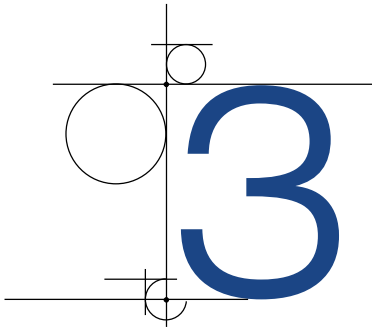


Kaskadowe arkusze stylów (CSS)

W początkowej fazie rozwoju język HTML był rzeczywiście strukturalnym językiem znaczników. W kolejnych jego wersjach dodano elementy opisujące wygląd strony (np. zmianę koloru czy wielkości tekstu). Z czasem pomieszanie warstwy strukturalnej dokumentu z warstwą prezentacyjną zaczęło stwarzać coraz większe problemy.

Rozwiązanie tych problemów stało się możliwe po oddzieleniu warstwy strukturalnej i treści od warstwy prezentacyjnej. Za warstwy: strukturalną, semantyczną i treści wciąż odpowiedzialny jest język HTML, a za warstwę prezentacyjną — język CSS (ang. *Cascading Style Sheets*).

Język CSS określa układ graficzny dokumentu HTML: parametry czcionki, wysokość i szerokość obrazków, ich położenie, rodzaj tła itp. Wszystkie polecenia dotyczące formatowania powinny zostać umieszczone w oddzielnym pliku (arkuszu) i powiązane z elementami zdefiniowanymi w kodzie HTML.

Cechy języka CSS:

- Oddziela strukturę informacyjną witryny od jej warstwy prezentacyjnej.
- Daje większe możliwości formatowania tekstu.
- Pozwala zapisać wszystkie informacje dotyczące wyglądu dokumentu w oddzielnym pliku tekstowym dołączonym później do dokumentów HTML.
- Umożliwia formatowanie wielu dokumentów przy użyciu jednego arkusza stylów.
- Umożliwia stosowanie różnych układów graficznych w zależności od typu urządzenia, na którym zamierzamy wyświetlać projektowaną stronę.

Formatowanie z wykorzystaniem CSS stało się podstawą tworzenia stron internetowych. Należy pamiętać również o tym, że w języku HTML występują przestarzałe znaczniki i atrybuty formatowania (elementy zdeprecjonowane), które będą stopniowo wycofywane z przeglądarek internetowych i zastępowane nowymi metodami formatowania dostępnymi w języku CSS.

Język CSS oferuje nowe możliwości formatowania, które były niedostępne w HTML, np. dotyczące formatowania tekstu, formatowania tła, definiowania obramowania, dodatkowe właściwości definiowania list, pozycjonowania, zmianę wyglądu odsyłaczy, stosowanie filtrów.

3.1. Wstawianie stylów

Kaskadowe arkusze stylów definiują jedynie sposób formatowania elementów dokumentu HTML, ale ich nie tworzą. Elementy te muszą zostać zdefiniowane za pomocą znaczników w dokumencie HTML.

W podstawowej składni kaskadowych arkuszy stylów jest kilka stałych elementów:

```
selektor {właściwość: wartość;}
```

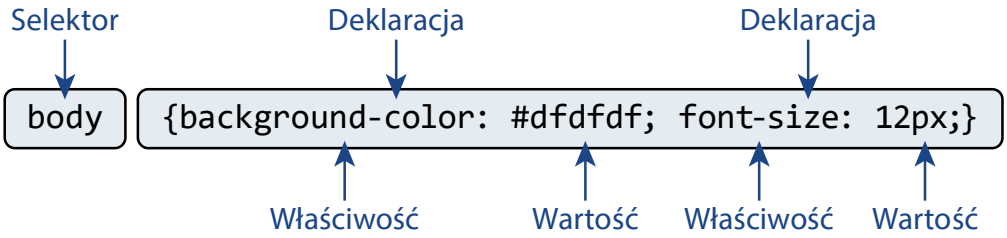
- *selektor* — określa, do jakich znaczników języka HTML odnosi się dalsza część definicji stylu. Selektorem może być dowolny znacznik HTML, np. `p`, `li`, `body`.
- *właściwość* — określa, jaki atrybut znacznika będzie ustawiany, np. wielkość czcionki dla akapitu, tło dla znacznika `body`.
- *wartość* — określa konkretną wartość atrybutu, np. `12pt`, `blue`.

Dla jednego selektora można zdefiniować kilka atrybutów. W definicji stylu należy je oddzielić średnikami.

Przykład 3.1

```
body {background-color: #dfdfdf; font-size: 12px;}
```

Przykład pokazuje ustawienie koloru tła strony i wielkość czcionki na stronie (rysunek 3.1).



Rysunek 3.1. Składnia CSS

Reguły obowiązujące w składni kaskadowych arkuszy stylów:

- Selektor wskazuje na element HTML, dla którego chcemy ustawić styl.
- Blok deklaracji zawiera jedną lub kilka deklaracji rozdzielonych średnikami.
- Każda deklaracja zawiera nazwę właściwości oraz po dwukropku wartość przypisaną tej właściwości.
- Każda deklaracja kończy się średnikiem.
- Blok deklaracji jest otoczony nawiasami klamrowymi.

W dokumencie HTML istnieje kilka sposobów na dołączanie stylów CSS:

- styl lokalny (w linii),
- wewnętrzny arkusz stylów,
- zewnętrzny arkusz stylów.

Sposób wstawiania stylów zależy od konkretnej sytuacji. Podczas projektowania witryny najczęściej stosuje się różne sposoby osadzania arkuszy stylów.

3.1.1. Styl lokalny (w linii)

Korzystając ze stylu lokalnego, można zdefiniować formatowanie pojedynczego elementu strony. Taki styl jest definiowany w tej samej linii, w której znajduje się element formatowany, dlatego nazywany jest on również stylem w linii (ang. *inline*). Umieszczany jest bezpośrednio w dokumencie HTML. Ten sposób definiowania stylów miesza warstwę struktury i treści z warstwą prezentacji. Nie spełnia wymogów kaskadowych arkuszy stylów i powinien być stosowany bardzo oszczędnie.

Przykład 3.2

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Kaskadowe arkusze stylów</title>
  <meta charset="UTF-8">
</head>
  <body style="background-color: #dfdfdf;">
  <p style="color: #911e5d;">Podróż dookoła świata</p>
</body>
</html>
```

Znacznik — rozciąganie stylu

Znacznik `<span>` służy do grupowania kilku elementów strony (np. słów, obrazków, akapitów) w celu nadania im określonego stylu. Ten znacznik sam w sobie nie ma narzuconych żadnych właściwości, nie wywiera żadnego wpływu na zgrupowane elementy, dlatego elementy te będzie określał wyłącznie styl. Styl dla znacznika `<span>` ustala się przy użyciu stylu CSS.

```
<span style="właściwość: wartość;"> … </span>
```

Znacznik `<span>` zwykle jest wykorzystywany wtedy, gdy trzeba inaczej sformatować kilka znaków w obszarze o określonym sposobie formatowania.

Przykład 3.3

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Kaskadowe arkusze stylów</title>
  <meta charset="UTF-8">
</head>
<body style="background-color: #dfdfdf;">
  <h3>Wybieramy się w <span style="color: red;">podróż</span> dookoła świata.
</h3>
</body>
</html>
```

Wynik interpretacji kodu został pokazany na rysunku 3.2.

Rysunek 3.2. Zastosowanie znacznika do fragmentu tekstu

Znaczniki wydzielonych bloków

Używając znaczników blokowych, elementy strony można grupować w bloki i formatować za pomocą stylów. Zdefiniowane bloki mogą zawierać m.in. akapity, listy oraz inne bloki. Znaczniki te umożliwiają wydzielanie na stronie większych, logicznych fragmentów w celu nadania im specyficznego formatowania z użyciem stylów.

Metoda definiowania stylu dla bloków jest podobna jak w przypadku znacznika . Ustala się go za pomocą stylu lokalnego, np.:

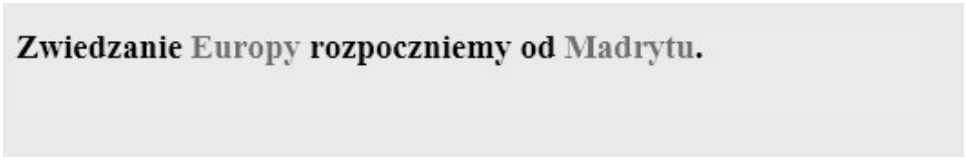
```
<div style="właściwość: wartość;"> … </div>
```

Przykład 3.4

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Bloki</title>
  <meta charset="UTF-8">
</head>
<body style="background-color: #DFDFDF;">
  <div style="width: 500px; height: 50px; background-color: yellow;">
```

```
    <h3>Zwiedzanie <span style="color: red">Europy</span> rozpoczniemy od <span
style="color: red">Madrytu</span>.</h3>
  </div>
</body>
</html>
```

Wynik interpretacji kodu został pokazany na rysunku 3.3.



Rysunek 3.3. Definiowanie stylu dla bloku

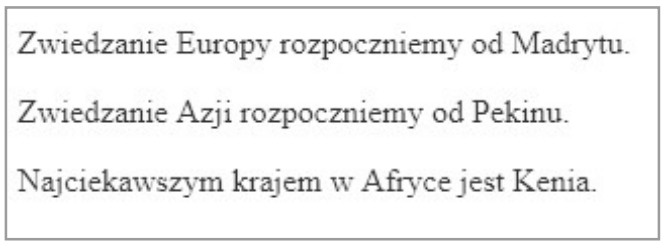
3.1.2. Wewnętrzny arkusz stylów

Wewnętrzny arkusz stylów jest umieszczany w dokumencie HTML dzięki zastosowaniu znacznika `<style>`, który zawsze występuje w części nagłówkowej dokumentu (`<head>`). Tę metodę wstawiania do dokumentu arkusza stylów stosuje się, gdy elementy formatowane pojawiają się na stronie wielokrotnie i wszystkie powinny mieć takie same atrybuty formatowania. Przyjmijmy, że chcemy, żeby wszystkie teksty zdefiniowane jako akapity miały kolor niebieski. Po wpisaniu odpowiedniej deklaracji stylów wszystkie akapity będą napisane czcionką niebieską bez definiowania jej koloru przy każdym elemencie. Zmiana koloru czcionki w deklaracji stylów zmieni kolor czcionki wszystkich akapitów.

Przykład 3.5

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Kaskadowe arkusze stylów</title>
  <meta charset="UTF-8">
  <style>
  p {color: blue;}
  </style>
</head>
<body>
  <p>Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
  <p>Zwiedzanie Azji rozpoczniemy od Pekinu.</p>
  <p>Najciekawszym krajem w Afryce jest Kenia.</p>
</body>
</html>
```

Wynik wykonania kodu prezentuje rysunek 3.4.

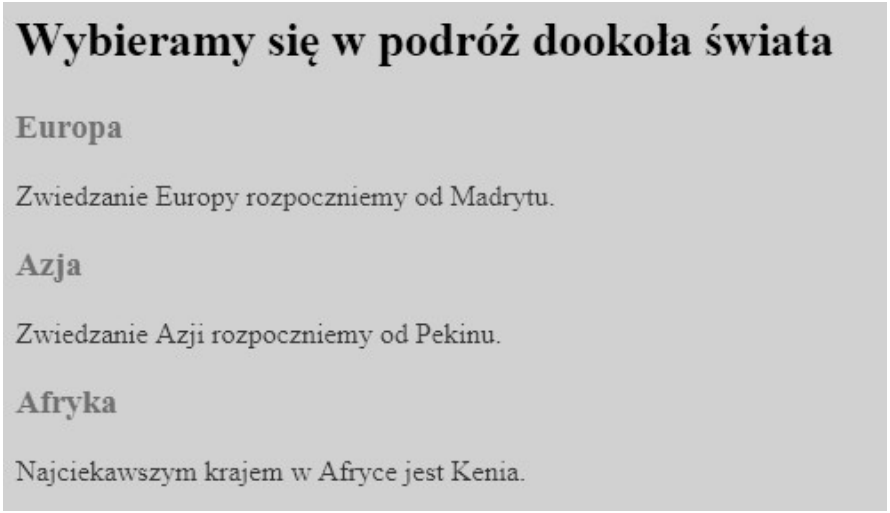


Rysunek 3.4. Definiowanie wewnętrznego arkusza stylów dla znacznika <p>

Przykład 3.6

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Wewnętrzny arkusz stylów</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c6bac0;}
  h2 {font-size: 20pt;}
  h3 {color: green; font-size: 14pt;}
  p {color: blue;}
  </style>
</head>
<body>
  <h2>Wybieramy się w podróż dookoła świata</h2>
  <h3>Europa</h3>
  <p>Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
  <h3>Azja</h3>
  <p>Zwiedzanie Azji rozpoczniemy od Pekinu.</p>
  <h3>Afryka</h3>
  <p>Najciekawszym krajem w Afryce jest Kenia.</p>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.5.



Rysunek 3.5. Definiowanie wewnętrznego arkusza stylów dla różnych znaczników

Ćwiczenie 3.1

Przygotuj teksty i obrazy opisujące wybraną dyscyplinę sportową. Za pomocą poznanych znaczników języka HTML zaprojektuj szablon strony internetowej na ten temat. Zaproponuj teksty, które znajdą się w nagłówku i stopce strony. Umieść przygotowane wcześniej teksty w odpowiednich blokach i podziel je na akapity. Używając stylu lokalnego, zdefiniuj rodzaj, kolor i rozmiar czcionki dla tekstów wyświetlanych na stronie. To samo wykonaj, używając wewnętrznego arkusza stylów. Kiedy warto zastosować styl lokalny, a kiedy wewnętrzny arkusz stylów?

3.1.3. Zewnętrzny arkusz stylów

Największą z zalet stosowania CSS jest możliwość wstawiania zewnętrznych arkuszy stylów. Polega ona na utworzeniu pliku tekstowego z rozszerzeniem *.css*, który będzie zawierał definicję wszystkich stylów używanych w projektowanej witrynie. W dokumencie HTML powinien znaleźć się odnośnik do tzw. zewnętrznego arkusza stylów, czyli do pliku z rozszerzeniem *.css*.

Odnośnik umieszczony w dokumencie HTML powinien mieć postać:

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

Wartością atrybutu `href` powinna być ścieżka dostępu do pliku zawierającego zdefiniowane style. Aby utrzymać porządek w strukturze plików strony, warto umieszczać pliki stylów w odrębnych folderach, np. *css*.

Odnośnik do arkusza stylów powinien zostać umieszczony w części nagłówkowej dokumentu (`<head>`).

Przykład 3.7

```
<!DOCTYPE html>
<html lang="pl-PL">
```

```
<head>
  <title>Kaskadowe arkusze stylów</title>
  <meta charset="UTF-8">
  <link rel="stylesheet" type="text/css" href="arkusz.css">
</head>
<body>
  <p>Wybieramy się w podróż dookoła świata</p>
  <p>Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
</body>
</html>
```

Zaletą tak zdefiniowanych arkuszy stylów jest to, że do wielu dokumentów HTML można dołączyć ten sam plik arkusza stylów, czyli za pomocą jednego pliku CSS można formatować w taki sam sposób wiele stron WWW (np. kilka stron internetowych tej samej witryny).

Można również w jednym dokumencie HTML umieszczać dowolną liczbę zewnętrznych arkuszy stylów. W dokumencie tym za każdym razem musi wystąpić osobny element `<link rel="stylesheet">` odwołujący się do arkusza stylów. Jeżeli pojawi się konflikt dotyczący sposobu formatowania tego samego elementu w różnych arkuszach stylów, wiążące będą deklaracje z arkusza dołączonego najpóźniej.

Ćwiczenie 3.2

Utwórz plik z rozszerzeniem *.css*. Zdefiniuj w nim style tekstów wyświetlanych na stronie. Dołącz go do dokumentu HTML opisującego stronę internetową z ćwiczenia 3.1.

3.1.4. Import arkusza stylów

Do zewnętrznego lub wewnętrznego arkusza stylów można zaimportować zewnętrzny arkusz stylów. Plik z zaimportowanym arkuszem stylów może się znajdować w dowolnym miejscu; należy tylko prawidłowo zdefiniować ścieżkę dostępu do niego. Importowanie arkusza stylów odbywa się podobnie jak dołączanie zewnętrznego arkusza. Priorytet dołączonego arkusza jest taki sam jak priorytet arkusza, do którego został zaimportowany.

Polecenie importowania arkusza ma postać:

```
<style>
@import url("adres importowanego arkusza stylów.css");
</style>
```

Adres arkusza stylów powinien zostać umieszczony w nawiasach okrągłych. Przed poleceniem `import` musi zostać umieszczony znak `@`, a na końcu adresu arkusza stylów — średnik.

Oprócz importowania arkusza można między znacznikami `<style>` i `</style>` umieszczać inne deklaracje stylów. Polecenia importowania arkusza muszą występować na początku arkusza stylów, przed właściwymi regułami CSS.

3.1.5. Kaskadowość arkuszy stylów

Zdarza się, że w dokumencie są umieszczone odwołania do kilku zewnętrznych arkuszy stylów lub na stronie używane są: zewnętrzny arkusz stylów, deklaracja stylów w nagłówku strony oraz style lokalne. Wtedy może się pojawić konflikt dotyczący sposobu formatowania tego samego elementu w różnych arkuszach stylów. Zawsze pierwszeństwo mają style, które są umieszczone bliżej formatowanego elementu. Kaskadowość arkuszy stylów polega na ścisłym określeniu priorytetów stylów i przestrzeganiu zasad formatowania zgodnie z priorytetami. Ważność stylów została ustalona w następujący sposób:

1. Styl lokalny.
2. Rozciąganie stylu.
3. Wydzielone bloki.
4. Wewnętrzny arkusz stylów.
5. Zewnętrzny arkusz stylów.
6. Import stylów do zewnętrznego arkusza.
7. Atrybuty definiowane w dokumencie HTML.

Styl z numerem 1 ma najwyższy priorytet, a styl z numerem 7 — najniższy. Styl z najwyższym priorytetem ma pierwszeństwo w formatowaniu elementów strony. Style o niższym priorytecie formatują element tylko wtedy, gdy style o wyższym priorytecie nie dotyczą tego elementu. Należy zauważyć, że atrybuty definiowane dla elementu w dokumencie HTML mają najniższy priorytet, a więc style CSS definiowane w dowolny sposób zawsze są ważniejsze od tych atrybutów.

Postępując zgodnie z zasadami kaskadowości stylów, w dokumencie HTML polecenie dołączenia zewnętrznego arkusza stylów trzeba umieścić przed definicją wewnętrznego arkusza stylów.

Zasady kaskadowości można zmieniać. Służy do tego polecenie `!important` umieszczone w deklaracji stylu po wartości, której dotyczy. Takie umiejscowienie tego polecenia spowoduje, że właściwość zdefiniowana dla elementu będzie miała pierwszeństwo, nawet jeżeli ma niższy priorytet. Polecenie `!important` dotyczy tylko właściwości, dla której zostało użyte; pozostałe właściwości elementu będą obsługiwane zgodnie z ogólnymi zasadami kaskadowości stylów.

