
<div style="position: relative; width: 25em; border: 1px dotted;">
   An absolutely positioned element can have its content
   <span style="position: absolute; top: 0; left: 1em; right: 1em; width: 10em;
   margin: 0 auto; background: silver;">shrink-wrapped</span>
   thanks to the way positioning rules work.
</div>
```

Tutaj marginesy lewy oraz prawy (mające wartość `auto`) ustawione są tak, by były równe. W praktyce wyśrodkuje to element, jak pokazano na rysunku 10.45.



Rysunek 10.45. Wyśrodkowanie elementu pozycjonowanego bezwzględnie w poziomie za pomocą marginesów o wartości auto

Wygląda to mniej więcej tak samo, jak wyśrodkowywanie dzięki marginesom o wartości `auto` w układzie normalnym. Można zatem spróbować ustawić marginesy na wartość inną niż `auto`:

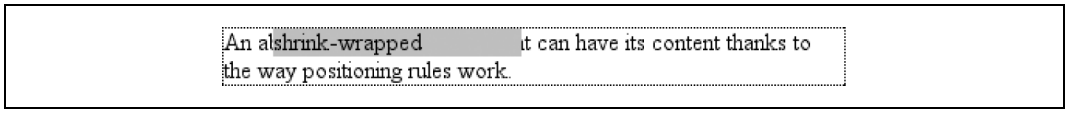
```
<div style="position: relative; width: 25em; border: 1px dotted;">
   An absolutely positioned element can have its content
   <span style="position: absolute; top: 0; left: 1em; right: 1em; width: 10em;
   margin-left: 1em; margin-right: 1em; background: silver;">shrink-wrapped</span>
   thanks to the way positioning rules work.
</div>
```

Teraz pojawia się problem. Właściwości pozycjonowanego elementu `span` w sumie mają tylko `14em`, podczas gdy blok zawierający element ma `25em` szerokości. Istnieje więc deficyt o wielkości jedenastu em, który trzeba gdzieś nadrobić.

Reguły nakazują, by w takiej sytuacji przeglądarka zignorowała wartość podaną dla `right` (w językach pisanych od lewa do prawa; w innym przypadku ignorowana jest właściwość `left`) i wykorzystała tę właściwość do pokrycia różnicy. Innymi słowy, wynik będzie taki sam, jakby zadeklarowano:

```
<span style="position: absolute; top: 0; left: 1em; right: 12em; width: 10em; margin-
left: 1em; margin-right: 1em; background: silver;">shrink-wrapped</span>
```

Da to rezultat zaprezentowany na rysunku 10.46.



Rysunek 10.46. Ignorowanie wartości podanej dla right w sytuacji występowania różnicy szerokości

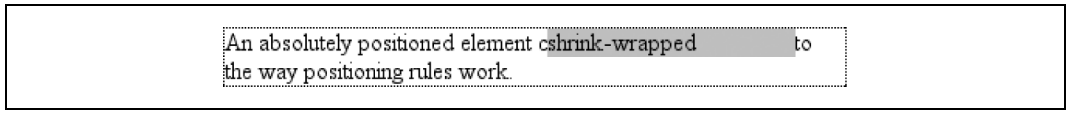
Gdyby jeden z marginesów został ustawiony na `auto`, zmieniony zostałby właśnie on. Gdyby na przykład zmienić style w następujący sposób:

```
<span style="position: absolute; top: 0; left: 1em; right: 1em; width: 10em; margin-
left: 1em; margin-right: auto; background: silver;">shrink-wrapped</span>
```

Wizualnie efekt byłby taki sam, jak na rysunku 10.46, ale należałoby to zawdzięczać wyliczeniu marginesu prawego na `12em`, a nie zmianie wartości przypisanej do właściwości `right`.

Gdyby jednak to lewy margines miał wartość `auto`, to *on* zostałby zmieniony, jak widać na rysunku 10.47:

```
<span style="position: absolute; top: 0; left: 1em; right: 1em; width: 10em;
margin-left: auto; margin-right: 1em; background: silver;">shrink-wrapped</span>
```



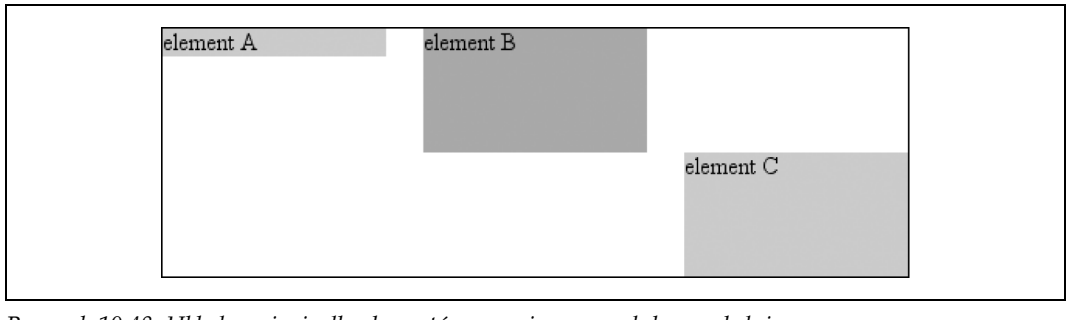
Rysunek 10.47. Ignorowanie wartości dla margin-right w sytuacji występowania różnicy szerokości

Ogólnie rzecz biorąc, jeśli tylko jedna właściwość ustawiona jest na `auto`, to właśnie ona zostanie zmodyfikowana tak, by dopasować równanie przedstawione wcześniej. Przy poniższych stylach szerokość elementu musiałaby zostać powiększona do rozmiaru, który byłby niezbędny, zamiast zmniejszyć się na tyle, żeby po prostu zmieścić zawartość elementu:

```
<span style="position: absolute; top: 0; left: 1em; right: 1em; width: auto;
margin-left: 1em; margin-right: 1em; background: silver;">shrink-wrapped</span>
```

Dotychczas omawiano jedynie zachowanie występujące wzdłuż osi poziomej, jednak w przypadku osi pionowej jest bardzo podobnie. Jeśli weźmie się kwestie omówione wcześniej i odwróci wszystko o dziewięćdziesiąt stopni, otrzyma się prawie takie samo zachowanie. Poniższy kod da na przykład rezultat pokazany na rysunku 10.48:

```
<div style="position: relative; width: 30em; height: 10em; border: 1px solid;">
   <div style="position: absolute; left: 0; width: 30%; background: #CCC; top: 0;">
      element A
   </div>
   <div style="position: absolute; left: 35%; width: 30%; background: #AAA; top: 0;
   height: 50%;">
      element B
   </div>
   <div style="position: absolute; left: 70%; width: 30%; background: #CCC; height:
   50%; bottom: 0;">
      element C
   </div>
</div>
```

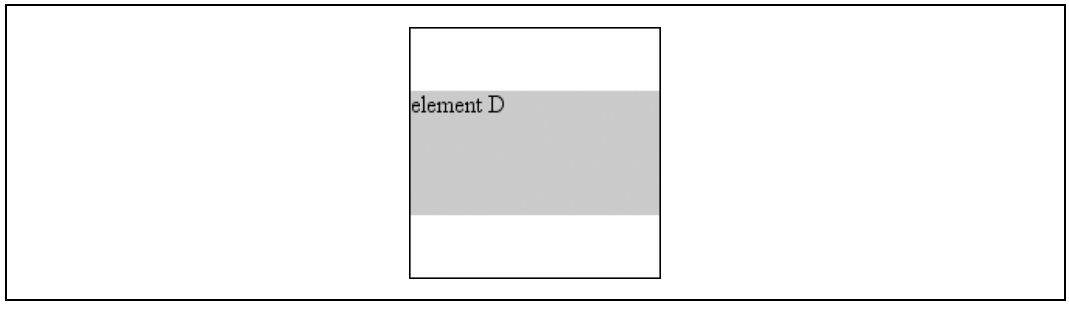


Rysunek 10.48. Układ w pionie dla elementów pozycjonowanych bezwzględnie

W pierwszym przypadku wysokość elementu została dopasowana do zawartości. W drugim — nieokreślona właściwość (`bottom`) została wykorzystana do nadrobienia różnicy pomiędzy dołem elementu pozycjonowanego a dołem bloku go zawierającego. W trzecim przypadku nieokreślona była właściwość `top` i to ona została wykorzystana do nadrobienia różnicy.

W ten sposób marginesy o wartości `auto` mogą prowadzić do wyśrodkowania w pionie. Przy zastosowaniu poniższych stylów pozycjonowany bezwzględnie element `div` będzie wyśrodkowany w pionie wewnątrz bloku go zawierającego, jak widać na rysunku 10.49:

```
<div style="position: relative; width: 10em; height: 10em; border: 1px solid;">
   <div style="position: absolute; left: 0; width: 100%; background: #CCC; top: 0;
   height: 5em; bottom: 0; margin: auto 0;">
      element D
   </div>
</div>
```



Rysunek 10.49. Wyśrodkowanie elementu pozycjonowanego bezwzględnie w pionie dzięki marginesom o wartości auto

Istnieją dwie zmiany w tej kwestii. W układzie poziomym albo `right`, albo `left` muszą być umieszczone zgodnie z pozycjami statycznymi, jeśli ich wartości równe są `auto`. W układzie pionowym tylko `top` przyjmuje swoją pozycję statyczną; `bottom` — z różnych względów — nie może tego zrobić.

W momencie pisania niniejszej książki żadna wersja przeglądarki Internet Explorer (nawet IE7) nie obsługiwała wyśrodkowania w pionie w oparciu o marginesy górne oraz dolne o wartości `auto` w elemencie pozycjonowanym bezwzględnie.

Jeśli zatem rozmiar elementu pozycjonowanego bezwzględnie nie zgadza się w pionie, właściwość `bottom` jest ignorowana. W ten sposób w poniższej sytuacji zadeklarowana wartość `bottom` zostanie zastąpiona przez wyliczoną wartość `5em`:

```
<div style="position: relative; width: 10em; height: 10em; border: 1px solid;">
   <div style="position: absolute; left: 0; width: 100%; background: #CCC; top: 0;
    height: 5em; bottom: 0; margin: 0;">
      element D
   </div>
</div>
```

Nie ma sposobu zignorowania właściwości `top`, jeśli wymiary się nie zgadzają.

Rozmieszczenie oraz rozmiar elementów zastępowanych

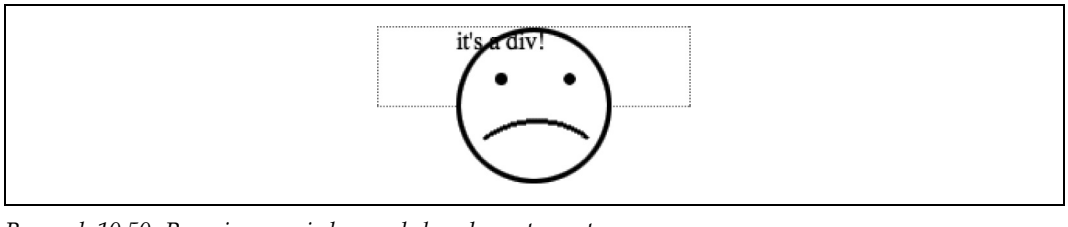
Reguły pozycjonowania dla elementów zastępowanych są inne od reguł dla elementów zastępowanych. Dzieje się tak, ponieważ elementy zastępowane posiadają własną wysokość oraz szerokość i tym samym nie zmieniają się, o ile autor jawnie tego nie zadeklaruje. W ten sposób w przypadku elementów zastępowanych nie występuje koncepcja bycia „tak dużym, by pasowało”.

Zachowania wiążące się z rozmieszczaniem oraz rozmiarem elementów zastępowanych są najłatwiej wyrażane przez serię reguł, których należy przestrzegać w następującej kolejności:

1. Jeśli właściwość `width` ustawiona jest na `auto`, wykorzystywana wartość `width` uzależniona jest od wewnętrznej szerokości zawartości elementu. W ten sposób jeśli szerokość obrazka równa jest pięćdziesięciu pikselom, to wykorzystywana wartość wyliczana jest na `50px`. Jeśli wartość `width` jest w jawny sposób zadeklarowana (na przykład jako `100px` czy `50%`), szerokość ustalana jest na tę wartość.
2. Jeśli właściwość `left` ma wartość `auto` w językach pisanych od lewa do prawa, należy zastąpić `auto` pozycją statyczną. W językach pisanych od prawa do lewa należy pozycją statyczną zastąpić wartość `auto` podaną dla właściwości `right`.
3. Jeśli albo `left`, albo `right` nadal mają wartość `auto` (to znaczy, nie zostały one zastąpione w poprzednim kroku), należy zastąpić `auto` we właściwościach `margin-left` bądź `margin-right` przez `0`.
4. Jeśli w tym momencie zarówno `margin-left`, jak i `margin-right` nadal zdefiniowane są jako `auto`, należy ustawić je tak, by były równe, tym samym wyśrodkowując element w bloku go zawierającym.
5. Na koniec, jeśli została tylko jedna wartość `auto`, należy zmienić ją tak, by była równa reszcie równania.

Prowadzi to do takich zachowań, jakie widać było w przypadku niezastępowanych elementów pozycjonowanych bezwzględnie, o ile założy się, że dla elementów niezastępowanych istnieje jawnie podana szerokość. W ten sposób poniższe dwa elementy będą miały tę samą szerokość oraz rozmieszczenie, zakładając, że szerokość samego obrazka równa jest sto pikseli (jak na rysunku 10.50):