Przykład 3.8

Użycie polecenia `!important` w definicji stylu:

```
p {font-size: 20pt !important; color: green;}
```

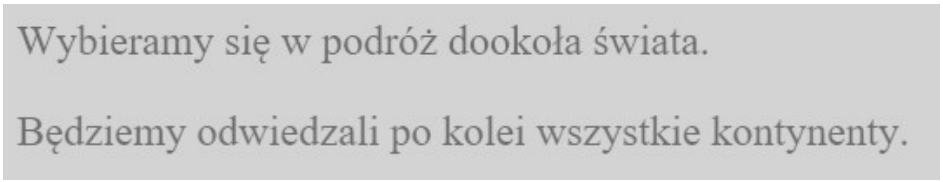
Fragment kodu:

```
<p style="font-size: 12pt; color: red">Kaskadowe arkusze stylów</p>
```

Przykład 3.9

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Arkusz stylów</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p {font-size: 18pt !important; color: green;}
  </style>
</head>
<body>
  <p style="font-size: 12pt; color: #a21b5f;">Wybieramy się w podróż dookoła
świata.</p>
  <p style="font-size: 14pt;">Będziemy odwiedzali po kolei wszystkie
kontynenty.</p>
</body>
</html>
```

Na rysunku 3.6 pokazany został efekt działania polecenia `!important`. Dla pierwszego znacznika `<p>` zastosowany został kolor czcionki zdefiniowany za pomocą stylu lokalnego (bo ma wyższy priorytet od stylu wewnętrznego), a rozmiar czcionki pochodzi z wewnętrznego arkusza stylów (mimo że ma on niższy priorytet, to polecenie `!important` wymusza pierwszeństwo zastosowania stylu czcionki z wewnętrznego arkusza stylów).



Rysunek 3.6. Efekt działania polecenia !important

Ćwiczenie 3.3

W zewnętrznym arkuszu stylów został zdefiniowany selektor `h2`:

```
h2 {color: blue; text-align: center; font-size: 12pt;}
```

W wewnętrznym arkuszu stylów również został zdefiniowany selektor `h2`:

```
h2 {text-align: right; font-size: 24pt;}
```

Jeżeli strona ma powiązanie z zewnętrznym arkuszem stylów, to jakie właściwości będzie miał znacznik `<h2>`? Sprawdź swoją odpowiedź, dołączając tak zdefiniowane style do swojej strony internetowej.

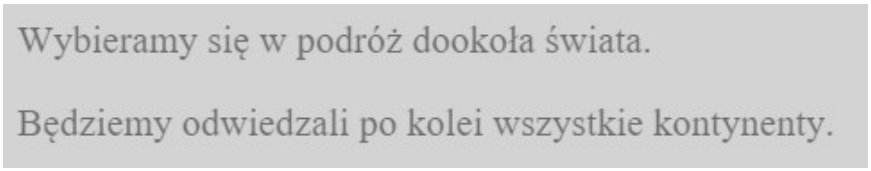
3.1.6. Dziedziczenie

W języku HTML często występuje zagnieżdżanie jednego elementu wewnątrz drugiego. Jeżeli dla elementu nadrzędnego w arkuszach stylów zostały zdefiniowane właściwości, to w większości przypadków elementowi podrzędnemu zostaną przypisane te same właściwości, nawet jeżeli nie zostały dla niego zdefiniowane. Mechanizm ten nazywamy **dziedziczeniem**.

Przykład 3.10

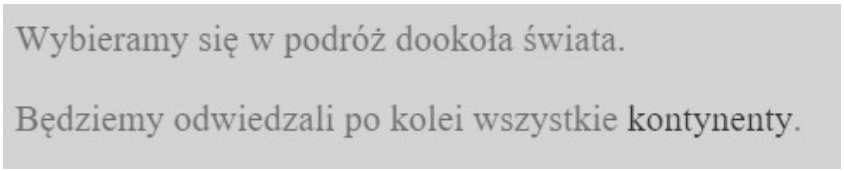
```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Dziedziczenie</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  div {font-size: 16pt; color: green;}
  </style>
</head>
<body>
  <div>
    <p>Wybieramy się w podróż dookoła świata.</p>
    <p>Będziemy odwiedzali po kolei wszystkie <span>kontynenty</span>.</p>
  </div>
</body>
</html>
```

W podanym przykładzie dla tekstu w bloku `<div>` został zdefiniowany kolor zielony. Wewnątrz bloku wystąpiły dwa akapity i znacznik `<span>`, którym nie nadano żadnego stylu, więc dziedziczą go po bloku `<div>` (rysunek 3.7).



Rysunek 3.7. Dziedziczenie stylu dla tekstu w akapitach

Na rysunku 3.8 widzimy efekt zdefiniowania atrybutu koloru dla fragmentu tekstu wewnątrz drugiego akapitu. Ten fragment nie odziedziczył stylu po bloku `<div>`.



Rysunek 3.8. Brak dziedziczenia stylu dla tekstu „kontynenty”

3.2. Składnia języka CSS

Arkusz stylów jest zwykłym plikiem tekstowym i można go utworzyć za pomocą dowolnego edytora tekstu. Można również skorzystać ze specjalnego edytora CSS.

Edytory CSS przyspieszają i ułatwiają pracę, ponieważ automatyzują niektóre działania. Na przykład można automatycznie zdefiniować wielkość i kolor czcionki, kolor tła czy kolor odsyłaczy.

Plik będący zewnętrznym arkuszem stylów musi zostać zapisany z rozszerzeniem *.css*.

Jeżeli do arkusza stylów będą importowane inne arkusze stylów, to polecenia importowania `@import` muszą się znaleźć na początku tego arkusza w postaci:

```
@import url(ścieżka/plik.css);
```

W języku CSS parametry formatowania definiuje się za pomocą **reguł stylów**. Każda reguła odnosi się do określonego elementu.

Podstawowa składnia wygląda następująco:

```
selektor {właściwość: wartość;}
```

Powyższa składnia dotyczy stylów wewnętrznych, definiowanych w części nagłówkowej dokumentu HTML, oraz stylów zewnętrznych. Natomiast dla stylów lokalnych składnia ma postać:

```
<znacznik style="właściwość: wartość;">
```

Do dokumentu HTML można dołączyć arkusz stylów zapisany w pliku *arkusz.css* z podaną przykładową zawartością.

Przykład 3.11

```
body {background-color: #DFDFDF;}

p {font-color: blue; font-size: 5;}
```

Jeżeli dokument HTML i plik z arkuszem stylów zostaną umieszczone w odpowiednich lokalizacjach i do dokumentu HTML zostanie dołączony zewnętrzny plik CSS przy użyciu elementu `<link>`, to po otwarciu dokumentu zdefiniowane w arkuszu stylów ustawienia dotyczące tła strony oraz wyglądu tekstu będą widoczne na tej stronie.

W pliku CSS można umieszczać komentarze. Komentarze zaczynają się znakami `/*`, a kończą się znakami `*/`.

Przykład 3.12

```
/*To jest komentarz*/

body {background-color: #DFDFDF;} /*tutaj też został dodany komentarz*/

p {color: blue; font-size: 5; font-family: courier new;}
```

3.2.1. Wartości i jednostki

Każda właściwość używana do definiowania selektora zawiera zbiór dopuszczalnych wartości oraz dopuszczalne rodzaje wartości.

Mogą to być:

- **Liczby** — całkowite i rzeczywiste. Liczba całkowita składa się z cyfr od `0` do `9`. Liczba rzeczywista może zawierać część całkowitą i część ułamkową. Część ułamkowa zapisana jest po kropce.
- **Procenty**. Wartość procentowa jest zawsze określana względem innej wartości. Na przykład może zostać użyta do określenia wielkości bloków. Podana w procentach wielkość bloku będzie się zmieniała wraz ze zmianą rozmiaru bloku, który go zawiera (rozmiar ten zwykle zależy od rozmiaru okna przeglądarki).
- **Słowa kluczowe**. Nie zawsze wartości wyrażone za pomocą słów kluczowych są poprawnie interpretowane przez przeglądarki.
- **Jednostki długości**. Mogą być zapisywane jako wartości bezwzględne i wartości względne.
  - **Wartości bezwzględne**: `in` (cal, 1 in = 2,54 cm), `cm` (centymetr), `mm` (milimetr), `pt` (punkty, 1 cal to 72 punkty), `pc` (pica, 1 pc = 12 pt), `px` (piksele, 1 px = 0,75 pt). Jednostki bezwzględne są rzadko stosowane. Należy ich używać wtedy, gdy chcemy, aby wybrany element miał taki sam rozmiar niezależnie od rozdzielczości monitora.
  - **Wartości względne**: `em` (1 em jest wielkością rozmiaru czcionki, jaką posiada dany element; wartość 0,1 em jest najmniejszą wartością, jaką przeglądarki są w stanie rozpoznać), `ex` (obliczana na podstawie wysokości małej litery `x` w użytej

czcionce). Jednostki względne są stosowane, gdy odwołujemy się do rozmiaru innego elementu.

- **Jednostki czasu**. Mogą **być liczone w sekundach** (`s`) lub **milisekundach** (`ms`).
- **Ciągi znaków**. Muszą być umieszczone pomiędzy pojedynczym lub podwójnym cudzysłowem, np. "*ciąg znaków*".
- **URL** — np. `url(zdjecia/obraz.jpg)`. Błędem jest wstawienie spacji przed nawiasem otwierającym.

W języku HTML przyjmuje się, że wartości liczbowe bez podanych jednostek mają jednostkę `px`. Natomiast w języku CSS brak jednostki jest traktowany jako błąd. Niektóre przeglądarki ignorują ten błąd i przyjmują jednostkę `px`. Gdy podana wartość jest równa 0, jednostka nie ma znaczenia.

3.2.2. Kolory

W języku HTML zostało zdefiniowanych 16 podstawowych kolorów. W wersji CSS 2.2 oprócz 16 podstawowych kolorów zdefiniowano dodatkowy kolor podstawowy, `orange` (`#FFA500`) — pomarańczowy. W CSS3 zostało zdefiniowanych dodatkowo 130 kolorów rozszerzonych.

Niektóre polecenia arkuszy stylów zawierają definicję kolorów, np.:

```
body {background-color: #c0c0c0;}

h3 {color: #f0f0f0;}

<p style= "background-color: yellow";>Podróż dookoła świata</p>

h2 {color: green; background-color: #c0c0c0;}
```

Wartość dla atrybutu `color` lub `background-color` może być podawana na trzy sposoby. Można stosować słowną angielską nazwę koloru (`blue`, `red`), wprowadzić odpowiednik koloru zapisany w kodzie heksadecymalnym (`#ffcc00`, `#aa2237`) lub w kodzie RGB (`rgb(120,12,270)`, `rgb(39,21,110)`).

Znak `#` przed kodem koloru oznacza, że kolor jest definiowany właśnie w kodzie heksadecymalnym. Kod musi się składać z sześciu lub trzech cyfr, nawet jeśli są to same zera. Im większa jest wartość liczbowa kodu, tym jaśniejszy kolor.

W modelu RGB kod koloru zawiera trzy składowe zapisane w postaci liczb oddzielonych przecinkiem. Napis `rgb` przed kodem koloru oznacza definiowanie koloru w kodzie RGB.

Poza kolorami zapisywanymi w systemie szesnastkowym można stosować nazwy kolorów. Lista dostępnych nazw kolorów jest ograniczona.

Paleta podstawowych kolorów zgodna ze specyfikacją języków HTML i XHTML została podana w tabeli 3.1.

Tabela 3.1. Paleta kolorów

Kolor	Nazwa	Kod HEX	Kod RGB
czarny	`black`	`#000000`	`rgb(0,0,0)`
srebrny	`silver`	`#c0c0c0`	`rgb(192,192,192)`
szary	`gray`	`#808080`	`rgb(128,128,128)`
biały	`white`	`#ffffff`	`rgb(255,255,255)`
kasztanowy	`maroon`	`#800000`	`rgb(128,0,0)`
czerwony	`red`	`#ff0000`	`rgb(255,0,0)`
purpurowy	`purple`	`#800080`	`rgb(128,0,128)`
fuksja	`fuchsia`	`#ff00ff`	`rgb(255,0,255)`
zielony	`green`	`#008000`	`rgb(0,128,0)`
limonkowy	`lime`	`#00ff00`	`rgb(0,255,0)`
oliwkowy	`olive`	`#808000`	`rgb(128,128,0)`
żółty	`yellow`	`#ffff00`	`rgb(255,255,0)`
granatowy	`navy`	`#000080`	`rgb(0,0,128)`
niebieski	`blue`	`#0000ff`	`rgb(0,0,255)`
morski	`teal`	`#008080`	`rgb(0,128,128)`
akwamaryna	`aqua`	`#00ffff`	`rgb(0,255,255)`

Więcej informacji o barwach można znaleźć m.in. na stronie *www.barwy.net*.

Ćwiczenie 3.4

Przeanalizuj sposób kodowania kolorów i odpowiedz na pytanie: Ile maksymalnie kolorów można uzyskać, stosując do definiowania koloru kod heksadecymalny?

Przykłady podawania wartości kolorów w CSS3:

1. *Predefiniowane* — podawanie kolorów w postaci ich nazw.

```
p {background-color: orange;}
```

2. `#rrggbb` — podawanie wartości każdej składowej kolorów podstawowych (`rr` — czerwony, `gg` — zielony, `bb` — niebieski). Wszystkie liczby są dwucyfrowe i zapisane w systemie szesnastkowym, np. `#ff00cc`.

```
p {background-color: #ff0000;}
```

3. `#rgb` — jako składowe kolorów trzeba podawać liczby jednocyfrowe w systemie szesnastkowym, np. `#f0c`.

```
p {background-color: #f00;}
```

4. `rgb(R, G, B)` — można podać oddzielnie każdą składową koloru w systemie dziesiętnym.

```
p {background-color: rgb(255,0,0);}
```

5. `rgb(R%, G%, B%)` — można podać oddzielnie każdą składową koloru w postaci wartości procentowej.

```
p {background-color: rgb(0%,0%,100%);}
```

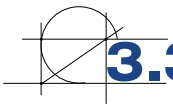
6. *28 kolorów systemowych* — kolory pochodzą z systemu operacyjnego (kolor pulpitu, okien, menu). Gdy użytkownik zmieni kolory w swoim systemie, zmienią się również kolory zdefiniowane tą metodą. Sama deklaracja wygląda tak samo jak w punkcie 1.
7. `hsl(H, S%, L%)` — odmienny od RGB sposób opisu kolorów. `H` (*Hue*) określa barwę, `S` (*Saturation*) — nasycenie, `L` (*Lightness*) — jasność. Kolor powstaje przez wybranie określonej barwy (`H`), a następnie sterowanie nasyceniem (`S`) i jasnością (`L`) aż do uzyskania pożądanego efektu.

```
p {background-color: hsl(120,65%,75%);}
```

8. `rgba(R, G, B, A)`, `rgba(R%, G%, B%, A%)`, `hsla(H, S%, L%, A)` — uzyskiwanie koloru z przezroczystością. `A` (*alpha*) określa wartość przezroczystości z zakresu od `0` (całkowita przezroczystość) do `1` (brak przezroczystości).

```
p {background-color: rgba(255,0,0,0.5);}
```

```
p {background-color: hsla(120,65%,75%,0.3);}
```



3.3. Selektory

Selektorem może być dowolny element języka HTML, dla którego chcemy zdefiniować parametry formatowania. W zależności od tego, w jaki sposób odwołujemy się w definicji reguły do formatowanych elementów, wyróżniamy następujące rodzaje selektorów:

- selektory elementów,
- selektory atrybutów,
- selektory specjalne,
- selektory pseudoklas,
- selektory pseudoelementów.

3.3.1. Selektory elementów

Selektory elementów wybierają elementy na podstawie ich nazwy. Składnia dla selektorów elementów to podana już podstawowa składnia języka CSS:

```
selektor {właściwość: wartość;}
```

Selektor typu elementu

Jest to podstawowy typ selektora. Służy do definiowania formatowania znaczników występujących na stronie (np.: `p`, `h3`, `table`, `img`).

Przykład 3.13

```
p {color: blue; font-size: 4pt;}

h3 {color: green; font-size: 24pt;}
```

Selektor uniwersalny

Selektor uniwersalny (inaczej: ogólny) to selektor pasujący do wszystkich znaczników. Jest oznaczany gwiazdką `*`. Składnia tego selektora to:

```
* {właściwość: wartość;}
```

Zamiast grupować elementy, np.:

```
p, h1, h2, h3, h4, table {font-family: courier new; color: green;}
```

można użyć selektora uniwersalnego:

```
* {font-family: courier new; color: green;}
```

Selektor potomka

Przy użyciu selektora potomka można formatować elementy, które są zawarte wewnątrz innych znaczników, czyli leżą niżej w hierarchii drzewa dokumentu. Składnia selektora potomka to:

```
selektor selektor {właściwość: wartość;}
```

Jeżeli w dokumencie HTML wewnątrz znacznika znajduje się inny znacznik, np. wewnątrz znacznika `<div>` został umieszczony znacznik `<span>` w podanej postaci:

```
<div>

Wybieramy się w podróż dookoła świata.

<p>Będziemy odwiedzali po kolei wszystkie <span>kontynenty</span>.</p>

</div>
```

to definicja stylu dla selektora potomka będzie wyglądać następująco:

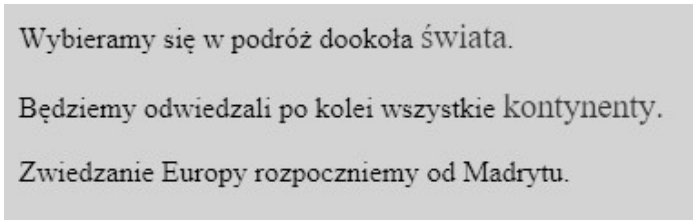
```
div span {font-size: 14pt; color: blue;}
```

W tym przypadku nie ma znaczenia, czy znacznik `<span>` znajduje się bezpośrednio w znaczniku `<div>`, czy w kolejnych znacznikach zawartych w jego wnętrzu.

Przykład 3.14

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Selektor potomka</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: grey;}
  div span {font-size: 14pt; color: blue;}
  </style>
</head>
<body>
  <div>
    Wybieramy się w podróż dookoła <span>świata</span>.
    <p>Będziemy odwiedzali po kolei wszystkie <span>kontynenty</span>.</p>
  </div>
  <p>Zwiedzanie Europy rozpoczniemy od <span>Madrytu</span>.</p>
</body>
</html>
```

Wynik interpretacji kodu został pokazany na rysunku 3.9.



Rysunek 3.9. Działanie selektora potomka

Jak widać, potomek nie musi być umieszczony bezpośrednio w znaczniku, którego jest potomkiem. W powyższym przykładzie w znaczniku `<div>` znajduje się znacznik `<p>`, a dopiero w nim znacznik `<span>`, którego dotyczy definiowany styl.

Selektor dziecka

Selektor dziecka służy do definiowania formatowania elementów, które znajdują się o jeden rząd niżej w hierarchii drzewa dokumentu. Ma on postać:

```
rodzic > dziecko {właściwość: wartość;}
```

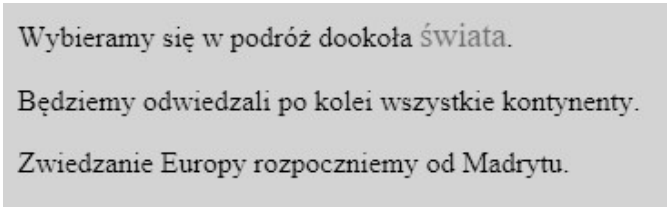
gdzie symbol > oznacza bezpośredni związek między elementami.

Znacznik będący dzieckiem musi wystąpić bezpośrednio wewnątrz znacznika nadrzędnego.

Przykład 3.15

```
<!DOCTYPE html>
<html lang="pl-PL" >
<head>
  <title>Selektor dziecka</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  div > span {font-size: 14pt; color: green;}
  </style>
</head>
<body>
  <div>
    Wybieramy się w podróż dookoła <span>świata</span>.
    <p>Będziemy odwiedzali po kolei wszystkie <span>kontynenty</span>.</p>
  </div>
  <p>Zwiedzanie Europy rozpoczniemy od <span>Madrytu</span>.</p>
</body>
</html>
```

Wynik interpretacji kodu został zilustrowany na rysunku 3.10.



Rysunek 3.10. Działanie selektora dziecka

Kolor *zielony* i wielkość liter `14pt` zostały nadane tylko tekstowi otoczonemu znacznikiem `<span>`, który został umieszczony bezpośrednio w bloku `<div>`. Znacznik `<span>` nieumieszczony bezpośrednio w znaczniku `<div>` nie został objęty zdefiniowanym stylem.

Selektor sąsiadującego brata

Dla elementów znajdujących się w tym samym rzędzie hierarchii można zdefiniować selektor sąsiadującego brata. Umożliwia on nadanie drugiemu z sąsiadujących elementów zdefiniowanych atrybutów formatowania. Selektor sąsiadującego brata przyjmuje postać:

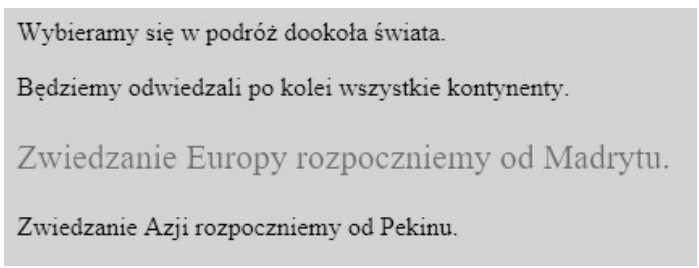
```
brat1 + brat2 {właściwość: wartość;}
```

gdzie symbol + oznacza następowanie po sobie elementów.

Przykład 3.16

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Selektor sąsiadującego brata</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  div + p {font-size: 16pt; color: red;}
  </style>
</head>
<body>
  <div>
    Wybieramy się w podróż dookoła <span>świata</span>.
    <p>Będziemy odwiedzali po kolei wszystkie <span>kontynenty</span>.</p>
  </div>
  <p>Zwiedzanie Europy rozpoczniemy od <span>Madrytu</span>.</p>
  <p>Zwiedzanie Azji rozpoczniemy od Pekinu.</p>
</body>
</html>
```

Wynik interpretacji kodu został pokazany na rysunku 3.11.



Rysunek 3.11. Działanie selektora sąsiadującego brata

W powyższym przykładzie tylko znacznik `<div>` oraz przedostatni znacznik `<p>` znajdują się w tym samym rzędzie hierarchii. Przedostatni znacznik `<p>` występuje

bezpośrednio po znaczniku i tylko dla tekstu otoczonego tym znacznikiem <p> jest realizowane formatowanie zdefiniowane dla selektora sąsiadującego brata.

Selektor braci

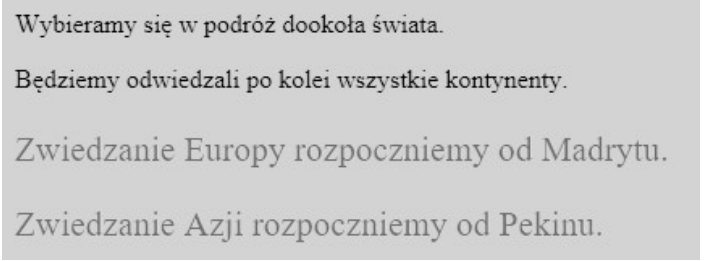
Dla elementów znajdujących się w tym samym rzędzie hierarchii można zdefiniować selektor braci. Umożliwia on nadanie drugiemu i wszystkim następnym sąsiadującym elementom zdefiniowanych atrybutów formatowania. Selektor braci przyjmuje postać:

```
brat1 ~ brat2 {właściwość: wartość;}
```

Przykład 3.17

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Selektor braci</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  div ~ p {font-size: 16pt; color: red;}
  </style>
</head>
<body>
  <div>
    Wybieramy się w podróż dookoła <span>świata</span>.
    <p>Będziemy odwiedzali po kolei wszystkie <span>kontynenty</span>.</p>
  </div>
  <p>Zwiedzanie Europy rozpoczniemy od <span>Madrytu</span>.</p>
  <p>Zwiedzanie Azji rozpoczniemy od Pekinu.</p>
</body>
</html>
```

Wynik wykonania kodu został zilustrowany na rysunku 3.12.

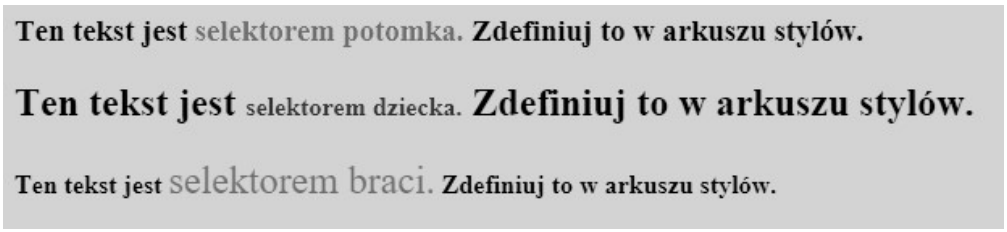


Rysunek 3.12. Działanie selektora braci

W powyższym przykładzie znacznik oraz dwa ostatnie znaczniki <p> znajdują się w tym samym rzędzie hierarchii. Dlatego dla tekstu otoczonego tymi znacznikami <p> jest realizowane formatowanie zdefiniowane dla selektora braci.

Ćwiczenie 3.5

Zastanów się, jak — wykorzystując arkusze stylów oraz selektory potomka, dziecka i braci — należy zdefiniować style, które pozwolą na wyświetlenie tekstów sformatowanych tak jak na rysunku 3.13. Przygotuj stronę, na której uzyskasz podobny rezultat.



Rysunek 3.13. Wynik działania różnych selektorów

3.3.2. Selektory atrybutów

W języku CSS można formatować znaczniki na podstawie atrybutów, które one mają. Na przykład jeżeli dla znacznika <p> (akapit) zostanie zdefiniowany atrybut id, to inaczej będzie wyświetlany akapit z id="lewy", a inaczej akapit z id="prawy".

Postać selektora atrybutów:

```
selektor[atrybut="wartość atrybutu"] {właściwość: wartość;}
```

Przykład 3.18

```
p[id="zolty"] {font-size: 16pt; color: yellow;}
```

Prosty selektor atrybutu

Prosty selektor atrybutu jest wykorzystywany dla elementów, którym został nadany określony atrybut, ale jego wartość nie ma znaczenia. Ma on postać:

```
selektor[atrybut] {właściwość: wartość;}
```

lub

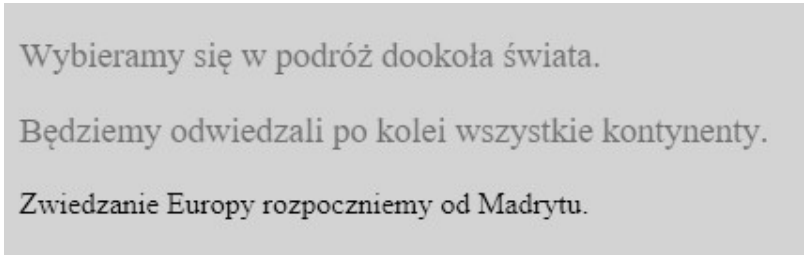
```
[atrybut] {właściwość: wartość;}
```

Przykład 3.19

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Selektor atrybutu</title>
```

```
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p[id] {font-size: 14pt; color: green;}
  </style>
</head>
<body>
  <p id="center">Wybieramy się w podróż dookoła świata.</p>
  <p id="left">Będziemy odwiedzali po kolei wszystkie kontynenty.</p>
  <p>Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
</body>
</html>
```

Wynik interpretacji kodu został pokazany na rysunku 3.14.



Rysunek 3.14. Działanie prostego selektora atrybutu

W powyższym przykładzie dla akapitów pierwszego i drugiego został zdefiniowany atrybut `id` o różnych wartościach. Niezależnie od wartości tego atrybutu akapity zostały sformatowane zgodnie z definicją stylu dla znacznika `<p>`. Ponieważ trzeci akapit nie ma zdefiniowanego atrybutu `id`, nie został sformatowany.

Selektor atrybutu o określonej wartości

Ten rodzaj selektora określa formatowanie, gdy atrybut ma określoną wartość. Ma on postać:

```
selektor[atrybut="wartość"] {właściwość: wartość;}
```

lub

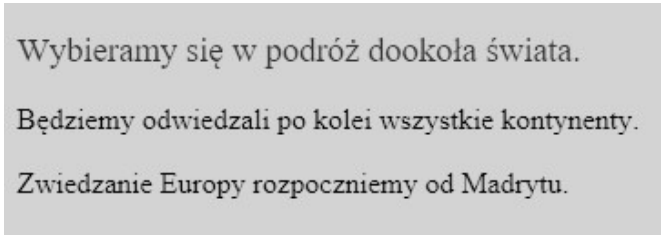
```
[atrybut="wartość"] {właściwość: wartość;}
```

Przykład 3.20

```
<!DOCTYPE html>
<html lang="pl-PL">
  <head>
```

```
  <title>Selektor atrybutu</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p[id="center"] {font-size: 14pt; color: blue;}
  </style>
</head>
<body>
  <p id="center">Wybieramy się w podróż dookoła świata.</p>
  <p id="left">Będziemy odwiedzali po kolei wszystkie kontynenty.</p>
  <p>Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
</body>
</html>
```

Wynik wykonania kodu jest widoczny na rysunku 3.15.



Rysunek 3.15. Działanie selektora atrybutu o określonej wartości

W tym przykładzie tylko pierwszy akapit został sformatowany zgodnie z definicją stylów, ponieważ tylko tu został zdefiniowany atrybut `id` z wartością `center`.

Selektor atrybutu zawierającego określony wyraz

Ten rodzaj selektora może zostać wykorzystany, gdy wartość atrybutu składa się z kilku wyrazów. Wystarczy, że w wartości atrybutu wystąpi podany wyraz, aby dany element został odpowiednio sformatowany. Postać selektora to:

```
selektor[atrybut~="wyraz"] {właściwość: wartość;}
```

lub

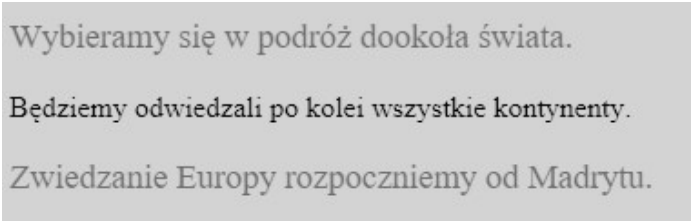
```
[atrybut~="wyraz"] {właściwość: wartość;}
```

Przykład 3.21

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Selektor atrybutu</title>
```

```
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p[title~="jest"] {font-size: 14pt; color: green;}
  </style>
</head>
<body>
  <p title="To jest mój styl">Wybieramy się w podróż dookoła świata.</p>
  <p title="Uwaga na style">Będziemy odwiedzali po kolei wszystkie
kontynenty.</p>
  <p title="To jest mój akapit">Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
</body>
</html>
```

Wynik interpretacji kodu został pokazany na rysunku 3.16.



Rysunek 3.16. Działanie selektora atrybutu zawierającego określony wyraz

Ćwiczenie 3.6

Wykorzystując arkusze stylów oraz selektory atrybutów, zdefiniuj style, które pozwolą na wyświetlenie tekstów umieszczanych na stronie i zapisanych w akapicie w różnych kolorach w zależności od wyrównania tych tekstów na stronie (np. dla tekstów wyrównanych do lewej będzie to kolor niebieski, dla tekstów wyśrodkowanych — kolor zielony, a dla tekstów wyrównanych do prawej — kolor czerwony). Natomiast dla tekstów umieszczonych w znaczniku `<h2>` kolory będą różne w zależności od zastosowanej wielkości czcionki (np. czcionka o rozmiarze `16pt` — kolor żółty, czcionka o rozmiarze `20pt` — kolor granatowy).

3.3.3. Selektory specjalne

Selektor klasy

Selektory pozwalają na nadanie określonych atrybutów takim samym elementom występującym na stronie. Ale czasami chcemy zastosować pewien styl formatowania do jednej grupy elementów, a do drugiej inny. Możemy wtedy dla każdej grupy utworzyć klasę o dowolnej nazwie i zdefiniować styl dla selektora klasy.

Selektor klasy ma postać:

```
selektor.nazwa_klasy {właściwość: wartość;}
```

gdzie *nazwa_klasy* to dowolna nazwa.

Nazwa klasy to pojedynczy wyraz, który musi spełniać kilka warunków:

- Musi zaczynać się od kropki.
- Mogą być użyte litery, cyfry, znak podkreślenia, łącznik.
- Po kropce musi zostać wpisana litera.
- Rozróżniane są małe i wielkie litery.
- Stosowanie polskich znaków nie jest zalecane.
- Nie może zawierać spacji.

Przykład 3.22

```
p.tekst1 {color: red; font-size: 24pt;}
p.tekst2 {color: green; font-size: 20pt;}
p.tekst3 {font-family: courier; color: blue; font-size: 16pt;}
```

Odwołanie do klasy w dokumencie HTML ma postać:

```
<znacznik class="nazwa_klasy"> … </znacznik>
```

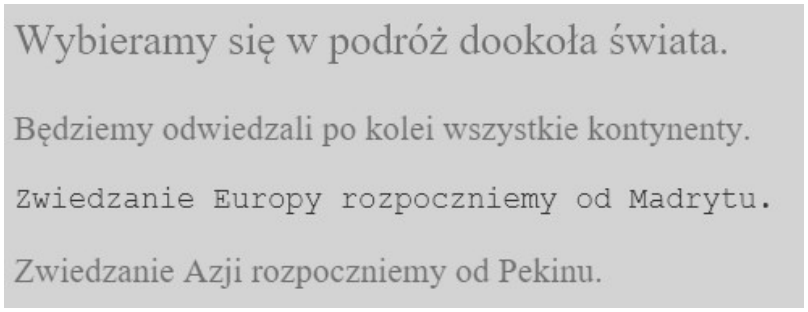
W odwołaniu nazwa klasy jest wartością atrybutu `class`. W nazwie klasy podawanej jako wartość atrybutu nie wstawia się kropki.

Przykład 3.23

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Selektor klasy</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p.tekst1 {color: red; font-size: 20pt;}
  p.tekst2 {color: green; font-size: 16pt;}
  p.tekst3 {font-family: courier; color: blue; font-size: 14pt;}
  </style>
</head>
<body>
  <p class="tekst1">Wybieramy się w podróż dookoła świata.</p>
```

```
  <p class="tekst2">Będziemy odwiedzali po kolei wszystkie kontynenty.</p>
  <p class="tekst3">Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
  <p class="tekst2">Zwiedzanie Azji rozpoczniemy od Pekinu.</p>
</body>
</html>
```

Wynik interpretacji kodu został zilustrowany na rysunku 3.17.



Rysunek 3.17. Działanie selektora klasy

Możliwość tworzenia klas i definiowania dla nich odpowiednich stylów jest jedną z podstawowych właściwości kaskadowych arkuszy stylów. Wykorzystanie tego narzędzia znacznie przyspiesza pracę podczas definiowania stylów dla witryny internetowej. Już podczas projektowania witryny powinno się określić, jakie grupy elementów będą w niej występowały, i dla wszystkich powtarzalnych elementów utworzyć klasy, a następnie umieścić deklaracje tych klas w zewnętrznym arkuszu stylów. Przy ewentualnej modyfikacji wyglądu witryny wystarczy zmodyfikować deklaracje odpowiednich klas w arkuszu stylów.

Uniwersalny selektor klasy

Zdarza się, że definiując klasę, nie określamy, jakich znaczników będzie ona dotyczyła. Tak zdefiniowana klasa może zostać użyta do sformatowania dowolnego elementu bez względu na typ znacznika. Ma ona postać:

```
.nazwa_klasy {właściwość: wartość;}
```

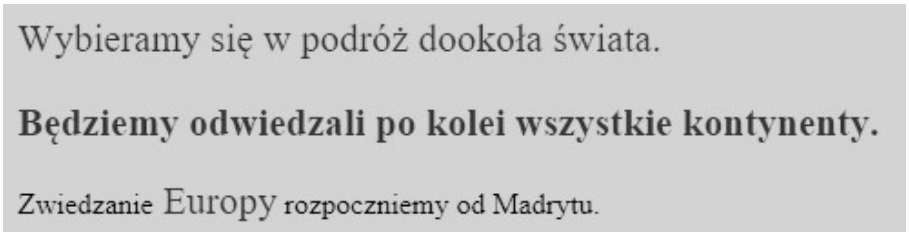
Wszystkie elementy, którym nadamy klasę o podanej nazwie, zostaną sformatowane zgodnie z definicją stylu podaną dla tej klasy.

Przykład 3.24

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Selektor klasy</title>
  <meta charset="UTF-8">
```

```
  <style>
  body {background-color: #c0c0c0;}
  .nad {color: blue; font-size: 16pt;}
  </style>
</head>
<body>
  <p class="nad">Wybieramy się w podróż dookoła świata.</p>
  <h2 class="nad">Będziemy odwiedzali po kolei wszystkie kontynenty.</h2>
  Zwiedzanie<span class="nad"> Europy</span> rozpoczniemy od Madrytu.
</body>
</html>
```

Wynik interpretacji kodu został przedstawiony na rysunku 3.18.



Rysunek 3.18. Efekt zastosowania klasy do formatowania różnych rodzajów znaczników

Deklaracja klasy jest przydatna zwłaszcza wtedy, gdy w różnych miejscach witryny mają występować elementy o takich samych parametrach formatowania, lecz nie możemy (lub nie chcemy) przy ich definiowaniu posługiwać się selektorem typu.

Selektor identyfikatora

Selektor identyfikatora jest stosowany, gdy chcemy nadać określone atrybuty formatowania elementowi, który ma przypisany jednoznaczny identyfikator, czyli występuje raz — w odróżnieniu od klasy.

Selektor identyfikatora ma postać:

```
selektor#identyfikator {właściwość: wartość;}
```

lub

```
#identyfikator {właściwość: wartość;}
```

Deklaracja identyfikatora musi się rozpoczynać znakiem #.