```
<div style="width: 200px; height: 50px; border: 1px dotted gray;">
   <img src="frown.gif" alt="a frowny face" style="position: absolute; top: 0; left:
   50px; margin: 0;">
   <div style="position: absolute; top: 0; left: 50px; width: 100px; height: 100px;
margin: 0;">
      it's a div!
   </div>
</div>
```



Rysunek 10.50. Pozycjonowanie bezwzględne elementu zastępowanego

Tak jak w przypadku elementów niezastępowanych, jeśli wartości się nie zgadzają, przeglądarka powinna zignorować wartość podaną dla `right` w językach pisanych od lewa do prawa lub dla `left` w językach pisanych od prawa do lewa. W ten sposób w poniższym przykładzie zadeklarowana wartość `right` jest zastępowana wyliczoną wartością `50px`:

```
<div style="position: relative; width: 300px;">
   <img src="frown.gif" alt="a frowny face" style="position: absolute; top: 0; left:
   50px; right: 125px; width: 200px; margin: 0;">
</div>
```

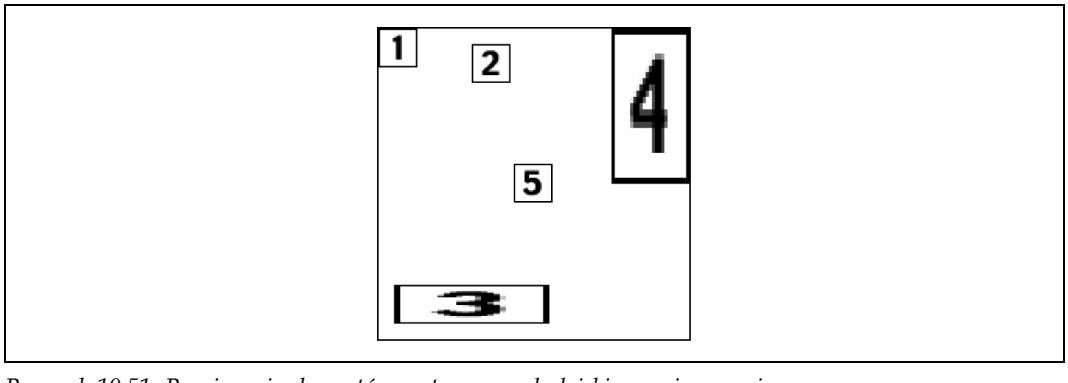
Układem wzdłuż osi pionowej rządzi podobna seria reguł, które mówią:

1. Jeśli właściwość `height` ustawiona jest na `auto`, wyliczona wartość `height` uzależniona jest od wewnętrznej wysokości zawartości elementu. W ten sposób jeśli wysokość obrazka równa jest pięćdziesięciu pikselom, to wykorzystywana wartość wyliczana jest na `50px`. Jeśli wartość `height` jest w jawny sposób zadeklarowana (na przykład jako `100px` czy `50%`), wysokość ustalana jest na tę wartość.
2. Jeśli właściwość `top` ma wartość `auto`, należy zastąpić `auto` pozycją statyczną elementu zastępowanego.
3. Jeśli właściwość `bottom` ma wartość `auto`, należy zastąpić `auto` we właściwościach `margin-top` bądź `margin-bottom` przez `0`.
4. Jeśli w tym momencie zarówno `margin-top`, jak i `margin-bottom` nadal zdefiniowane są jako `auto`, należy ustawić je tak, by były równe, tym samym wyśrodkowując element w bloku go zawierającym.
5. Na koniec, jeśli została tylko jedna wartość `auto`, należy zmienić ją tak, by była równa reszcie równania.

Tak jak w przypadku elementów niezastępowanych, jeśli wartości się nie zgadzają, przeglądarka powinna zignorować wartość podaną dla `bottom`.

W ten sposób poniższy kod dałby rezultat widoczny na rysunku 10.51:

```
<div style="position: relative; height: 200px; width: 200px; border: 1px solid;">
   <img src="one.gif" alt="one" width="25" height="25" style="position: absolute; top:
   0; left: 0; margin: 0;">
   <img src="two.gif" alt="two" width="25" height="25" style="position: absolute; top:
   0; left: 60px; margin: 10px 0; bottom: 4377px;">
   <img src="three.gif" alt=" three" width="25" height="25" style="position: absolute;
   left: 0; width: 100px; margin: 10px; bottom: 0;">
   <img src="four.gif" alt=" four" width="25" height="25" style="position: absolute;
   top: 0; height: 100px; right: 0; width: 50px;">
   <img src="five.gif" alt="five" width="25" height="25" style="position: absolute;
   top: 0; left: 0; bottom: 0; right: 0; margin: auto;">
</div>
```



Rysunek 10.51. Rozciąganie elementów zastępowanych dzięki pozycjonowaniu

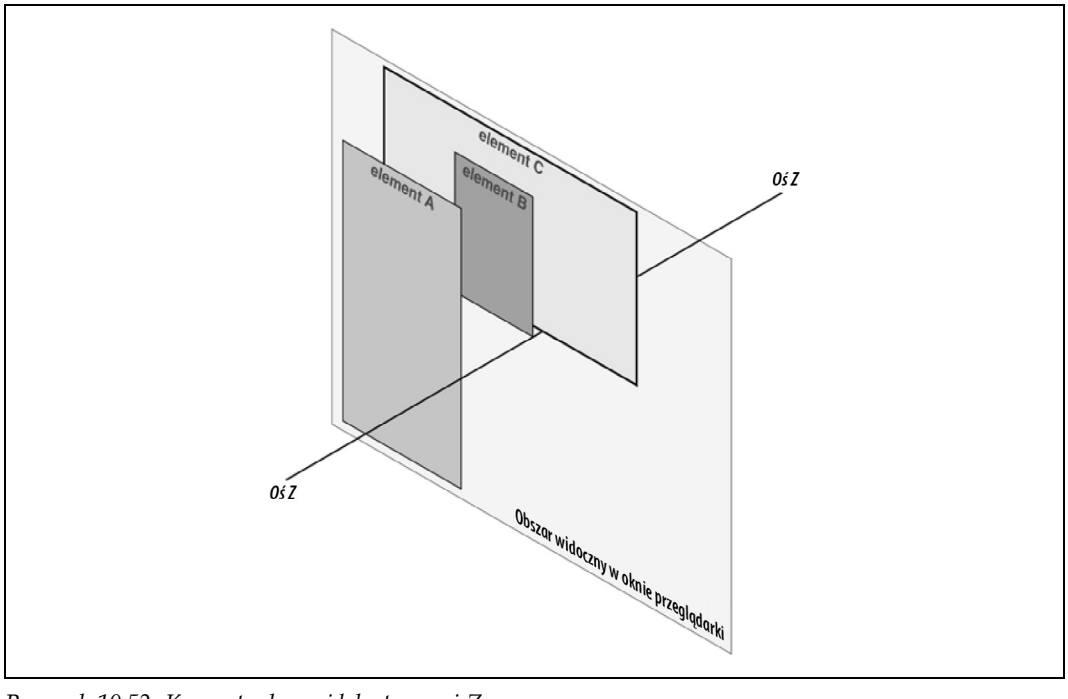
Rozmieszczenie elementów na osi Z

Przy stosowaniu pozycjonowania nieuchronnie można się spotkać z sytuacją, w której — mówiąc obrazowo — dwa elementy będą próbowały istnieć w tym samym miejscu. Jeden z nich będzie musiał zachodzić na drugi, ale otwarte pozostaje pytanie o to, w jaki sposób można kontrolować, który z nich znajdzie się „na wierzchu". Z tego powodu wprowadzono właściwość `z-index`.

z-index

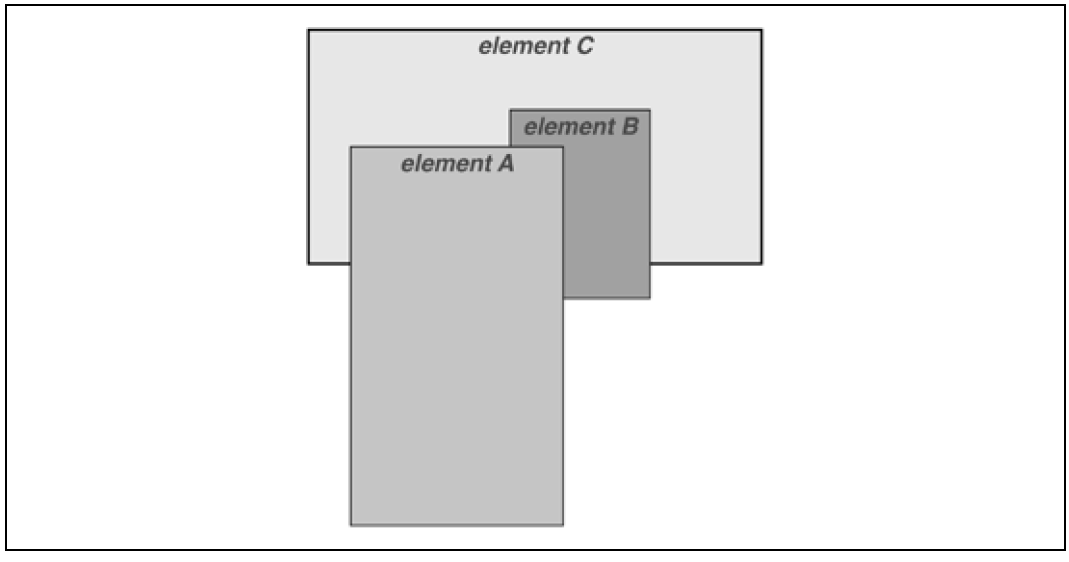
Wartości:	`<integer> | auto | inherit`
Wartość początkowa:	`auto`
Stosuje się do:	Elementów pozycjonowanych
Dziedziczona:	Nie
Wartość wyliczona:	Jak określono

Właściwość `z-index` pozwala autorom zmienić sposób zachodzenia elementów na siebie. Właściwość ta bierze swoją nazwę z układu współrzędnych, w którym kierunek lewo-prawo to oś X, a góra-dół to oś Y. W takim przypadku trzecia oś, która biegnie od przodu do tyłu lub — ujmując to inaczej — od użytkownika do monitora, nazwana jest osią Z. Elementom przydzielane są wartości wzdłuż osi Z przedstawiane za pomocą właściwości `z-index`. Rysunek 10.52 prezentuje ten układ.



Rysunek 10.52. Konceptualny widok stosu osi Z

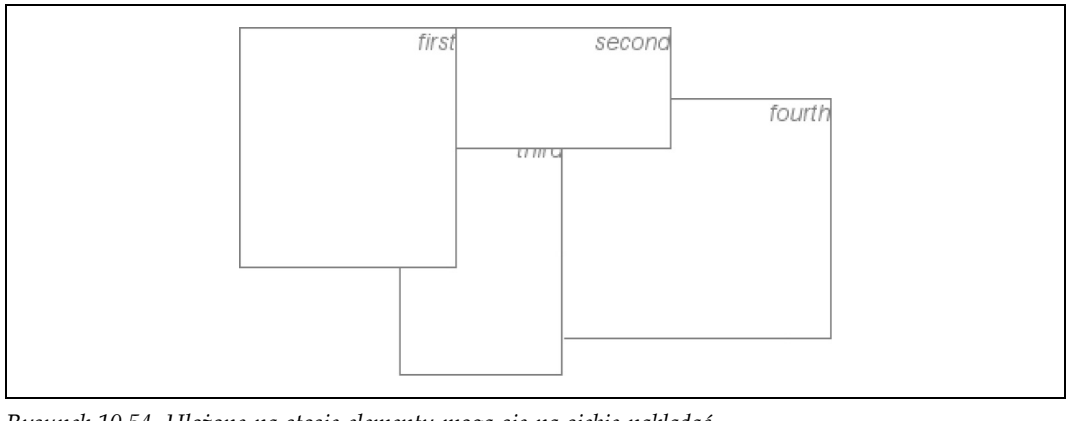
W tym układzie współrzędnych element z większą wartością `z-index` jest optycznie bliżej użytkownika niż ten, który ma mniejszą wartość `z-index`. Spowoduje to, że elementy z wyższą wartością będą zachodzić na inne, jak pokazano na rysunku 10.53, będącym widokiem z góry na rysunek 10.52. Czynność tę nazywa się *układaniem stosu* (ang. *stacking*).



Rysunek 10.53. Sposób układania elementów na stosie

Jako wartość `z-index` może być podana dowolna liczba całkowita (włączając w to liczby ujemne). Przypisanie elementowi ujemnej wartości `z-index` optycznie odsuwa go od użytkownika, co oznacza, że element zostanie umieszczony niżej w stosie. Warto zastanowić się nad poniższymi stylami, zobrazowanymi na rysunku 10.54.