Przykład 3.25

```
h3#nagl3 {color: red;}
```

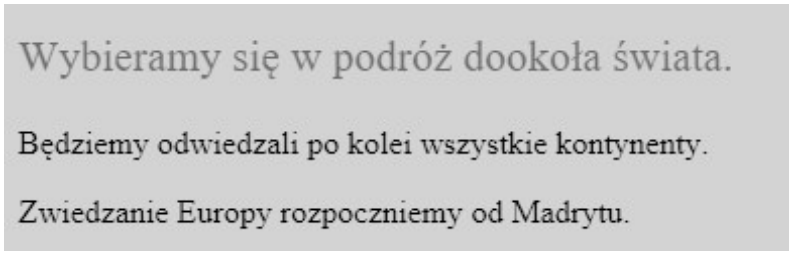
W ten sposób znacznikowi <h3> o id równym nagl3 nadano kolor czerwony. Odwołanie w kodzie do identyfikatora ma postać:

```
<h3 id="nagl3">Podróż dookoła świata</h3>
```

Przykład 3.26

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Selektor identyfikatora</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  #tekst {font-size: 16pt; color: green;}
  </style>
</head>
<body>
  <p id="tekst">Wybieramy się w podróż dookoła świata.</p>
  <p>Będziemy odwiedzali po kolei wszystkie kontynenty.</p>
  <p>Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.19.



Rysunek 3.19. Efekt zastosowania identyfikatora

3.3.4. Pseudoklasy

W języku CSS style są dodawane do elementów lub grup elementów na podstawie ich nazw, atrybutów lub zawartości.

Jest też inna możliwość dodawania stylu. Element nabywa styl lub traci go w związku z działaniem użytkownika lub zmienia się w zależności od umiejscowienia. Przykładem takiego zachowania są linki (*odsyłacze*), które zmieniają swój wygląd po najechaniu

na nie kursorem myszy, ich kliknięciu lub otwarciu linku. Element otrzymuje lub traci pseudoklasę podczas interakcji z użytkownikiem. Do takiej dynamicznej zmiany stylu elementu służą **pseudoklasy**.

Odsyłacze (linki) mają właściwości, które określają działanie użytkownika. Są to:

- `:link` — link nieaktywny, nie został przez użytkownika odwiedzony;
- `:visited` — link odwiedzony, strona była otwierana;
- `:hover` — link gotowy do kliknięcia, kursor myszy ustawiony nad linkiem;
- `:active` — link odwiedzany, strona jest obecnie wczytana.

Odsyłacz podstawowy (pseudoklasa :link)

Odsyłacz podstawowy nadaje atrybuty formatowania wszystkim jeszcze nieodwiedzonym odnośnikom.

```
a:link {właściwość: wartość;}
```

Po zadeklarowaniu klasy można ustawić atrybuty formatowania dla wybranych odsyłaczy, które nie były jeszcze odwiedzane.

```
a.nazwa_klasy:link {właściwość: wartość;}
```

Przy takim zapisie atrybuty formatowania zostaną nadane tylko tym nieodwiedzanym odsyłaczom, które należą do podanej klasy.

Przykład 3.27

```
a:link {color: green; background: yellow;}

a.nowa:link {color: orange;}
```

Odsyłacz odwiedzony (pseudoklasa :visited)

Odsyłacz odwiedzony nadaje atrybuty formatowania odsyłaczowi, który był otwierany, a informacja o tym została umieszczona w pamięci przeglądarki.

```
a:visited {właściwość: wartość;}
```

Z deklaracją klasy ustawiane są atrybuty formatowania dla odsyłaczy wybranej klasy, które były otwierane.

```
a.nazwa_klasy:visited {właściwość: wartość;}
```

Przykład 3.28

```
a:visited {color: red;}

a.po_nowa:visited {color: red;}
```

Wskazanie kursorem myszy (pseudoklasa :hover)

Nadaje atrybuty formatowania odsyłaczowi, gdy myszka znajduje się nad nim, ale nie aktywuje go.

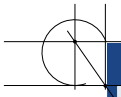
```
a:hover {właściwość: wartość;}
```

Z deklaracją klasy ustawiane są atrybuty formatowania dla odsyłaczy wybranej klasy, które są w danej chwili wskazane myszą.

```
a.nazwa_klasy:hover {właściwość: wartość;}
```

Przykład 3.29

```
a:hover {color: yellow;}
a.po_nowa:hover {color: yellow;}
```



UWAGA

Pseudoklasa `:hover` może być używana do definiowania stylów elementów innych niż odsyłacze.

Przykład 3.30

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Pseudoklasa :hover</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  .blok {height: 25px; width: 100px; background: aqua; color: black;}
  .blok:hover {background: yellow; color: red;}
  </style>
</head>
<body>
  <div class="blok">Europa</div>
</body>
</html>
```

Przykład pokazuje zastosowanie pseudoklasy `:hover` dla znacznika `<div>`.

Odsyłacz aktywny (pseudoklasa :active)

Odsyłacz aktywny nadaje atrybuty formatowania odsyłaczowi, który w danej chwili jest aktywny, np. gdy użytkownik naciśnie i przytrzyma lewy przycisk myszy nad obszarem wybranego elementu, aż do momentu, gdy przycisk ten zostanie puszczony.

```
a:active {właściwość: wartość;}
```

Z deklaracją klasy ustawiane są atrybuty formatowania dla odsyłaczy wybranej klasy, które są w danej chwili aktywne.

```
a.nazwa_klasy:active {właściwość: wartość;}
```

Przykład 3.31

```
a:active {color: blue;}
a.po_nowa:active {color: blue;}
```

Kolejność deklarowania w arkuszu stylów pseudoklas przypisanych do odsyłaczy ma znaczenie dla ich prawidłowego działania.

Poprawna kolejność deklarowania pseudoklas:

```
a:link {właściwość: wartość;}
a:visited {właściwość: wartość;}
a:hover {właściwość: wartość;}
a:active {właściwość: wartość;}
```

Przykład 3.32

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Pseudoklasa odsyłacz</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  a:link {color: green;}
  a:visited {color: red;}
  a:hover {color: yellow;}
  a:active {color: blue;}
  </style>
</head>
<body>
```

```
  <p>Link do strony wydawnictwa: <a href="http://helion.pl/">HELION</a></p>
</body>
</html>
```

Podany kod działa następująco: odnośnik znajdujący się pod tekstem HELION po otwarciu strony jest wyświetlany w kolorze zielonym (pseudoklasa `:link`), po najechaniu myszą na tekst jego kolor zmienia się na żółty (pseudoklasa `:hover`), po wciśnięciu i przytrzymaniu klawisza myszy kolor tekstu zmienia się na niebieski (pseudoklasa `:active`), a po odwiedzeniu strony kolor tekstu zmieni się na czerwony (pseudoklasa `:visited`).

Pseudoklasa :focus

Nadaje atrybuty formatowania odsyłaczowi (wcześniej wybranemu) lub polu formularza, na którym został ustawiony kursor. Na przykład wtedy, gdy odsyłacz został wybrany za pośrednictwem klawisza *Tab* lub w polu formularza znalazł się kursor.

```
selektor:focus {właściwość: wartość;}
```

Ćwiczenie 3.7

Wykorzystując arkusze stylów, zdefiniuj pseudoklasy, które umożliwią dynamiczną zmianę stylu linków znajdujących się na stronie internetowej zaprojektowanej w poprzednich ćwiczeniach. Link podstawowy powinien mieć kolor czerwony, odwiedzany — kolor zielony, wskazany przez kursor myszy — kolor pomarańczowy, a link aktywny — kolor żółty.

3.3.5. Selektory pseudoelementów

W języku HTML nie ma mechanizmów dostępu do takich elementów strony jak pierwsza litera lub pierwsza linia akapitu bez otaczania ich znacznikami. Do formatowania tych specjalnych elementów można wykorzystać **pseudoelementy** dostępne w języku CSS.

Pierwsza linia (:first-line)

Pseudoelement `:first-line` nadaje określone formatowanie wszystkim pierwszym liniom znacznika, do którego odnosi się selektor.

```
selektor:first-line {właściwość: wartość;}
```

Selektorem może być dowolny znacznik języka HTML.

Przykład 3.33

```
p:first-line {color: red; font-size: 16pt;}
```

Pierwsza litera (:first-letter)

Pseudoelement `:first-letter` nadaje odrębne formatowanie pierwszej literze np. akapitu. Służy głównie do poprawienia wyglądu tekstu poprzez zaprojektowanie ozdobnej pierwszej litery (inicjału).

```
selektor:first-letter {właściwość: wartość;}
```

Przykład 3.34

```
p:first-letter {font-family: arial; font-size: 24pt; color: blue;}
```

3.3.6. Grupowanie selektorów

Jeżeli w arkuszu stylów zostały zdefiniowane elementy o tym samym stylu, to można te elementy połączyć w grupy i nadać im styl poprzez wspólną deklarację.

Przykład 3.35

```
p {color: green;}
h2 {color: green;}
h3 {color: green;}
```

Ten ciąg deklaracji można zastąpić jedną wspólną definicją stylu.

Przykład 3.36

```
p, h2, h3 {color: green;}
```

Kolejne selektory przypisane do stylu powinny zostać oddzielone przecinkami.

3.4. Właściwości elementów

W kaskadowych arkuszach stylów formatowanie elementów strony jest realizowane przez ustawianie właściwości tych elementów.

3.4.1. Czcionki

Czcionki są najistotniejszym elementem strony, który podlega formatowaniu. Za pomocą właściwości czcionki można określić rodzinę czcionki (`font-family`), jej styl (`font-style`), rozmiar (`font-size`) i pogrubienie (`font-weight`).

Rodzaj czcionki

Ze względu na to, że nie wszystkie czcionki są dostępne na komputerze użytkownika, przy wyborze rodzaju czcionki określa się jej preferowany typ. Można również zdefiniować kilka rodzajów czcionek. Jeżeli na komputerze użytkownika nie ma pierwszego rodzaju czcionki, to z listy zostanie wybrany kolejny.

Rodzaj czcionki ustala się za pomocą atrybutu `font-family`.

```
selektor {font-family: rodzaj1, rodzaj2, rodzaj3,…}
```

Definiując rodzaj czcionki, możemy podać kilka nazw własnych (np. *Arial*) oraz nazwy rodziny (np. *serif*). Podczas renderowania strony przeglądarka spróbuje użyć pierwszej z czcionek. Jeżeli nie zostanie ona znaleziona na komputerze użytkownika, zostanie podjęta próba z kolejnymi czcionkami. Przy wprowadzaniu kilku rodzajów czcionek należy ich nazwy oddzielić przecinkiem i spacją. Jeżeli nazwa czcionki składa się z kilku wyrazów, całą nazwę ujmujemy w apostrofy, np. `'Times New Roman'`.

Zamiast nazwy czcionki można podać nazwę rodziny czcionek. Rozwiązanie to warto stosować równolegle z nazwami własnymi. Dzięki temu w przypadku gdy przeglądarka nie znajdzie żadnej z zadeklarowanych czcionek, zostanie podjęta próba dopasowania innej czcionki z danej rodziny.

Nazwy rodzin czcionek to:

- `serif` — czcionki szeryfowe, mające wykończenie liter i cyfr. Do tej grupy należą m.in.: Times New Roman, Georgia, Bodoni.
- `sans-serif` — czcionki bezszeryfowe. Końcówki znaków są proste; np.: Arial, Verdana, Futura.
- `monospace` — czcionki monotypiczne. Znaki mają stałą szerokość; np. Courier, Courier New.
- `cursive` — czcionki mające cechy czcionki pochyłej. Znaki wyglądają jak pisane ręcznie; np. Comic Sans, Florence.
- `fantasy` — czcionki dekoracyjne; np. Impact, OldTown.

Przykład 3.37

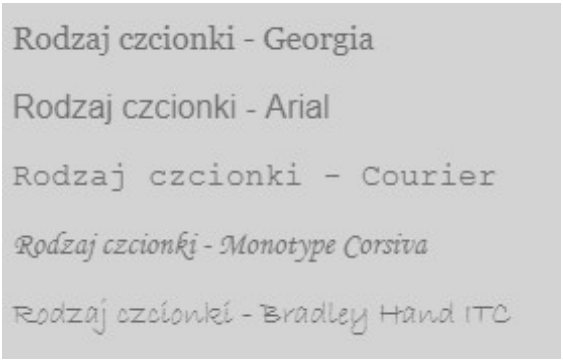
```
<p style="font-family: Georgia, Courier, Arial">Arkusze stylów</p>
<p style="font-family: Georgia, Courier, serif">Arkusze stylów</p>
```

Przykład 3.38

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Rodziny czcionek</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p {font-size: 18pt; color: green;}
  </style>
```

```
</head>
<body>
  <p style="font-family: Georgia;">Rodzaj czcionki - Georgia</p>
  <p style="font-family: Arial;">Rodzaj czcionki - Arial</p>
  <p style="font-family: Courier;">Rodzaj czcionki - Courier</p>
  <p style="font-family: 'Monotype Corsiva';">Rodzaj czcionki - Monotype
Corsiva</p>
  <p style="font-family: 'Bradley Hand ITC';">Rodzaj czcionki - Bradley Hand
ITC</p>
</body>
</html>
```

Wynik interpretacji kodu został pokazany na rysunku 3.20.



Rysunek 3.20. Zastosowanie różnych rodzajów czcionek

Rozmiar czcionki

Wielkość czcionki jest definiowana w atrybucie `font-size`.

```
selektor {font-size: rozmiar;}
```

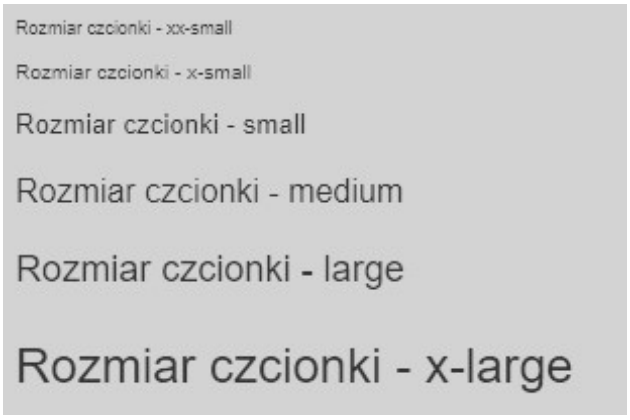
Wielkość czcionki można ustawiać na cztery sposoby:

1. **Według słów kluczowych**, które pozwalają określić rozmiar czcionki względem rozmiaru podstawowego. W języku CSS występuje siedem słów kluczowych:
   - `xx-small` — czcionka najmniejsza,
   - `x-small` — czcionka mniejsza,
   - `small` — czcionka mała,
   - `medium` — czcionka średnia (jest to podstawowy rozmiar czcionki),
   - `large` — czcionka duża,
   - `x-large` — czcionka większa,
   - `xx-large` — czcionka największa.

Przykład 3.39

```
<!DOCTYPE html>
<html lang="pl-PL" >
<head>
  <title>Rozmiar czcionki</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p {color: blue; font-family: Arial;}
  </style>
</head>
<body>
  <p style="font-size: xx-small;">Rozmiar czcionki - xx-small</p>
  <p style="font-size: x-small;">Rozmiar czcionki - x-small</p>
  <p style="font-size: small;">Rozmiar czcionki - small</p>
  <p style="font-size: medium;">Rozmiar czcionki - medium</p>
  <p style="font-size: large;">Rozmiar czcionki - large</p>
  <p style="font-size: x-large;">Rozmiar czcionki - x-large</p>
</body>
</html>
```

Wynik interpretacji kodu został zilustrowany na rysunku 3.21.



Rysunek 3.21. Zastosowanie różnych rozmiarów czcionki

2. **Za pomocą wartości względnych** — wielkość czcionki jest ustalana względem wielkości czcionki elementu nadrzędnego:
   - `smaller` — czcionka mniejsza od bieżącej,
   - `larger` — czcionka większa od bieżącej.

Przykład 3.40

```
<p style="font-size: smaller;">Wybieramy się w podróż dookoła świata.</p>
<p style="font-size: larger;">Będziemy odwiedzali po kolei wszystkie
kontynenty.</p>
```

3. **Poprzez wielkość podaną w jednostkach miary**. Jednostki, które mogą zostać użyte do określenia wielkości czcionki, to:
   - *piksele* (`px`),
   - *punkty* (`pt`),
   - *centymetry* (`cm`),
   - *cale* (`in`),
   - *milimetry* (`mm`),
   - *pica* (`pc`),
   - *em* (`em`).

Przykład 3.41

```
<p style="font-size: 12pt;">Wybieramy się w podróż dookoła świata.</p>
<p style="font-size: 1cm;">Będziemy odwiedzali po kolei wszystkie kontynenty.
</p>
```

4. **Poprzez wielkość podaną w procentach**. Każda przeglądarka ma zdefiniowany podstawowy rozmiar tekstu. Wartości procentowe określają wielkość czcionki w stosunku do rozmiaru podstawowego.

Przykład 3.42

```
<p style="font-size: 120%;">Wybieramy się w podróż dookoła świata.</p>
<p style="font-size: 75%;">Będziemy odwiedzali po kolei wszystkie kontynenty.
</p>
```

5. **Poprzez zastosowanie responsywnego rozmiaru czcionki**. Rozmiar tekstu można ustawić za pomocą jednostki *vw* (ang. *viewport width*). W ten sposób rozmiar tekstu widocznego na stronie będzie związany z rozmiarem okna przeglądarki. Jednostka *vw* oznacza 1% aktualnego rozmiaru szerokości okna, w którym wyświetlana jest zawartość strony.

Przykład 3.43

```
<p style="font-size: 16;">Wybieramy się w podróż dookoła świata.</p>
<p style="font-size: 10vw;">Będziemy odwiedzali po kolei wszystkie
kontynenty.</p>
```

Jeżeli rozmiar czcionki nie zostanie określony, to na stronie zastosowany zostanie najprawdopodobniej domyślny rozmiar zwykłego tekstu, czyli 16 pikseli.

Styl czcionki

Styl czcionki jest definiowany za pomocą atrybutu `font-style`.

```
selektor {font-style: styl;}
```

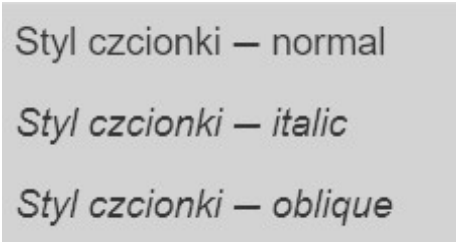
Dostępne są trzy style:

- `normal` — czcionka podstawowa,
- `italic` — czcionka pochylona,
- `oblique` — czcionka pochylona (podobna do poprzedniej, może być wygenerowana przez pochylenie zwykłej czcionki; jeżeli styl czcionki `italic` jest niedostępny, automatycznie wybierany jest styl `oblique`).

Przykład 3.44

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Styl czcionki</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p {font-size: 14pt; color: blue; font-family: Arial;}
  </style>
</head>
<body>
  <p>Styl czcionki - normal</p>
  <p style="font-style: italic;">Styl czcionki - italic</p>
  <p style="font-style: oblique;">Styl czcionki - oblique</p>
</body>
</html>
```

Wynik interpretacji kodu został pokazany na rysunku 3.22.



Rysunek 3.22. Zastosowane różne style czcionki

Wariant czcionki

Atrybut `font-variant` pozwala wybrać między czcionką normalną a kapitalikami (wyświetlanie wielkich liter o rozmiarze czcionki przeznaczonym dla małych liter).

```
selektor {font-variant: wartość;}
```

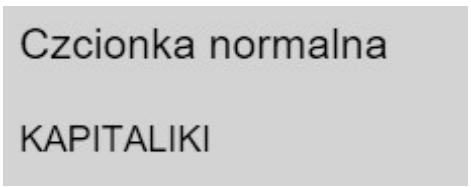
Atrybut `font-variant` może przyjmować wartości:

- `normal` — czcionka normalna,
- `small-caps` — kapitaliki.

Przykład 3.45

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Kapitaliki</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p {font-size: 16pt; font-family: Arial;}
  </style>
</head>
<body>
  <p>Czcionka normalna</p>
  <p style="font-variant: small-caps;">kapitaliki</p>
</body>
</html>
```

Wynik interpretacji kodu został pokazany na rysunku 3.23.



Rysunek 3.23. Czcionka normalna i kapitaliki

Waga czcionki

Za pomocą atrybutu `font-weight` można określić wagę czcionki:

```
selektor {font-weight: wartość;}
```

Waga czcionki określana jest następującymi wartościami:

- `normal` — czcionka normalna,
- `bold` — czcionka pogrubiona,
- `100`, `200`, `300`, `400` (równoważne z `normal`), `500`, `600`, `700` (równoważne z `bold`), `800`, `900` — określają wagę czcionki. Nie każda czcionka ma wszystkie dziewięć stopni grubości.

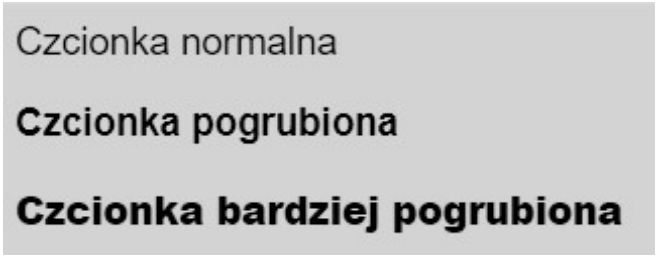
Można również używać wartości względnych:

- `lighter` — czcionka mniej wytłuszczona od przypisanej domyślnie,
- `bolder` — czcionka bardziej wytłuszczona od przypisanej domyślnie.

Przykład 3.46

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Kapitaliki</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
  p {font-size: 14pt; font-family: Arial;}
  </style>
</head>
<body>
  <p style="font-weight: normal;">Czcionka normalna</p>
  <p style="font-weight: bold;">Czcionka pogrubiona</p>
  <p style="font-weight: 900;">Czcionka bardziej pogrubiona</p>
</body>
</html>
```

Wynik interpretacji kodu został przedstawiony na rysunku 3.24.



Rysunek 3.24. Czcionka normalna i czcionka pogrubiona

Atrybuty czcionki

Podczas definiowania formatu czcionki można użyć następujących atrybutów:

- `font-style`,
- `font-variant`,
- `font-weight`,
- `font-size`,
- `font-family`.

Często czcionce przypisywanych jest wiele atrybutów równocześnie. W takim przypadku można użyć wspólnego polecenia zawierającego wszystkie atrybuty dotyczące czcionek:

```
selektor {font: wartości atrybutów;}
```

Jako wartości atrybutów powinny zostać podane konkretne wartości, które będą definiowane dla czcionek. Należy oddzielić je od siebie spacjami. Wartości atrybutów powinny zostać wpisane w podanej wyżej kolejności. Zmiana kolejności może spowodować złą interpretację atrybutu lub jego pominięcie. Niektóre z wartości atrybutów mogą zostać pominięte, ale wartości dla atrybutów `font-size` i `font-family` są wymagane.

Przykład 3.47

```
<p style="font: italic bold 12pt Georgia">
```

Ćwiczenie 3.8

Wykorzystując arkusze stylów, zdefiniuj style określające właściwości czcionki (rodzaj, rozmiar, styl) dla znaczników: `<h2>`, `<h3>`, `<p>` na Twojej stronie internetowej.

3.4.2. Tekst

Język CSS pozwala na dowolne formatowanie tekstu poprzez dodanie do niego stylu. Umożliwia to nie tylko szybką zmianę wyglądu tekstu na stronie, ale także swobodne manipulowanie tekstem i grafiką umieszczaną w tekście.

Wcięcie tekstu

Przy użyciu atrybutu `text-indent` można definiować wcięcie pierwszego wiersza akapitu.

```
selektor {text-indent: wartość;}
```

Jako wartość należy podać wielkość wcięcia.

Przykład 3.48

```
p {text-indent: 20px;}
```

Wszystkie akapity oznaczone znacznikiem `<p>` będą miały w pierwszym wierszu wcięcie równe `20px`.

Wyrównanie tekstu

Wyrównanie tekstu definiujemy za pomocą atrybutu `text-align`. Odpowiada on atrybutowi `align` w języku HTML.

```
selektor {text-align: wartość;}
```

Wyrównanie określa się za pomocą wartości:

- `left` — wyrównanie tekstu do lewego marginesu,
- `right` — wyrównanie tekstu do prawego marginesu,
- `center` — wyśrodkowanie tekstu,
- `justify` — rozłożenie tekstu równomiernie między marginesami.

Przykład 3.49

```
p {text-align: justify;}
h2 {text-align: right;}
h3 {text-align: center;}
```

Odstęp między wierszami tekstu

Do określenia wielkości odstępu między wierszami tekstu zapisanego w bloku służy atrybut `line-height`.

```
selektor {line-height: odstęp;}
```

Jako wartość atrybutu należy podać konkretną wielkość odstępu. Może to być:

- *liczba* — określa odstęp jako wielokrotność obecnego rozmiaru czcionki;
- *wysokość* — jest podawana z jednostką miary: `px`, `pt`, `cm` itp.; określa stałą wysokość odstępu;
- *wartość procentowa* — określa odstęp jako wartość procentową obecnego rozmiaru czcionki.

Przykład 3.50

```
p {line-height: 0.8;}
```

Ozdabianie tekstu

Do dodawania efektów, takich jak podkreślenie, przekreślenie tekstu, umieszczenie linii nad tekstem, używany jest atrybut `text-decoration`.

```
selektor {text-decoration: wartość;}
```

Rodzaj efektu określa się za pomocą wartości:

- `none` — bez zmian,
- `underline` — tekst podkreślony,

- `overline` — linia umieszczona nad tekstem,
- `line-through` — tekst przekreślony,
- `blink` — migotanie tekstu (nie wszystkie przeglądarki interpretują ten efekt).

Przykład 3.51

```
p {text-decoration: underline;}
h2 {text-decoration: overline;}
h3 {text-decoration: line-through;}
```

Podane wartości można łączyć w jednej deklaracji.

Przykład 3.52

```
p {text-decoration: underline line-through overline;}
```

Odstęp między literami

Do definiowania odstępu między literami tekstu stosuje się atrybut `letter-spacing`.

```
selektor {letter-spacing: wartość;}
```

Przykład 3.53

```
p {letter-spacing: 3px;}
```

Odstęp między wyrazami

Do definiowania odstępu między wszystkimi wyrazami tekstu stosuje się atrybut `word-spacing`.

```
selektor {word-spacing: wartość;}
```

Przykład 3.54

```
p {word-spacing: 12px;}
```

Transformacja tekstu

Do ustawiania i zmiany wielkości liter tekstu służy atrybut `text-transform`. Atrybut ten kontroluje wielkość liter w tekście i dokonuje ich transformacji.

```
selektor {text-transform: wartość;}
```

Atrybut może przyjmować wartości:

- `capitalize` — zamienia na wielkie litery wszystkie pierwsze litery wyrazów w tekście, np. słowa „jan kowalski” zamieni na „Jan Kowalski”.
- `uppercase` — zamienia wszystkie litery tekstu na wielkie, np. słowa „jan kowalski” zamieni na „JAN KOWALSKI”.

- `lowercase` — zamienia wszystkie litery tekstu na małe, np. słowa „JAN KOWALSKI” zamieni na „jan kowalski”.
- `none` — bez transformacji. Tekst jest wyświetlany na stronie tak, jak został umieszczony w kodzie.

Przykład 3.55

```
h3 {text-transform: capitalize;}
h2 {text-transform: uppercase;}
```

Białe znaki

Przy użyciu atrybutu `white-space` można sterować wyświetlaniem na stronie spacji oraz innych białych znaków. Wstawienie w kodzie kilku spacji obok siebie spowoduje ich zamianę na pojedynczą spację. Również wstawienie spacji na początku tekstu jest niemożliwe. Atrybut `white-space` pozwoli pozbyć się tych niedogodności.

```
selektor {white-space: wartość;}
```

Atrybut może przyjmować wartości:

- `normal` — zamienia kilka sąsiednich spacji na jedną. Wiersze tekstu są przełamywane automatycznie.
- `pre` — umożliwia wyświetlanie kilku spacji obok siebie. Wiersze tekstu są przełamywane w miejscu wystąpienia znaku nowej linii (*Enter*).
- `nowrap` — zamienia kilka sąsiednich spacji na jedną. Wiersze tekstu są przełamywane w miejscu wystąpienia znacznika `<br>`.
- `pre-wrap` — umożliwia wyświetlanie kilku spacji obok siebie. Wiersze tekstu są przełamywane w miejscu wystąpienia znaku nowej linii (*Enter*) oraz automatycznie na końcu szerokości strony.
- `pre-line` — zamienia kilka sąsiednich spacji na jedną. Wiersze tekstu są przełamywane w miejscu wystąpienia znaku nowej linii (*Enter*) oraz automatycznie na końcu szerokości strony.

Nie wszystkie przeglądarki prawidłowo interpretują ten atrybut.

Przykład 3.56

```
<!DOCTYPE html>
<html lang="pl-PL" >
<head>
  <title>Białe znaki</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #c0c0c0;}
```

```
  p {white-space: pre-line;}
  </style>
</head>
<body>
  <p>
  Wybieramy się w podróż dookoła świata.
  Będziemy odwiedzali po kolei wszystkie kontynenty.
  Zwiedzanie Europy rozpoczniemy od Madrytu. Zwiedzanie Azji rozpoczniemy od
Pekinu. Najciekawszym krajem w Afryce jest Kenia. Australię planujemy zwiedzać
na końcu.
  </p>
</body>
</html>
```

Wynik interpretacji kodu jest widoczny na rysunku 3.25.

Wybieramy się w podróż dookoła świata.
Będziemy odwiedzali po kolei wszystkie kontynenty.
Zwiedzanie Europy rozpoczniemy od Madrytu. Zwiedzanie Azji rozpoczniemy od Pekinu. Najciekawszym krajem w Afryce jest Kenia. Australię planujemy zwiedzać na końcu.

Rysunek 3.25. Tekst został przełamany w miejscu wystąpienia znaku Enter. Nie ma powtarzających się spacji

3.4.3. Tło i kolor

W języku CSS kolor tekstu oraz właściwości tła można definiować dla różnych elementów z użyciem stylów.

Kolor

Korzystając z atrybutu `color`, można opisać pierwszoplanowy kolor wybranego elementu.

```
selektor {color: wartość;}
```

Przykład 3.57

```
h3 {color: #f0f0f0;}
body {color: green;}
```

Kolor tła

Kolor tła dla elementu jest definiowany za pomocą atrybutu `background-color`.

```
selektor {background-color: wartość;}
```

Przykład 3.58

```
body {background-color: #c0c0c0;}
p {background-color: lightblue;}
<p style="background-color: yellow;">Tło koloru żółtego</p>
h2 {color: green; background-color: #c0c0c0;}
```

Grafika jako tło

Do umieszczenia obrazka jako tła służy atrybut `background-image`.

```
selektor {background-image: wartość;}
```

Wartość atrybutu jest nazwą pliku, w którym została zapisana grafika.

Przykład 3.59

```
body {background-color: #f0f0f0; background-image: url("tlo.jpg");}
```

W podanym przykładzie obrazek został zlokalizowany w tym samym folderze co arkusz stylów. Gdy odwołujemy się do obrazka znajdującego się w innej lokalizacji, musimy podać pełną ścieżkę do pliku.

Powtarzanie grafiki w tle

Przy powtarzaniu obrazka w tle można określić sposób jego powtarzania, stosując atrybut `background-repeat`.

```
selektor {background-repeat: wartość;}
```

Jako wartość powtarzania można wybrać:

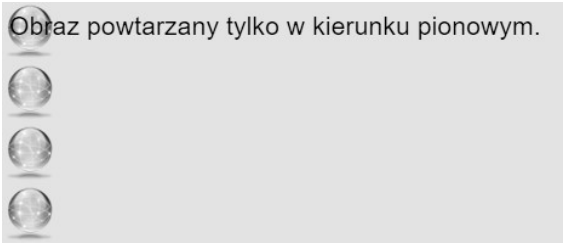
- `repeat` — powtarzanie tła w obu kierunkach,
- `repeat-x` — powtarzanie tła tylko w kierunku poziomym,
- `repeat-y` — powtarzanie tła tylko w kierunku pionowym,
- `no-repeat` — brak powtarzania tła.

Przykład 3.60

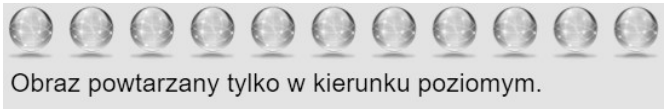
```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Grafika w tle</title>
  <meta charset="UTF-8">
  <style>
  body {
    background-color: #cad8f3;
```

```
    background-image: url("obraz.png");
    background-repeat: repeat-y;
  }
  p {
    font-size: 16pt;
    font-family: Arial;
  }
  </style>
</head>
<body>
  <p>Obraz powtarzany tylko w kierunku pionowym.</p>
</body>
</html>
```

Wynik powtarzania tła tylko w kierunku pionowym został pokazany na rysunku 3.26. Natomiast wynik powtarzania tła tylko w kierunku poziomym — na rysunku 3.27.



Rysunek 3.26. Wstawiony obraz powtarzany w pionie



Rysunek 3.27. Wstawiony obraz powtarzany w poziomie

Blokada grafiki w tle

Jeżeli treść strony jest przewijana za pomocą suwaka, to wstawiona jako tło strony grafika przesuwa się razem z tekstem. Aby grafika w trakcie takich działań była nieruchoma, należy użyć atrybutu `background-attachment`.

```
selektor {background-attachment: blokada;}
```

Jako wartość blokady można wybrać:

- `scroll` — przewijanie tła,
- `fixed` — tło nieruchome względem okna przeglądarki,
- `local` — tło nieruchome względem elementu, dla którego zostało zdefiniowane.

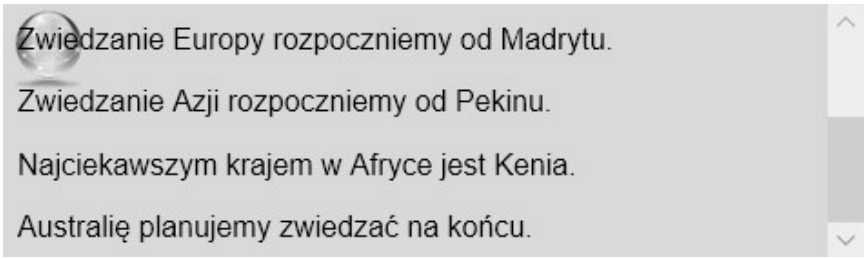
Przykład 3.61

```
body {background-color: #ffccff; background-image: url("obraz.png");
background-attachment: fixed;}
```

Przykład 3.62

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Pozycja tła</title>
  <meta charset="UTF-8">
  <style>
  body {
    background-color: #c0cfc0;
    background-image: url("obraz.png");
    background-repeat: no-repeat;
    background-attachment: fixed;
  }
  p {
    font-size: 12pt;
    font-family: Arial;
    white-space: pre-wrap;
  }
  </style>
</head>
<body>
  <p>Obraz został zablokowany.</p>
  <p>Wybieramy się w podróż dookoła świata.</p>
  <p>Będziemy odwiedzali po kolei wszystkie kontynenty.</p>
  <p>Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
  <p>Zwiedzanie Azji rozpoczniemy od Pekinu.</p>
  <p>Najciekawszym krajem w Afryce jest Kenia.</p>
  <p>Australię planujemy zwiedzać na końcu.</p>
</body>
</html>
```

Wynik wykonania kodu został zilustrowany na rysunku 3.28.



Rysunek 3.28. Grafika tła unieruchomiona. Suwak okna przesunięty w dół

Ćwiczenie 3.9

Dla utworzonej strony internetowej umieść grafikę jako tło strony lub jako tło umieszczonego na niej elementu. Sprawdź działanie atrybutu `background-attachment` dla wartości: `scroll`, `fixed` i `local`.

Pozycja tła

Grafika wstawiona na stronę zostanie domyślnie umieszczona w lewym górnym rogu ekranu. Do zmiany standardowych ustawień i pozycjonowania grafiki w dowolnym miejscu na stronie służy atrybut `background-position`.

```
selektor {background-position: pozycja;}
```

Wartość atrybutu może być ustalana w różny sposób. Można określać położenie grafiki na stronie, podając jedną wartość, można to zrobić za pomocą dwóch parametrów, można też zapisać jej położenie jako wartość procentową szerokości ekranu.

Niżej zostały podane wartości parametru *pozycja*.

Jedna wartość:

- `center` — grafika zostanie umieszczona na środku strony;
- `left` — grafika zostanie umieszczona z lewej strony;
- `right` — grafika zostanie umieszczona z prawej strony;
- `top` — grafika zostanie umieszczona na górze strony;
- `bottom` — grafika zostanie umieszczona na dole strony;
- `jednostka długości` — grafika zostanie umieszczona w określonej odległości od lewego marginesu.

Dwie wartości:

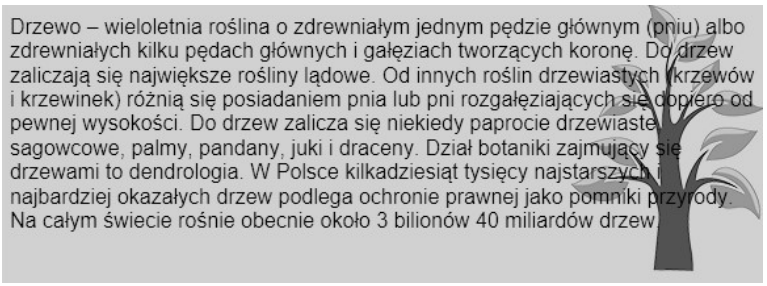
- `left top` — grafika zostanie umieszczona w lewym górnym rogu strony;
- `left bottom` — grafika zostanie umieszczona w lewym dolnym rogu strony;
- `right top` — grafika zostanie umieszczona w prawym górnym rogu strony;
- `right bottom` — grafika zostanie umieszczona w prawym dolnym rogu strony.

Definiowanie tych parametrów ma sens, jeżeli ograniczymy powtarzanie grafiki na stronie (mogą być użyte parametry powtarzania: `no-repeat`, `repeat-x`, `repeat-y`).

Przykład 3.63

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Pozycja tła</title>
  <meta charset="UTF-8">
  <style>
  body {
    background-color: #f5b270;
    background-image: url("drzewo1.png");
    background-repeat: no-repeat;
    background-position: right top;
  }
  p {
    font-size: 12pt;
    font-family: Arial;
  }
  </style>
</head>
<body>
  <p>
  Drzewo – wieloletnia roślina o zdrewniałym jednym pędzie głównym (pniu) albo zdrewniałych kilku pędach głównych i gałęziach tworzących koronę.
  Do drzew zaliczają się największe rośliny lądowe. Od innych roślin drzewiastych (krzewów i krzewinek) różnią się posiadaniem pnia lub pni rozgałęziających się dopiero od pewnej wysokości.
  Do drzew zalicza się niekiedy paprocie drzewiaste, sagowcowe, palmy, pandany, juki i draceny.
  Dział botaniki zajmujący się drzewami to dendrologia.
  W Polsce kilkadziesiąt tysięcy najstarszych i najbardziej okazałych drzew podlega ochronie prawnej jako pomniki przyrody.
  Na całym świecie rośnie obecnie około 3 bilionów 40 miliardów drzew.
  </p>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.29.



Rysunek 3.29. Grafika została umieszczona w prawym górnym rogu strony

Usunięcie tła

Jeżeli na stronie (lub na kilku stronach) nie chcemy stosować wcześniej zdefiniowanego tła dla stron lub elementów, to najprostszym rozwiązaniem jest niedołączanie do tych stron zewnętrznego arkusza stylów. Takie rozwiązanie nie zawsze jest dobre, ponieważ inne deklaracje znajdujące się w zewnętrznym arkuszu stylów najczęściej są wykorzystywane na stronie. Lepszym rozwiązaniem jest zachowanie odnośnika do zewnętrznego arkusza stylów, a dodatkowo w wewnętrznym arkuszu stylów lub w stylu `inline` usunięcie tła dla danej strony. Można do tego użyć polecenia:

```
selektor {background: none;}
```

Polecenie to usuwa wszystkie atrybuty dotyczące tła.

Podobną metodą można usunąć tylko wybrane atrybuty tła. Należy wtedy w poleceniu użyć nazwy atrybutu, który zostanie usunięty.

Atrybuty tła

Możliwe jest również zdefiniowanie wszystkich właściwości tła przy użyciu jednego atrybutu `background`.