```
p#first {position: absolute; top: 0; left: 0; width: 20%; height: 10em; z-index: 8;}
p#second {position: absolute; top: 0; left: 10%; width: 30%; height: 5em; z-index: 4;}
p#third {position: absolute; top: 15%; left: 5%; width: 15%; height: 10em; z-index: 1;}
p#fourth {position: absolute; top: 10%; left: 15%; width: 40%; height: 10em; z-index: 0;}
```



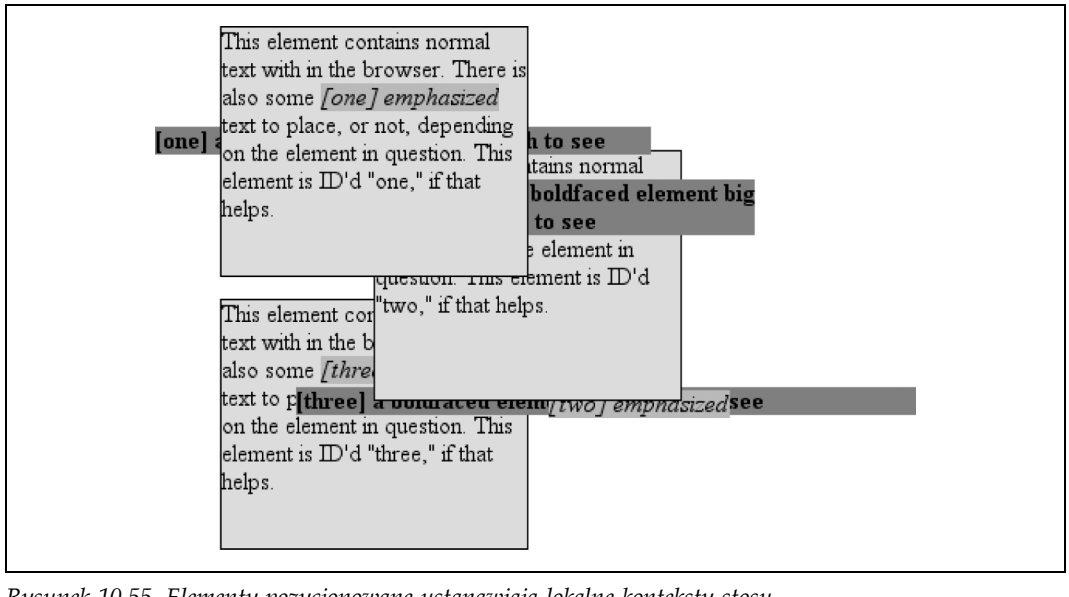
Rysunek 10.54. Ułożone na stosie elementy mogą się na siebie nakładać

Każdy element pozycjonowany jest tutaj według swoich stylów, ale normalna kolejność układania w stos została zmieniona poprzez dodanie wartości `z-index`. Zakładając, że akapity były ułożone w kolejności liczbowej, rozsądnym ułożeniem w stos byłoby ustawienie ich od liczby najmniejszej do największej: `p#first`, `p#second`, `p#third`, `p#fourth`. Umieściłoby to akapit `p#first` za trzema pozostałymi elementami, a akapit `p#fourth` — przed wszystkimi. Teraz jednak dzięki właściwości `z-index` kolejność ustawienia w stos jest w rękach autora strony.

Jak pokazał poprzedni przykład, poszczególne wartości `z-index` nie muszą być kolejnymi liczbami. Można im przydzielić liczby całkowite o dowolnych wielkościach. Jeśli chce się mieć pewność, że jeden element będzie widoczny przed wszystkimi pozostałymi, można do reguły stylu włączyć deklarację taką, jak `z-index: 100000`. Deklaracja zadziała tak, jak się tego można spodziewać, chociaż jeśli innemu elementowi zadeklaruje się wartość `100001` (lub wyższą), to pojawi się on z przodu.

Kiedy już przydzieli się elementowi wartość `z-index` (inną niż `auto`), ustawia on swój własny *kontekst stosu* (ang. *stacking context*). Oznacza to, że każdy potomek elementu będzie miał własną kolejność układania stosu względem elementu przodka. Jest to bardzo podobne do tego, w jaki sposób element ustanawia nowy blok zawierający elementy. Przydzielając poniższe style, można otrzymać coś takiego, jak na rysunku 10.55:

```
p {border: 1px solid; background: #DDD; margin: 0;}
b {background: #808080;}
em {background: #BBB;}
#one {position: absolute; top: 0; left: 0; width: 50%; height: 10em; z-index: 10;}
#two {position: absolute; top: 5em; left: 25%; width: 50%; height: 10em; z-index: 7;}
#three {position: absolute; top: 11em; left: 0; width: 50%; height: 10em; z-index: 1;}
#one b {position: absolute; right: -5em; top: 4em; width: 20em; z-index: -404;}
#two b {position: absolute; right: -3em; top: auto; z-index: 36;}
#two em {position: absolute; bottom: -0.75em; left: 7em; right: -2em; z-index: -42;}
#three b {position: absolute; left: 3em; top: 3.5em; width: 25em; z-index: 23;}
```



Rysunek 10.55. Elementy pozycjonowane ustanawiają lokalne konteksty stosu

Warto zwrócić uwagę na to, gdzie w stosie pojawiają się elementy `b` oraz `em`. Oczywiście każdy z nich jest poprawnie pozycjonowany w stosunku do swojego rodzica. Jednak można przypatrzeć się dokładnie potomkom elementu `p#two`. Podczas gdy element `b` znajduje się z przodu swojego rodzica, element `em` znajduje się za rodzicem, a obydwa znajdują się przed elementem `p#three`! Dzieje się tak, ponieważ wartości `z-index` równe `36` i `-42` odnoszą się do elementu `p#two`, a nie do całego dokumentu. W pewnym sensie `p#two` i każdy z jego potomków dzielą pomiędzy sobą wartość `7` właściwości `z-index`, mając jednocześnie swoje miniindeksy `z-index` szeregujące je w kontekście `p#two`.

Można spojrzeć na to jeszcze w taki sposób, jakby element `b` miał wartość `z-index` równą `7,36`, podczas gdy element `em` ma wartość `7,-42`. Są to jedynie wartości konceptualne; nie mają odpowiednika w specyfikacji. Jednak taki system pomaga zilustrować, w jaki sposób ustalana jest końcowa kolejność w stosie. Można rozważyć to w następujący sposób:

```
p#one        10
p#one b      10,-404
p#two b      7,36
p#two        7
p#two em     7,-42
p#three b    1,23
p#three      1
```

Takie przedstawienie wartości dokładnie opisuje kolejność, w której będą ułożone elementy. Każdy z potomków może się znaleźć przed lub za rodzicem w kolejności układania w stosie, ale zawsze każdy z nich będzie zgrupowany razem ze swoim przodkiem.

Dzieje się też tak, że element ustanawiający kontekst stosu dla swoich potomków jest umieszczany na pozycji `0` w osi Z tego kontekstu. W ten sposób można następująco rozszerzyć powyższy system:

```
p#one        10,0
p#one b      10,-404
p#two b      7,36
p#two        7,0
p#two em     7,-42
p#three b    1,23
p#three      1,0
```

Do zbadania pozostała już tylko jedna wartość. Specyfikacja wypowiada się na temat wartości domyślnej `auto` następująco:

> Poziom stosu generowanego pojemnika w obecnym kontekście stosu jest taki sam, jak pojemnika rodzica. Pojemnik nie ustanawia nowego lokalnego kontekstu stosu (CSS2.1, 9.9.1).

W ten sposób każdy element z zadeklarowanym `z-index: auto` może być traktowany tak, jakby miał ustawione `z-index: 0`. Można się jednak zastanawiać, co dzieje się z elementami mającymi ujemne wartości `z-index`, które są częścią kontekstu stosu początkowego bloku zawierającego. Co się na przykład stanie w sytuacji takiej, jak poniższa?

```
<body>
   <p style="position: absolute; z-index: -1;">Gdzie jestem?</p>
</body>
```

Biorąc pod uwagę reguły związane z ustawianiem w stos, element `body` powinien znajdować się w tym samym kontekście stosu co pojemnik jego rodzica, stąd `z-index` miałby wartość `0`. Nie ustanawia on nowego kontekstu stosu, dlatego pozycjonowany bezwzględnie element `p`

umieszczany jest w tym samym kontekście stosu co element `body` (czyli w kontekście początkowego bloku zawierającego). Innymi słowy, akapit umieszczany jest za elementem `body`. Jeśli `body` nie ma przezroczystego tła, akapit zniknie.

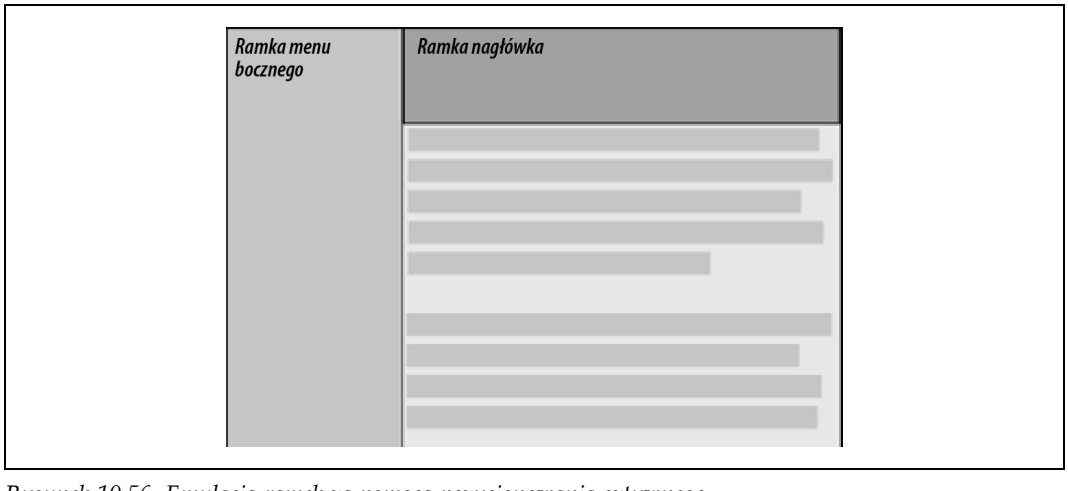
W CSS2 taki wynik był jak najbardziej możliwy. W CSS2.1 reguły układania w stos zostały zmienione w taki sposób, żeby element nigdy nie został ustawiony na stosie za tłem jego kontekstu stosu. Innymi słowy, można rozważyć przypadek, w którym element `body` ustanawia blok zawierający dla swoich potomków (tak jakby był on na przykład pozycjonowany względnie). Element pozycjonowany bezwzględnie będący potomkiem `body` nigdy nie może być ułożony na stosie pod tłem elementu `body`, choć może znaleźć się pod zawartością tego elementu.

W chwili pisania niniejszej książki Mozilla i powiązane przeglądarki całkowicie chowały akapit, nawet jeśli ustawiło się elementom `body` oraz `html` przezroczyste tło. Jest to błąd. Inne przeglądarki, takie jak Internet Explorer, umieszczają akapit nad tłem elementu `body` — nawet jeśli ma on jakieś tło. Zgodnie z CSS2.1 takie zachowanie jest poprawne. Efekt tego jest taki, że ujemne wartości `z-index` mogą prowadzić do nieprzewidywalnych rezultatów, dlatego należy ich używać ostrożnie.

Pozycjonowanie sztywne

Jak zasugerowano w poprzednim podrozdziale, *pozycjonowanie sztywne* (ang. *fixed positioning*) jest bardzo podobne do pozycjonowania bezwzględnego, z tą jednak różnicą, że blokiem zawierającym element jest zawsze widoczny obszar dokumentu. W tym przypadku element jest całkowicie usuwany z układu dokumentu i nie jest pozycjonowany względem żadnej konkretnej części dokumentu.

Pozycjonowanie sztywne można wykorzystać na wiele ciekawych sposobów. Na początek można za jego pomocą zbudować interfejs, który będzie wyglądał tak, jakby przy jego tworzeniu zastosowano układ ramek. Warto przyjrzeć się rysunkowi 10.56, prezentującemu bardzo powszechny schemat układu strony.



Rysunek 10.56. Emulacja ramek za pomocą pozycjonowania sztywnego

Można go uzyskać przy użyciu poniższych stylów:

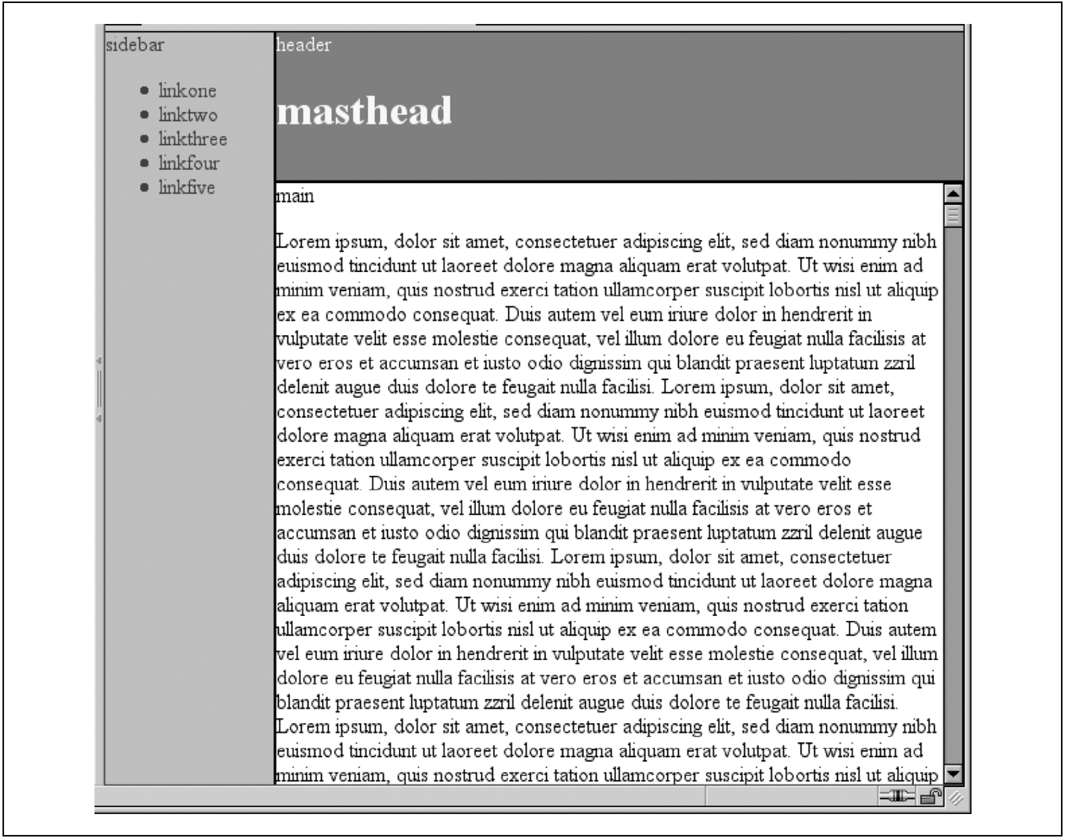
```
div#header {position: fixed; top: 0; bottom: 80%; left: 20%; right: 0; background:
gray;}
div#sidebar {position: fixed; top: 0; bottom: 0; left: 0; right: 80%; background:
silver;}
```

Ustabilizuje to nagłówek oraz menu na górze i po boku obszaru wyświetlania, gdzie pozostaną one bez względu na przewijanie dokumentu. Jednak takie rozwiązanie ma jedną wadę — pozostała część dokumentu zostanie zasłonięta przez elementy o sztywnym pozycjonowaniu. Dlatego dobrze by było objąć ją elementem `div` i użyć jeszcze jednej reguły:

```
div#main {position: absolute; top: 20%; bottom: 0; left: 20%; right: 0; overflow:
scroll; background: white;}
```

Można nawet stworzyć małe przerwy pomiędzy tymi trzema pozycjonowanymi elementami `div` poprzez dodanie odpowiednich marginesów, jak zademonstrowano na rysunku 10.57:

```
body {background: black; color: silver;} /* kolory ze względów bezpieczeństwa */
div#header {position: fixed; top: 0; bottom: 80%; left: 20%; right: 0; background:
gray; margin-bottom: 2px; color: yellow;}
div#sidebar {position: fixed; top: 0; bottom: 0; left: 0; right: 80%; background:
silver; margin-right: 2px; color: maroon;}
div#main {position: absolute; top: 20%; bottom: 0; left: 20%; right: 0; overflow:
auto; background: white; color: black;}
```



Rysunek 10.57. Rozdzielanie „ramek" marginesami

W takim przypadku można także wyłożyć obrazkiem tło elementu `body`. Taka grafika prześwitywałaby w miejscach, w których za pomocą marginesów zostały utworzone szczeliny — mogłyby one oczywiście zostać powiększone, gdyby autor uznał to za stosowne.

Innym zastosowaniem pozycjonowania sztywnego jest umieszczenie na ekranie stałego elementu, takiego jak krótka lista odnośników. Można utworzyć stałą stopkę, na przykład z informacjami o prawach autorskich:

```
div#footer {position: fixed; bottom: 0; width: 100%; height: auto;}
```

Powyższa reguła umieści stopkę na dole widocznego obszaru wyświetlania i pozostawi ją tam bez względu na sposób przewijania dokumentu.

Wadą wykorzystywania pozycjonowania sztywnego jest to, że przeglądarka Internet Explorer dla systemu Windows nie obsługiwała tej właściwości w wersjach wcześniejszych od IE7. Istnieją sposoby ominięcia tego ograniczenia poprzez użycie JavaScriptu, który wprowadza częściową obsługę tych możliwości w starszych wersjach IE/Win, jednak nie zawsze są one akceptowalne dla autorów, ponieważ wyświetlanie jest o wiele mniej spójne niż w przypadku pełnej obsługi pozycjonowania sztywnego. Inną możliwością jest pozycjonowanie bezwzględne w IE/Win i stosowanie pozycjonowania sztywnego w bardziej zaawansowanych przeglądarkach, choć opcja ta nie będzie działała we wszystkich układach dokumentów.

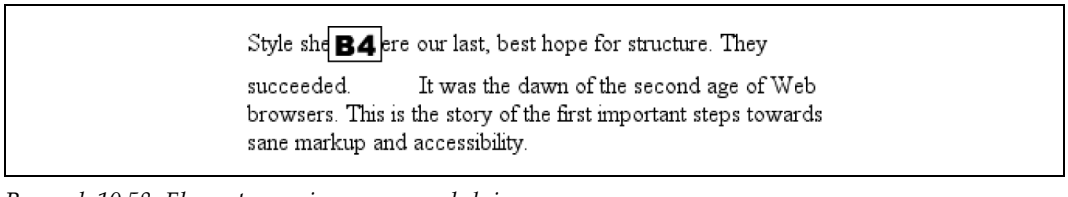
Na stronie *http://css-discuss.incutio.com/?page=EmulatingFixedPositioning* można przeczytać o emulowaniu pozycjonowania sztywnego w starszych wersjach IE/Win.

Pozycjonowanie względne

Pozycjonowanie względne jest najprostszym do zrozumienia schematem pozycjonowania. W schemacie tym element pozycjonowany przesuwany jest przy użyciu właściwości przesunięcia. Może to mieć pewne interesujące następstwa.

Na pozór wydaje się to proste. Powiedzmy, że chcemy przesunąć obrazek w górę i w lewo. Rysunek 10.58 pokazuje wynik działania poniższych stylów:

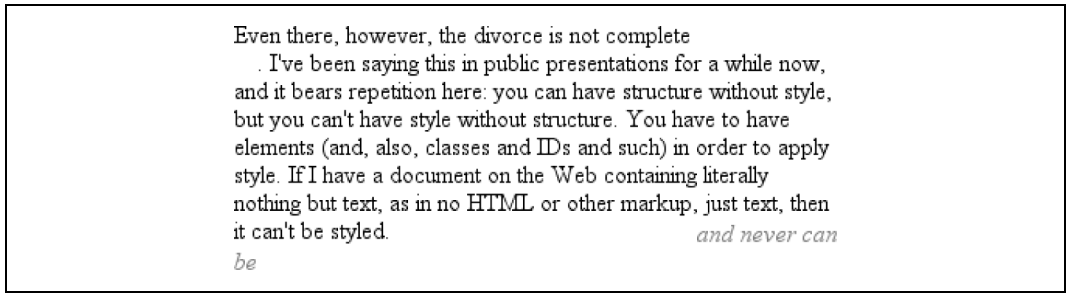
```
img {position: relative; top: -20px; left: -20px;}
```



Rysunek 10.58. Element pozycjonowany względnie

W powyższym przykładzie górny brzeg obrazka został przesunięty o dwadzieścia pikseli w górę, natomiast jego lewy brzeg — o dwadzieścia pikseli w lewo. Należy jednak zwrócić uwagę na pustą przestrzeń w miejscu, gdzie obrazek byłby umieszczony, gdyby nie był pozycjonowany. Przestrzeń ta istnieje, ponieważ element pozycjonowany względnie przesuwany jest ze swojego normalnego miejsca, ale miejsce, które normalnie by zajmował, nie znika. Warto przyjrzeć się rezultatowi zastosowania poniższej reguły, zobrazowanemu na rysunku 10.59:

```
em {position: relative; top: 8em; color: gray;}
```



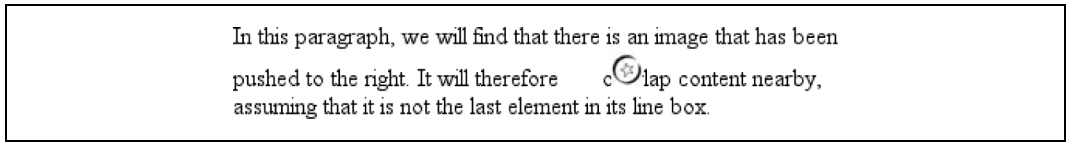
Rysunek 10.59. Element pozycjonowany względnie

Jak widać na rysunku, akapit zawiera pustą przestrzeń. Tam właśnie w normalnych okolicznościach znajdowałby się element `em`, jednak teraz znajduje się on na swojej nowej pozycji; jego kształt dokładnie odzwierciedla puste miejsce, które po nim zostało.

Oczywiście możliwe jest również przesunięcie pozycjonowanego względnie elementu w taki sposób, żeby zachodził na inną część dokumentu. W takim przypadku kod przytoczony poniżej będzie wyświetlony tak, jak na rysunku 10.60:

```
img.slide {position: relative; left: 30px;}

<p>
   In this paragraph, we will find that there is an image that has been pushed to the
   right.  It will therefore <img src="star.gif" alt="A star!" class="slide"> overlap
   content nearby, assuming that it is not the last element in its line box.
</p>
```



Rysunek 10.60. Elementy pozycjonowane względnie mogą nakładać się na pozostałą zawartość dokumentu

Jak widać było w poprzednich podrozdziałach, kiedy element pozycjonuje się względnie, natychmiast ustanawia on dla swoich potomków nowy blok zawierający. Ten blok zawierający odpowiada miejscu, w którym element został wypozycjonowany.

I jeszcze jedna ciekawostka. Co się stanie, gdy pozycjonowany względnie element ma wymuszane wartości pozycjonowania sprzeczne ze sobą? Na przykład:

```
strong {position: relative; top: 10px; bottom: 20px;}
```

Podane są wartości, które implikują dwa odmienne zachowania. Jeśli weźmie się pod uwagę tylko `top: 10px`, to element zostanie odsunięty w dół o dziesięć pikseli, jednak deklaracja `bottom: 20px` jasno nakazuje, by element był przesunięty w górę o dwadzieścia pikseli.

Oryginalna specyfikacja CSS2 nie podaje, co powinno się stać w takim przypadku. W CSS2.1 podano, że w przypadku podania sprzecznych wartości pozycjonowania względnego jedna z wartości zostanie skasowana, tak by była równa odwrotności drugiej wartości. Zatem wartość `bottom` zawsze będzie równać się `-top`. Oznacza to, że poprzedni przykład zostałby potraktowany tak, jakby był odpowiednikiem następującego:

```
strong {position: relative; top: 10px; bottom: -10px;}
```

Element `strong` zostanie zatem przesunięty o dziesięć pikseli w dół. Zaproponowana w specyfikacji zmiana zależy także od kierunku pisania. W językach stosujących zapis od lewej do prawej przy pozycjonowaniu względnym `right` będzie równe `-left`, ale w językach pisanych od prawej do lewej odbędzie się to w drugą stronę i `left` będzie zawsze równe `-right`.

Podsumowanie

Pływanie i pozycjonowanie są bardzo interesującymi możliwościami CSS. Mają one również duże szanse stać się niezłym treningiem w opanowywaniu nerwów, jeśli nie używa się ich wystarczająco uważnie. Nakładanie się elementów, kolejność układania na stosie, rozmiar oraz rozmieszczenie — wszystkie te kwestie muszą zostać rozważone z uwagą przy pozycjonowaniu elementów, natomiast w przypadku elementów pływających należy również wziąć pod uwagę normalny układ dokumentu. Tworzenie układów dokumentu za pomocą pozycjonowania i pływania może wymagać pewnego dostosowania, jednak jest to warte zachodu.

Choć prawdą jest, że duża część układu stron może teraz zostać uwolniona od tabel, to nadal istnieją powody, by w Internecie z tabel korzystać, na przykład przy prezentacji wyceny akcji czy wyników sportowych. W następnym rozdziale omówione zostanie stosowanie CSS w przypadku układu tabel.