```
selektor {background: wartości_atrybutów;}
```

Jako wartości atrybutów trzeba podać konkretne wartości oddzielone spacjami. W ten sposób można zdefiniować następujące atrybuty tła:

- kolor,
- tło obrazkowe,
- powtarzanie,
- blokadę tła,
- pozycję.

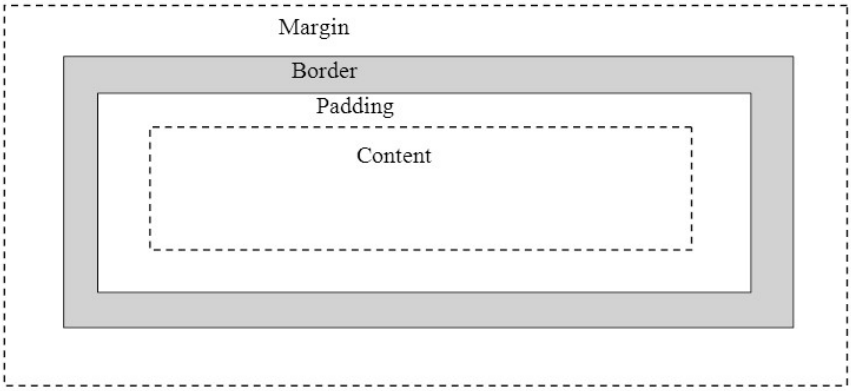
Nie trzeba podawać pełnej listy atrybutów. Opuszczonemu atrybutowi zostanie przypisana wartość domyślna. Nie wszystkie przeglądarki akceptują tę skróconą formę deklaracji atrybutów tła i wtedy trzeba podawać ich pełną deklarację.

3.5. Model blokowy CSS

Wszystkie elementy projektowane w języku HTML można traktować jak prostokątne bloki odpowiednio sformatowane i rozmieszczone na stronie internetowej. Każdy z takich bloków ma marginesy, obramowania, wypełnienie i zawartość.

Model blokowy został zamieszczony na rysunku 3.30.

Rysunek 3.30. Model blokowy CSS



- ***Margin*** — margines wokół ramki (margines zewnętrzny). Jest to pusty obszar wokół ramki, nie ma koloru tła i jest przezroczysty.
- ***Border*** — obramowanie wokół zawartości elementu; ma styl i kolor.
- ***Padding*** — odstęp między obramowaniem i zawartością elementu (margines wewnętrzny).
- ***Content*** — zawartość elementu, np. teksty lub obrazy.

Każdy element w modelu blokowym jest opisywany przez trzy atrybuty: *padding*, *margin* i *border*. Podane atrybuty tworzą układ graficzny strony internetowej.

3.5.1. Obramowanie (border)

Obramowanie to ramka (ang. *border*) narysowana wokół elementu. Obramowanie może być wykorzystywane do dekoracji elementu lub do oddzielenia go od innych elementów. Można je definiować dla jednej lub dla wielu krawędzi bloku.

Szerokość obramowania

Do ustawienia szerokości obramowania służy atrybut `border-width`. Przy jego użyciu można definiować obramowanie jednakowe dla wszystkich krawędzi lub dla każdej z osobna.

```
selektor {border-width: wartości;}
```

Możliwe ustawienia atrybutu:

- *jedna wartość* — jednakowa szerokość dla wszystkich krawędzi;
- *dwie wartości* — taka sama szerokość dla krawędzi poziomych i taka sama szerokość dla krawędzi pionowych;

- *trzy wartości* — pierwsza wartość określa szerokość krawędzi górnej, druga — jednocześnie szerokość dwóch krawędzi pionowych, trzecia — szerokość krawędzi dolnej;
- *cztery wartości* — każda wartość określa osobno szerokość kolejnych krawędzi (górnej, prawej, dolnej, lewej).

Szerokość krawędzi może być definiowana w dowolnych jednostkach lub za pomocą trzech wartości:

- `thin` — cienkie obramowanie,
- `medium` — średnie obramowanie,
- `thick` — grube obramowanie.

Możliwe jest również definiowanie szerokości pojedynczych krawędzi. Służą do tego atrybuty przypisane do poszczególnych krawędzi.

Atrybut `border-top-width` definiuje szerokość górnej krawędzi.

```
selektor {border-top-width: szerokość;}
```

Atrybut `border-bottom-width` definiuje szerokość dolnej krawędzi.

```
selektor {border-bottom-width: szerokość;}
```

Atrybut `border-left-width` definiuje szerokość lewej krawędzi.

```
selektor {border-left-width: szerokość;}
```

Atrybut `border-right-width` definiuje szerokość prawej krawędzi.

```
selektor {border-right-width: szerokość;}
```

Przykład 3.64

```
p {border-width: 1px;}
div {border-bottom-width: 20px;}
h3 {border-width: thick;}
```

Styl obramowania

Do definiowania stylu obramowania służy atrybut `border-style`. Za jego pomocą można definiować styl obramowania jednakowy dla wszystkich krawędzi lub dla każdej inny.

```
selektor {border-style: wartości;}
```

Podobnie jak w przypadku atrybutu `border-width`, możliwe są następujące ustawienia atrybutu:

- *jedna wartość* — jednakowy styl dla wszystkich krawędzi;
- *dwie wartości* — taki sam styl dla krawędzi poziomych i taki sam styl dla krawędzi pionowych;

- *trzy wartości* — pierwsza wartość określa styl krawędzi górnej, druga — jednocześnie styl dwóch krawędzi pionowych, trzecia — styl krawędzi dolnej;
- *cztery wartości* — każda wartość określa osobno styl kolejnych krawędzi (górnej, prawej, dolnej, lewej).

Możliwe jest, jak w przypadku atrybutu `border-width`, definiowanie stylu pojedynczych krawędzi. Służą do tego atrybuty przypisane do poszczególnych krawędzi.

Atrybut `border-top-style` definiuje styl górnej krawędzi.

```
selektor {border-top-style: styl;}
```

Atrybut `border-bottom-style` definiuje styl dolnej krawędzi.

```
selektor {border-bottom-style: styl;}
```

Atrybut `border-left-style` definiuje styl lewej krawędzi.

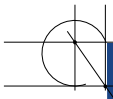
```
selektor {border-left-style: styl;}
```

Atrybut `border-right-style` definiuje styl prawej krawędzi.

```
selektor {border-right-style: styl;}
```

Wartości, które mogą być przypisywane do stylów obramowania, to:

- `none` — brak obramowania,
- `hidden` — obramowanie ukryte,
- `dotted` — linia kropkowana,
- `dashed` — linia kreskowana,
- `solid` — linia ciągła,
- `double` — linia ciągła podwójna,
- `groove` — „rowek”,
- `ridge` — „grzbiet”,
- `inset` — „ramka”,
- `outset` — „przycisk”.



UWAGA

Przy projektowaniu ramek obowiązkowe jest podanie wartości dla stylu i dla szerokości.

Kolor obramowania

Do definiowania koloru obramowania służy atrybut `border-color`. Jako wartości koloru można używać kodów kolorów zapisanych heksadecymalnie (`#ff00ff`), kodów `rgb(r,g,b)` lub ich nazw.

```
selektor {border-color: kolor;}
```

Podobnie jak w przypadku innych atrybutów obramowania, możliwe jest podanie jednej, dwóch, trzech lub czterech wartości koloru.

Możliwe jest też definiowanie koloru pojedynczych krawędzi. Służą do tego atrybuty:

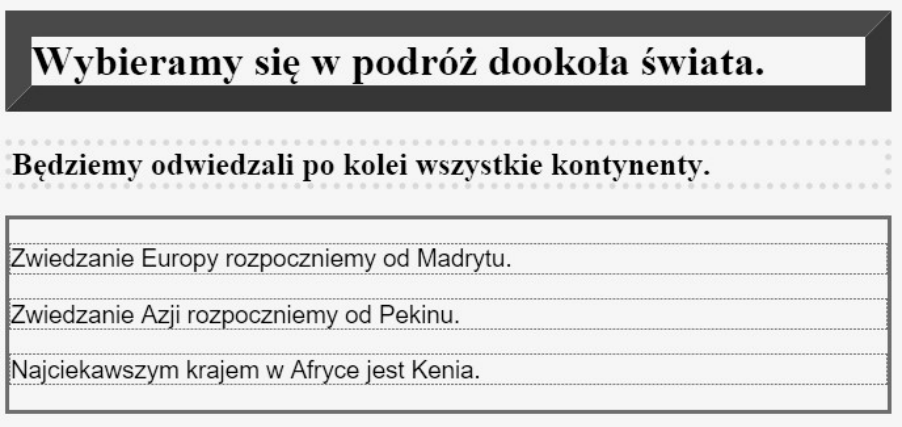
- `border-top-color`,
- `border-bottom-color`,
- `border-left-color`,
- `border-right-color`.

Przykład 3.65

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Obramowanie</title>
  <meta charset="UTF-8">
  <style>
  body {
    background-color: #f2f2f2;
  }
  p {
    font-size: 14pt;
    font-family: Arial;
    border-width: 1px;
    border-style: dashed;
    border-color: red;
  }
  h1 {
    border-width: 20px;
    border-style: outset;
    border-color: blue;
  }
  h2 {
    border-width: thick;
    border-style: dotted;
    border-color: gold;
  }
```

```
  div {
    border-width: medium;
    border-style: solid;
    border-color: green;
  }
  </style>
</head>
<body>
  <h1>Wybieramy się w podróż dookoła świata.</h1>
  <h2>Będziemy odwiedzali po kolei wszystkie kontynenty.</h2>
  <div>
    <p>Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
    <p>Zwiedzanie Azji rozpoczniemy od Pekinu.</p>
    <p>Najciekawszym krajem w Afryce jest Kenia.</p>
  </div>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.31.



Rysunek 3.31. Definiowanie koloru obramowania (border)

Atrybuty obramowania

Podobnie jak w przypadku tła, używając atrybutu `border`, można zdefiniować równocześnie kilka właściwości obramowania.

```
selektor {border: wartości_atrybutów;}
```

Wartości atrybutów dotyczą szerokości (`width`), stylu (`style`) i koloru (`color`). Mogą być wpisane w dowolnej kolejności i muszą być oddzielone znakiem spacji.

Przykład 3.66

```
p {border: double 1px red;}
h2 {border: 20px outset blue;}
h3 {border: thick dotted gold;}
```

3.5.2. Marginesy zewnętrzne (margin)

Margines zewnętrzny określa przestrzeń wokół definiowanego elementu. Przestrzeń ta oddziela element od innych elementów strony.

Dla każdego elementu można zdefiniować jego odległość od góry, dołu, lewej i prawej strony. Służy do tego atrybut `margin`.

```
selektor {margin-pozycja: rozmiar;}
```

Jako *pozycja* może wystąpić jedno z określeń dotyczących położenia marginesu:

- `top` — margines górny,
- `bottom` — margines dolny,
- `left` — margines lewy,
- `right` — margines prawy.

Jako *rozmiar* podawana jest wartość w jednostkach długości (np. `px`, `cm`). Można wpisać jako rozmiar `auto`. Spowoduje to nadanie przez przeglądarkę automatycznej wartości dla marginesu.

Przykład 3.67

```
p {margin-top: 1cm;}
h2 {margin-bottom: 2cm;}
h3 {margin-left: 20%;}
```

Za pomocą atrybutu `margin` można też ustawiać marginesy dla wszystkich czterech boków elementu.

```
selektor {margin: wartości;}
```

Jako wartości atrybutu można podać:

- *jedną wartość* — wszystkie marginesy jednakowe;
- *dwie wartości* — pierwsza oznacza marginesy górny i dolny, druga — marginesy lewy i prawy;
- *trzy wartości* — pierwsza oznacza margines górny, druga — marginesy lewy i prawy, trzecia — margines dolny;
- *cztery wartości* — każda wartość określa osobno marginesy górny, prawy, dolny, lewy.

Jeśli przy takim zapisie chcemy pominąć jeden z marginesów, wystarczy ustawić jego wartość na `0`.

Przykład 3.68

```
p {margin: 1cm;}
h2 {margin: 20px 10px;}
h3 {margin: 2cm 15px 1.5cm 12px;}
```

3.5.3. Marginesy wewnętrzne (padding)

Margines wewnętrzny to przestrzeń między zawartością elementu a obramowaniem.

Podobnie jak dla marginesów zewnętrznych, można zdefiniować tę odległość od góry, dołu, lewej i prawej strony. Do definiowania marginesów wewnętrznych służy atrybut `padding`.

```
selektor {padding-pozycja: rozmiar;}
```

Jako *pozycja* mogą wystąpić określenia dotyczące położenia marginesów:

- `top` — margines górny,
- `bottom` — margines dolny,
- `left` — margines lewy,
- `right` — margines prawy.

Przykład 3.69

```
p {padding-top: 25px;}
h3 {padding-left: 2cm;}
```

Można również ustawiać marginesy wewnętrzne dla wszystkich czterech boków elementu na takich samych zasadach jak dla marginesu zewnętrznego.

```
selektor {padding: wartości;}
```

Wartości atrybutu:

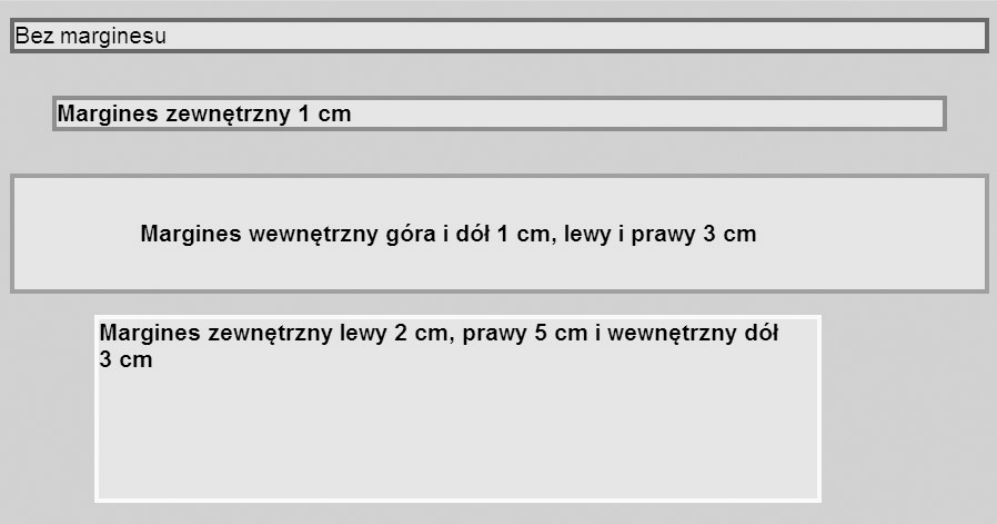
- *jedna wartość* — wszystkie marginesy jednakowe;
- *dwie wartości* — pierwsza oznacza marginesy górny i dolny, druga — marginesy lewy i prawy;
- *trzy wartości* — pierwsza oznacza margines górny, druga — marginesy lewy i prawy, trzecia — margines dolny;
- *cztery wartości* — każda wartość określa osobno marginesy górny, prawy, dolny, lewy.

Aby pominąć jeden z marginesów wewnętrznych, należy wpisać wartość `0`.

Różnice związane z definiowaniem marginesów zewnętrznych i wewnętrznych pokazuje rysunek 3.32.

Ćwiczenie 3.10

Wykorzystując poznane narzędzia, zaprojektuj stronę, która będzie wyglądała podobnie jak ta na rysunku 3.32.



Rysunek 3.32. Definiowanie marginesów wewnętrznych i zewnętrznych

3.5.4. Obrys

Obramowanie wokół elementu może być tworzone również poprzez zdefiniowanie obrysu. Obrys różni się od obramowania tym, że:

- nie zajmuje miejsca, dzięki czemu nie wpływa na rozmiar i położenie elementu;
- jest tworzony na wierzchu elementu;
- jest definiowany dla wszystkich krawędzi równocześnie.

Obrys można definiować przy użyciu atrybutu `outline`.

```
selektor {outline: wartości;}
```

W ten sposób wybiera się kolor (ang. *color*), styl (ang. *style*) i grubość (ang. *width*) obrysu. Obowiązkowym atrybutem jest styl obrysu. Pozostałe dwa atrybuty występują opcjonalnie. Sposób definiowania obrysu jest taki sam jak przy definiowaniu obramowania.

Przykład 3.70

```
p {outline: solid 3px red;}
h2 {outline-style: dashed;}
h3 {outline-width: 10px; outline-color: blue;}
```

3.5.5. Rozmiary elementów

W modelu blokowym można ustalać dokładne rozmiary różnych elementów, np. akapitu, tabeli, obrazka, bloku `div`.

Szerokość

Do definiowania szerokości elementu służy atrybut `width`.

```
selektor {width: szerokość;}
```

Przykład 3.71

```
h3 {width: 20%;}
```

Szerokość minimalna

Za pomocą atrybutu `min-width` można określić minimalną szerokość elementu. Pozwala to ustawić rozmiar elementu tak, aby nigdy nie był mniejszy od podanej wartości.

```
selektor {min-width: szerokość;}
```

Przykład 3.72

```
img {min-width: 200px;}
```

Szerokość maksymalna

Atrybut `max-width` pozwala określić maksymalną szerokość elementu. Element może być węższy od podanej wartości, ale nie może być od niej szerszy.

```
selektor {max-width: szerokość;}
```

Przykład 3.73

```
img {max-width: 300px;}
```

Wysokość

Atrybut `height` ustala wysokość elementu.

```
selektor {height: wysokość;}
```

Przykład 3.74

```
h3 {height: 3cm;}
```

Wysokość minimalna

Za pomocą atrybutu `min-height` można określić minimalną wysokość elementu. Oznacza to, że wysokość elementu nigdy nie będzie mniejsza od podanej wartości.

```
selektor {min-height: wysokość;}
```

Przykład 3.75

```
img {min-height: 300px;}
```

Wysokość maksymalna

Atrybut `max-height` pozwala określić maksymalną wysokość elementu. Element może być niższy od podanej wartości, ale nie może być od niej wyższy.

```
selektor {max-height: wysokość;}
```

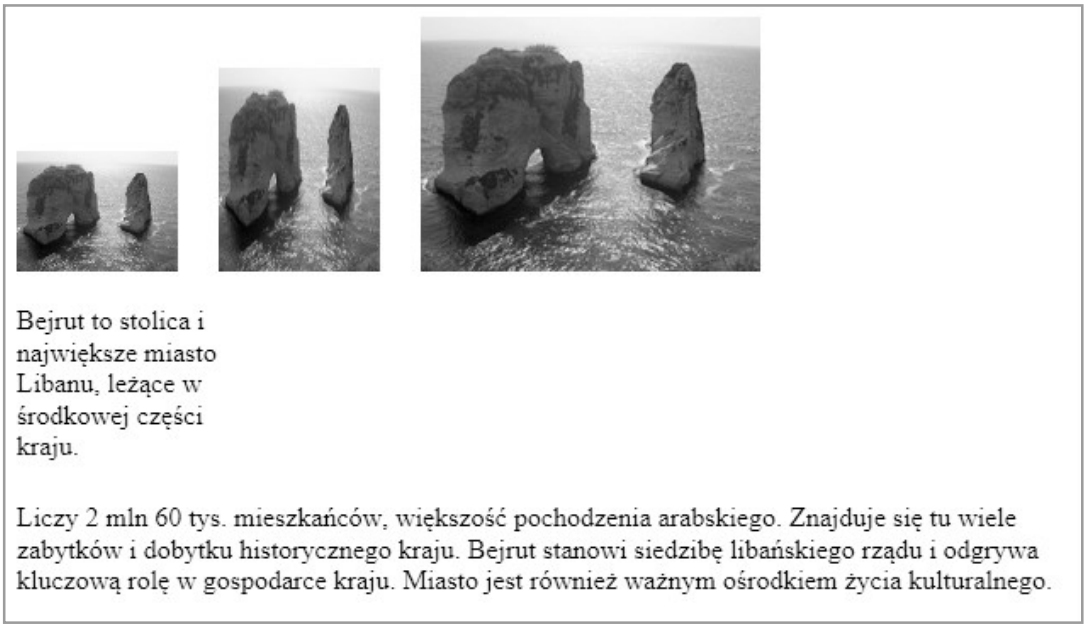
Przykład 3.76

```
img {max-height: 5cm;}
```

Przykład 3.77

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Bejrut</title>
  <meta charset="UTF-8">
  <style>
  img.roz1 {height: auto; margin-right: 20px;}
  img.roz2 {height: 120px; margin-right: 20px;}
  p.blok {height: 100px; width: 130px;}
  </style>
</head>
<body>
  <img class="roz1" src="bejr.jpg" width="95" height="84" alt="Obrazek">
  <img class="roz2" src="bejr.jpg" width="95" height="84" alt="Obrazek">
  <img src="bejr.jpg" alt="Obrazek">
  <p class="blok">
  Bejrut to stolica i największe miasto Libanu, leżące w środkowej części
kraju.
  </p>
  <p>
  Liczy 2 mln 60 tys. mieszkańców, większość pochodzenia arabskiego.
  Znajduje się tu wiele zabytków i dobytku historycznego kraju.
  Bejrut stanowi siedzibę libańskiego rządu i odgrywa kluczową rolę w gospo-
darce kraju.
  Miasto jest również ważnym ośrodkiem życia kulturalnego.
  </p>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.33.



Rysunek 3.33. Określenie rozmiarów elementów strony

Przepełnienie

Jeżeli zawartość elementu nie mieści się w rozmiarach podanych za pomocą atrybutów `width` i `height`, możliwe jest ukrycie niemieszczącej się zawartości, powiększenie rozmiarów elementu i pokazanie całej zawartości lub wyświetlenie suwaków, które pozwolą (poprzez przewijanie) na obejrzenie całej zawartości elementu. Do określania sposobu pokazywania zawartości elementu w takiej sytuacji służy atrybut `overflow`.

```
selektor {overflow: sposób;}
```

Wartościami parametru *sposób* mogą być:

- `visible` — pokazywana jest cała zawartość elementu, niezależnie od zdefiniowanego rozmiaru;
- `hidden` — treść niemieszcząca się w zdefiniowanym rozmiarze elementu zostanie ukryta;
- `scroll` — powoduje przepełnienie i dodanie paska przewijania, który pozwoli na przewijanie zawartości wewnątrz bloku i obejrzenie całej zawartości elementu; pasek przewijania zostanie dodany zarówno w poziomie, jak i w pionie;
- `auto` — jeżeli jest to konieczne, suwaki zostaną wyświetlone.

Atrybutu `overflow` można używać tylko dla elementów blokowych ze zdefiniowaną wysokością. Domyślną wartością jest `visible`, co oznacza, że element nie jest obcinany, nawet jeżeli nie mieści się w zdefiniowanych rozmiarach.

Przykład 3.78

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Przepełnienie</title>
  <meta charset="UTF-8">
  <style>
  div {width: 250px; height: 200px; overflow: scroll;}
  </style>
</head>
<body>
  <div>
    <img src="zdj.jpg" alt="zdjecie">
  </div>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.34.

Rysunek 3.34. Suwaki pozwalają obejrzeć całe zdjęcie

Wartość `auto` działa podobnie jak wartość `scroll`, ale paski przewijania są dodawane tylko, gdy to jest konieczne.

Do określenia sposobu wyświetlania treści tylko w poziomie lub pionie służą atrybuty `overflow-x` i `overflow-y`. Atrybut `overflow-x` określa, co należy zrobić z prawą/lewą krawędzią treści, a atrybut `overflow-y` — co zrobić z górną/dolną krawędzią.

3.6. Inne elementy

3.6.1. Listy

W języku HTML można definiować listy punktowane (`<ul>`) i numerowane (`<ol>`). Jeżeli nie odpowiada nam sposób, w jaki te listy są wyświetlane przez przeglądarki, w języku CSS możemy zdefiniować inny sposób wyświetlania list.

Możliwe są: zmiana rodzaju punktora, zdefiniowanie własnego punktora oraz określenie położenia punktora i jego odległości od tekstu.

Styl listy

Większość przeglądarek w listach punktowanych wyświetla okrągłe punktory, a w listach numerowanych cyfry.

Za pomocą atrybutu `list-style-type` można zmienić punktor lub cyfrę wykazu na liście.

```
selektor {list-style-type: typ;}
```

Selektorami mogą być znaczniki definiujące listy: `<ul>`, `<ol>` oraz `<li>`.

Parametr *typ* określa wygląd punktora (*markera*).

Wybrane typy punktorów:

- `disc` — koło,
- `circle` — okrąg,
- `square` — kwadrat,
- `decimal` — liczby arabskie,
- `lower-roman` — małe cyfry rzymskie,
- `upper-roman` — duże cyfry rzymskie,
- `lower-alpha` — małe litery,
- `upper-alpha` — duże litery,
- `none` — brak punktora.

Na rysunku 3.35 zostały przedstawione różne typy punktorów.

Rysunek 3.35. Definiowanie punktorów listy

Ćwiczenie 3.11

Zaprojektuj na stronie internetowej listę punktowaną, listę numerowaną oraz listę z podpunktami. Zastosuj różne typy punktorów. Sprawdź ich wygląd. Stronę przetestuj w kilku przeglądarkach internetowych.

Obraz jako punktor

Punktorem listy może być obrazek zapisany w pliku. Do ustawienia obrazka jako markera służy atrybut `list-style-image`.

```
selektor {list-style-image: url(ścieżka_dostępu);}
```

Ścieżka dostępu określa położenie pliku z obrazkiem, który ma być punktorem.

Przykład 3.79

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Punktor graficzny</title>
  <meta charset="UTF-8">
  <style>
  ul {list-style-image: url(punktor.png);}
  </style>
</head>
<body>
  <ul>
    <li>Ameryka Północna</li>
    <li>Ameryka Południowa</li>
    <li>Australia</li>
  </ul>
</body>
</html>
```

Wynik wykonania kodu zilustrowano na rysunku 3.36.

Rysunek 3.36. Obraz jako punktor

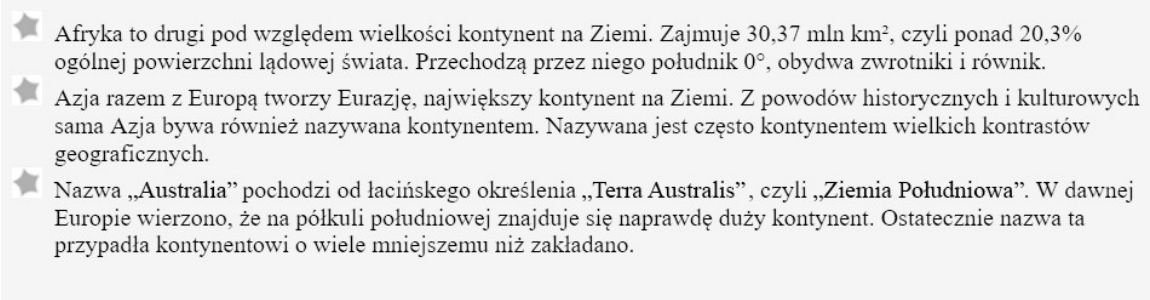
Pozycja punktora

Oprócz typu punktora można ustalić położenie punktora względem tekstu. Służy do tego atrybut `list-style-position`.

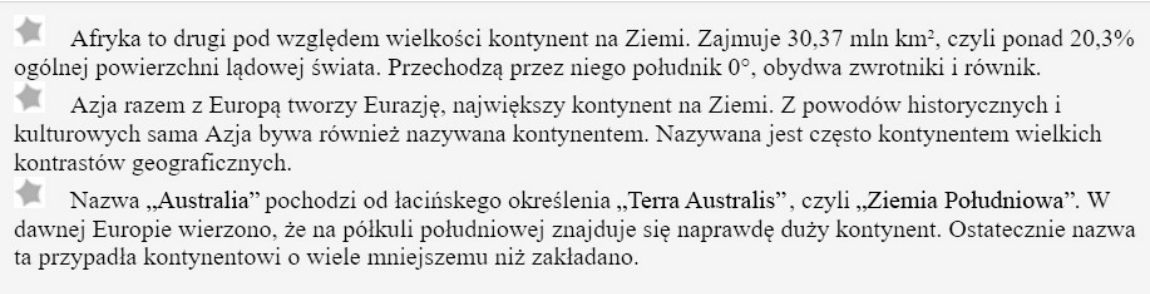
```
selektor {list-style-position: pozycja;}
```

Parametr *pozycja* określa zachowanie się tekstu, który nie zmieści się w jednej linii, względem punktora. Możliwe są dwie wersje:

- `outside` — punktor pojawia się na zewnątrz zawartości (rysunek 3.37);
- `inside` — punktor pojawia się wewnątrz zawartości (rysunek 3.38).



Rysunek 3.37. Punktor na zewnątrz zawartości



Rysunek 3.38. Punktor wewnątrz zawartości

Atrybuty listy

Używając atrybutu `list-style`, można równocześnie zdefiniować właściwości listy: `list-style-type`, `list-style-image` oraz `list-style-position`. Wartości atrybutów muszą być oddzielone spacjami.

```
selektor {list-style: wartości_atrybutów;}
```

Przykład 3.80

```
<ul style="list-style: square inside;">
  <li>Europa</li>
  <li>Azja</li>
  <li>Afryka</li>
</ul>
```

3.6.2. Tabele

W języku HTML tabele są stosowane do przedstawiania danych. Jeżeli chcemy za pomocą tabel zaprezentować dane, to użycie stylów CSS pozwoli znacznie poprawić ich wygląd.

Podpis

W celu określenia położenia podpisu dodanego do tabeli używamy atrybutu `caption-side`.

```
selektor {caption-side: ustawienie;}
```

Parametr *ustawienie* może przyjmować wartości:

- `top` — podpis zostanie wyświetlony nad tabelą;
- `bottom` — podpis zostanie wyświetlony pod tabelą;
- `left` — podpis zostanie wyświetlony z lewej strony tabeli;
- `right` — podpis zostanie wyświetlony z prawej strony tabeli.

Nie wszystkie przeglądarki prawidłowo interpretują ten atrybut.

Obramowanie

Do definiowania krawędzi tabeli używa się atrybutu `border`.

```
selektor {border: wartości;}
```

Dla tego atrybutu można ustawić grubość, rodzaj oraz kolor obramowania.

Przykład 3.81

```
table, tr, td {border: 2px solid red;}
```

Atrybut `border-collapse` określa sposób obramowania komórek tabeli. Możliwe jest obramowanie sąsiednich komórek za pomocą jednej linii lub obramowanie każdej komórki osobno.

```
selektor {border-collapse: model;}
```

Wartości parametru *model* to:

- `collapse` — pojedyncze obramowanie;
- `separate` — komórki tabeli będą od siebie odseparowane.

Przykład 3.82

```
table {border-collapse: separate;}
```

Można także określić sposób obramowania dla pustych komórek. Służy do tego atrybut `empty-cells`.

```
selektor {empty-cells: wartość;}
```

Atrybut może przyjmować wartości:

- `show` — obramowanie wokół pustych komórek powinno być wyświetlane;
- `hide` — obramowanie wokół pustych komórek powinno zostać ukryte.

Do tabel oraz do ich komórek można stosować wszystkie atrybuty definiowane dla obramowań w modelu blokowym.

Rozmieszczenie

Do rozplanowania tabeli na stronie można użyć atrybutu `table-layout`.

```
selektor {table-layout: rozmieszczenie;}
```

Atrybut może przyjmować wartości:

- `auto` — szerokość tabeli jest ustawiana automatycznie i zależy od zawartości komórek;
- `fixed` — szerokość tabeli wynika z zadeklarowanej szerokości poszczególnych komórek oraz szerokości całej tabeli.

Różne przeglądarki różnie interpretują to polecenie, jeśli nie zostanie podana ogólna szerokość tabeli lub jeśli szerokość tabeli jest większa niż suma szerokości jej komórek.

Przykład 3.83

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Obramowanie tabel</title>
  <meta charset="UTF-8">
  <style>
  table, tr, td {
    border: 1px solid blue;
    border-collapse: collapse;
    table-layout: fixed;
  }
  th {
    border: 1px solid green;
    border-collapse: collapse;
  }
  </style>
```

```
</head>
<body>
  <table>
    <tr><th>Europa</th><th>Azja</th><th>Afryka</th></tr>
    <tr><td>Francja</td><td>Indie</td><td>Egipt</td></tr>
    <tr><td>Hiszpania</td><td>Chiny</td><td>Nigeria</td></tr>
    <tr><td>Włochy</td><td>Japonia</td><td>Kongo</td></tr>
  </table>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.39.

Europa	Azja	Afryka
Francja	Indie	Egipt
Hiszpania	Chiny	Nigeria
Włochy	Japonia	Kongo

Rysunek 3.39. Wyświetlenie tabeli ze zdefiniowanymi obramowaniami

Odstępy między komórkami

Do definiowania odstępów między komórkami służy atrybut `border-spacing`.

```
selektor {border-spacing: odstęp;}
```

Odstęp to zapisana w jednostkach długości odległość między komórkami tabeli. Można podać jedną wartość (i wtedy dotyczy ona wszystkich odstępów) lub dwie wartości rozdzielone spacją (i wtedy definiują one poziomy oraz pionowy odstęp między komórkami).

Szerokość i wysokość tabeli

Za pomocą właściwości `width` i `height` można zdefiniować szerokość i wysokość tabeli.

Przykład 3.84

```
table {width: 100%;}
th {height: 50px;}
```

Przykład 3.85

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
```

```
  <title>Obramowanie tabel</title>
  <meta charset="UTF-8">
  <style>
  table {
    width: 100%
  }
  table, tr, td {
    border: 1px solid blue;
    border-collapse: collapse;
    table-layout: fixed;
  }
  th {
    border: 1px solid green;
    border-collapse: collapse;
    height: 50px;
  }
  </style>
</head>
<body>
  <table>
    <tr><th>Europa</th><th>Azja</th><th>Afryka</th></tr>
    <tr><td>Francja</td><td>Indie</td><td>Egipt</td></tr>
    <tr><td>Hiszpania</td><td>Chiny</td><td>Nigeria</td></tr>
    <tr><td>Włochy</td><td>Japonia</td><td>Kongo</td></tr>
  </table>
</body>
</html>
```

Wynik wykonania kodu przedstawiono na rysunku 3.40.

Europa	Azja	Afryka
Francja	Indie	Egipt
Hiszpania	Chiny	Nigeria
Włochy	Japonia	Kongo

Rysunek 3.40. Tabela ze zdefiniowanymi: wysokością i szerokością

3.6.3. Pozycjonowanie

Pozycjonowanie pozwala zdefiniować położenie elementów na stronie internetowej. Elementy można rozmieszczać nie tylko względem brzegów strony, ale również względem jej poszczególnych elementów. Można też umieszczać elementy tak, że jeden będzie przykrywał inny.

Do pozycjonowania elementów służy atrybut `position`.

```
selektor {position: rodzaj; parametry;}
```

Parametr *rodzaj* określa sposób pozycjonowania elementu i może przyjmować wartości:

- `static` — przywraca normalne pozycjonowanie elementu. Jest to wartość przydatna wtedy, gdy wcześniej (np. w arkuszach stylów) została podana deklaracja pozycjonowania tego typu elementów. Użycie w definicji stylu `inline` dla wybranego elementu deklaracji `position: static` zniesie tę deklarację dla danego elementu.
- `relative` (inaczej: pozycjonowanie względne) — pozwala przesunąć wybrany element w inne miejsce w stosunku do położenia pierwotnego. Poprzez ustawienie parametrów można określać sposób przesunięcia.
- `absolute` — pozwala przesunąć wybrany element w inne miejsce względem brzegów strony lub bloku (jeżeli element został umieszczony w bloku). Poprzez ustawienie parametrów można określić sposób przesunięcia. Element zostanie umieszczony „na sztywno" w miejscu określonym przez parametry.
- `fixed` — działa podobnie jak pozycjonowanie absolutne. Różnica polega na tym, że pozycjonowanie `fixed` ustala położenie elementu zawsze względem krawędzi okna przeglądarki. Tak pozycjonowany element jest nieruchomy przy przewijaniu strony (jest przez cały czas widoczny w tym samym miejscu).

Jako *parametr* należy podawać:

- `left:` *wartość;* — przesunięcie o określoną wartość w stosunku do lewej krawędzi położenia pierwotnego,
- `top:` *wartość;* — przesunięcie o określoną wartość w stosunku do górnej krawędzi położenia pierwotnego,
- `right:` *wartość;* — przesunięcie o określoną wartość w stosunku do prawej krawędzi położenia pierwotnego,
- `bottom:` *wartość;* — przesunięcie o określoną wartość w stosunku do dolnej krawędzi położenia pierwotnego.

Można łączyć ze sobą różne parametry. Trzeba jednak pamiętać o tym, że parametr `left` ma pierwszeństwo przed `right`, a parametr `top` ma pierwszeństwo przed `bottom`.

Wartość parametru *parametr* oznacza określoną odległość od podanej krawędzi; odległość ta powinna być podawana w jednostkach długości. Wpisanie wartości ujemnej spowoduje przesunięcie w przeciwną stronę. Można użyć słowa `auto`, aby ustawić wartość domyślną.

Przykład 3.86

```
h1 {
  position: absolute;
  top: 100px;
  left: 200px;
}
img {
  position: relative;
  left: 350px;
  bottom: 150px;
}
div {
  position: fixed;
  top: 20px;
  right: 10px;
  border: dotted 1px;
  padding: 5px;
  background-color: yellow;
}
```

Przezroczystość

Przy użyciu atrybutu `opacity` można zdefiniować przezroczystość dla elementów wstawianych na stronę.

```
selektor {opacity: nieprzezroczystość;}
```

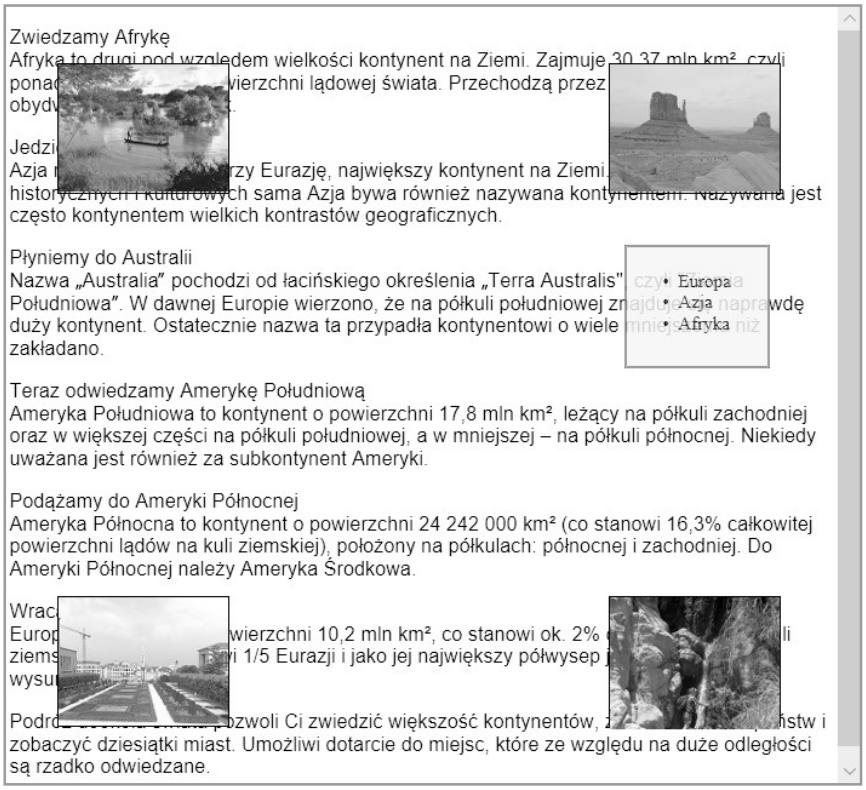
Parametr *nieprzezroczystość* to liczba z przedziału od `0,0` do `1,0`. `0` oznacza całkowitą przezroczystość, `1` — brak przezroczystości.

Zdefiniowany poziom przezroczystości jest nadawany wszystkim elementom znajdującym się wewnątrz określonego elementu. Elementom wewnętrznym nie można zmniejszyć przezroczystości (np. nie mogą one stać się zupełnie nieprzezroczyste), ale można im nadać większą przezroczystość w stosunku do elementu zewnętrznego.

Ćwiczenie 3.12

Wykorzystując poznane narzędzia do pozycjonowania elementów, na utworzonej stronie, w rogach rozmieść cztery zdjęcia zgodnie z rysunkiem 3.41. Zaprojektuj stronę tak, aby lista widoczna z prawej była elementem nieruchomym w czasie przewijania zawartości strony. Wstawione elementy odpowiednio sformatuj.

Rysunek 3.41. Strona zawiera pozycjonowanie obrazów



Rozwiązanie

Rozwiązaniem ćwiczenia 3.12 może być następujący kod:

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Pozycjonowanie</title>
  <meta charset="UTF-8">
  <style>
  p {
    font-size: 13pt;
    font-family: Arial;
  }
  div {
    width: 150px;
    height: 113px;
    border: solid 1px black;
  }
  #lista {
```

```
    position: fixed;
    top: 210px;
    right: 60px;
    border: double 2px;
    padding: 5px;
    background-color: #f0f0f0;
    width:3cm;
    height:2.5cm;
    opacity: 0.8
  }
  #blok1 {
    position: absolute;
    top: 50px;
    left: 50px;
  }
  #blok2 {
    position: absolute;
    top: 50px;
    right: 50px;
  }
  #blok3 {
    position: absolute;
    bottom: 50px;
    right: 50px;
  }
  #blok4 {
    position: absolute;
    bottom: 50px;
    left: 50px;
  }
  </style>
</head>
<body>
  <div id="lista">
    <ul>
```

```
      <li>Europa</li>
      <li>Azja</li>
      <li>Afryka</li>
    </ul>
  </div>
  <div id="blok1">
    <img src="obr1.jpg" alt="obraz1">
  </div>
  <div id="blok2">
    <img src="obr2.jpg" alt="obraz2">
  </div>
  <div id="blok3">
    <img src="obr3.jpg" alt="obraz3">
  </div>
  <div id="blok4">
    <img src="obr4.jpg" alt="obraz4">
  </div>
  <p>
  Zwiedzamy Afrykę<br>
  Afryka to drugi pod względem wielkości kontynent na Ziemi. Zajmuje 30,37 mln
km², czyli ponad 20,3% ogólnej powierzchni lądowej świata. Przechodzą przez
niego południk 0°, obydwa zwrotniki i równik.
  </p>
  <p>
  Jedziemy do Azji<br>
  Azja razem z Europą tworzy Eurazję, największy kontynent na Ziemi. Z powodów
historycznych i kulturowych sama Azja bywa również nazywana kontynentem. Nazy-
wana jest często kontynentem wielkich kontrastów geograficznych.
  </p>
  <p>
  Płyniemy do Australii<br>
  Nazwa Australia pochodzi od łacińskego określenia "Terra Australis", czyli
"Ziemia Południowa". W dawnej Europie wierzono, że na półkuli południowej znaj-
duje się naprawdę duży kontynent. Ostatecznie nazwa ta przypadła kontynentowi
o wiele mniejszemu niż zakładano.
  </p>
```

```
  <p>
  Teraz odwiedzamy Amerykę Południową<br>
  Ameryka Południowa to kontynent o powierzchni 17,8 mln km² leżący na półkuli
zachodniej oraz w większej części na półkuli południowej, a w mniejszej – na
półkuli północnej. Niekiedy uważana jest również za subkontynent Ameryki.
  </p>
  <p>Podążamy do Ameryki Północnej<br>
  Ameryka Północna to kontynent o powierzchni 24 242 000 km² (co stanowi 16,3%
całkowitej powierzchni lądów na kuli ziemskiej), położony na półkulach: północ-
nej i zachodniej. Do Ameryki Północnej należy Ameryka Środkowa.
  </p>
  <p>Wracamy do Europy<br>
  Europa to kontynent o powierzchni 10,2 mln km², co stanowi ok. 2% całej po-
wierzchni kuli ziemskiej. Europa stanowi 1/5 Eurazji i jako jej największy pół-
wysep jest najbardziej wysunięta na zachód.
  </p>
  <p>
  Podróż dookoła świata pozwoli Ci zwiedzić większość kontynentów, zawitać do
wielu państw i zobaczyć dziesiątki miast. Umożliwi dotarcie do miejsc, które ze
względu na duże odległości są rzadko odwiedzane.
  </p>
</body>
</html>
```

Ćwiczenie 3.13

Wykorzystując poznane narzędzia do pozycjonowania elementów, ułóż zdjęcia na stronie podobnie jak na rysunku 3.42.

Rysunek 3.42. Przykładowe położenie elementów na stronie uzyskane w wyniku pozycjonowania

Rozwiązanie

Rozwiązaniem ćwiczenia 3.13 może być następujący kod:

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Pozycjonowanie relative</title>
  <meta charset="UTF-8">
  <style>
  #blok1 {position: relative; left: 40px;}
  #blok2 {position: relative; left: 80px; top: -80px;}
  #blok3 {position: relative; left: 120px; top: -160px;}
  #blok4 {position: relative; left: 160px; top: -240px;}
  </style>
</head>
<body>
  <div id="blok1">
    <img src="bejr.jpg" alt="bejrut">
  </div>
  <div id="blok2">
    <img src="bejr.jpg" alt="bejrut">
  </div>
  <div id="blok3">
    <img src="bejr.jpg" alt="bejrut">
  </div>
  <div id="blok4">
    <img src="bejr.jpg" alt="bejrut">
  </div>
</body>
</html>
```

Nakładanie elementów (warstwy)

Gdy kilka elementów nachodzi na siebie, to można określić, w jaki sposób te elementy będą nakładane. Służy do tego atrybut `position`. Nakładającym się elementom można nadać numer (`z-index`) i zastosować regułę, że elementy z wyższym numerem będą nakładane na elementy z niższym numerem.

```
selektor {position: rodzaj; parametry; z-index: numer;}
```

Wyrażenie `position: rodzaj; parametry;` określa parametry pozycjonowania.

Parametr *numer* mówi o kolejności nakładania pozycjonowanych elementów. Element, któremu nadamy numer `1`, znajdzie się na samym spodzie i będzie zakryty przez inne elementy.

Przykład 3.87

```
#blok3 {position: relative; left: 120px; top: -160px; z-index: 4;}
```

Poprawnie interpretowane są również wartości ujemne.

Przykład 3.88

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Pozycjonowanie relative</title>
  <meta charset="UTF-8">
  <style>
  #blok1 {position: relative; left: 40px; z-index: 1;}
  #blok2 {position: relative; left: 80px; top: -80px; z-index: 2;}
  #blok3 {position: relative; left: 120px; top: -160px; z-index: 4;}
  #blok4 {position: relative; left: 160px; top: -240px; z-index: 3;}
  </style>
</head>
<body>
  <div id="blok1">
    <img src="bejr3.jpg" alt="bejrut">
  </div>
  <div id="blok2">
    <img src="bejr3.jpg" alt="bejrut">
  </div>
  <div id="blok3">
    <img src="bejr3.jpg" alt="bejrut">
  </div>
  <div id="blok4">
    <img src="bejr3.jpg" alt="bejrut">
  </div>
</body>
</html>
```

Wynik wykonania kodu został zilustrowany na rysunku 3.43.

Rysunek 3.43. Zmiana kolejności elementów

Obcinanie elementu

Do wycięcia z elementu jego fragmentu w kształcie prostokąta służy atrybut `position: absolute; clip: rect;` z odpowiednimi parametrami.

```
selektor {position: absolute; clip: rect(góra, prawo, dół, lewo);}
```

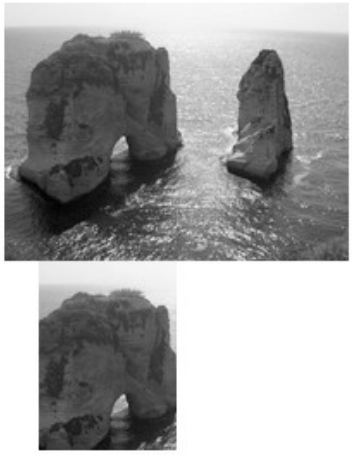
Wartości: *góra*, *prawo*, *dół*, *lewo* oznaczają współrzędne wyciętego prostokąta. Wartości te są określane względem lewego górnego rogu elementu. Powinny być podawane w jednostkach długości. Można używać określenia `auto`, co oznacza pominięcie cięcia z danej strony.

Przykład 3.89

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Obcinanie elementu</title>
  <meta charset="UTF-8">
  <style>
  #blok {
    position: absolute;
    clip: rect(0px, 100px, 110px, 20px);
  }
  </style>
</head>
<body>
  <img src="bejr2.jpg" alt="bejrut">
  <div id="blok"><img src="bejr2.jpg" alt="bejrut"></div>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.44.

Rysunek 3.44.
Obcinanie elementu



Wyrównanie w pionie

Atrybut `vertical-align` pozwala na zdefiniowanie wyrównania elementu w stosunku do innych elementów strony.

```
selektor {vertical-align: sposób;}
```

Jako wartość parametru *sposób* należy podać:

- `baseline` — wyrównuje linię bazową elementu do linii bazowej elementu nadrzędnego;
- `middle` — ustawia element na środku wysokości elementów sąsiednich;
- `text-top` — wyrównuje górną krawędź elementu do górnej krawędzi tekstu elementu nadrzędnego;
- `text-bottom` — wyrównuje podstawę elementu do podstawy tekstu elementu nadrzędnego;
- `super` — ustawia element jako indeks górny;
- `sub` — ustawia element jako indeks dolny;
- `top` — wyrównuje górną krawędź elementu do górnej krawędzi najwyższego elementu w linii;
- `bottom` — wyrównuje podstawę elementu do podstawy elementu położonego najniżej w linii.

Przykład 3.90

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Wyrównanie w pionie</title>
  <meta charset="UTF-8">
```

```
  <style>
  body {background-color: #d0d0d0;}
  span {vertical-align: 4mm;}
  </style>
</head>
<body>
  <h3>Atrybut <span>vertical-align</span> definiuje wyrównanie elementu.</h3>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.45.

Rysunek 3.45. Wyrównanie elementu w stosunku do innych elementów

Ustawienie w poziomie

Do ustawiania elementu względem elementów, które z nim sąsiadują w poziomie, służy atrybut `float`. Potocznie mówi się, że element „pływa" po lewej lub prawej stronie okna przeglądarki lub bloku.

```
selektor {float: sposób;}
```

Parametr *sposób* może przyjmować wartości:

- `left` — element zostanie ustawiony po lewej stronie względem sąsiednich elementów;
- `right` — element zostanie ustawiony po prawej stronie względem sąsiednich elementów;
- `none` — element nie będzie sąsiadował z innymi elementami.

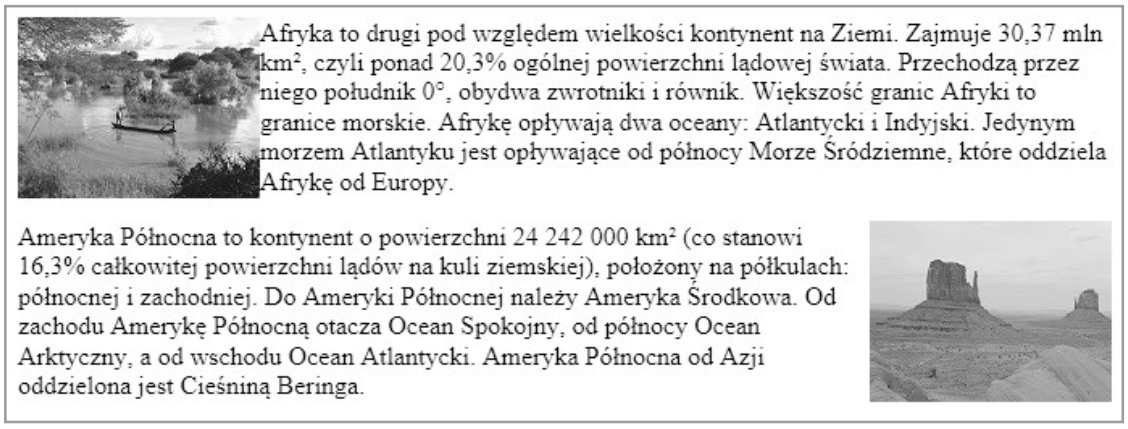
Najczęściej atrybut jest wykorzystywany do określenia sposobu oblewania tekstem elementu pływającego (`float`).

Przykład 3.91

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Pływanie elementów</title>
  <meta charset="UTF-8">
```

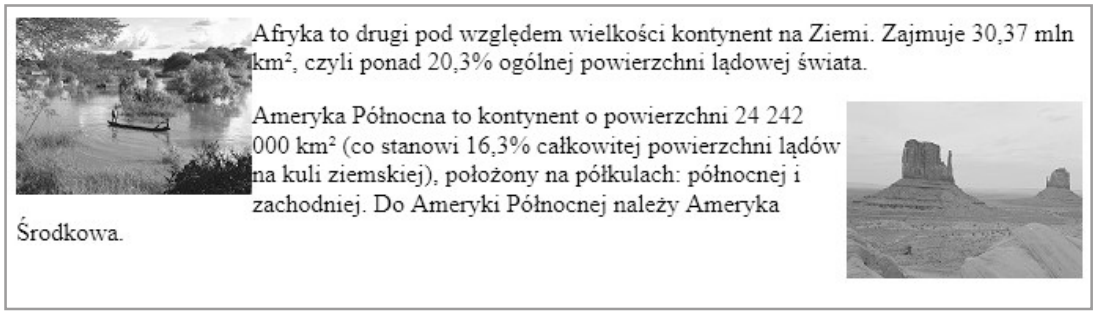
```
  <style>
  #blok1 {float:left;}
  #blok2 {float:right;}
  </style>
</head>
<body>
  <img id="blok1" src="czad.jpg" alt="czad">
  <p>Afryka to drugi pod względem wielkości kontynent na Ziemi. Zajmuje 30,37 mln km², czyli ponad 20,3% ogólnej powierzchni lądowej świata. Przechodzą przez niego południk 0°, obydwa zwrotniki i równik. Większość granic Afryki to granice morskie. Afrykę opływają dwa oceany: Atlantycki i Indyjski. Jedynym morzem Atlantyku jest opływające od północy Morze Śródziemne, które oddziela Afrykę od Europy.</p>
  <img id="blok2" src="utah.jpg" alt="utah">
  <p>Ameryka Północna to kontynent o powierzchni 24 242 000 km² (co stanowi 16,3% całkowitej powierzchni lądów na kuli ziemskiej), położony na półkulach: północnej i zachodniej. Do Ameryki Północnej należy Ameryka Środkowa. Od zachodu Amerykę Północną otacza Ocean Spokojny, od północy Ocean Arktyczny, a od wschodu Ocean Atlantycki. Ameryka Północna od Azji oddzielona jest Cieśniną Beringa.</p>
</body>
</html>
```

Wynik wykonania kodu zilustrowano na rysunku 3.46.



Rysunek 3.46. Zastosowanie atrybutu float do ustawienia tekstu wokół zdjęcia

Jeżeli tekst umieszczony z prawej lub lewej strony zdjęcia będzie zbyt krótki, to zdjęcia nie zostaną ułożone jedno pod drugim, tak jak na rysunku wyżej, lecz znajdą się obok siebie (rysunek 3.47). Dodatkowo ułożenie zdjęć będzie zależało od rozdzielczości ekranu i rozmiaru okna przeglądarki.



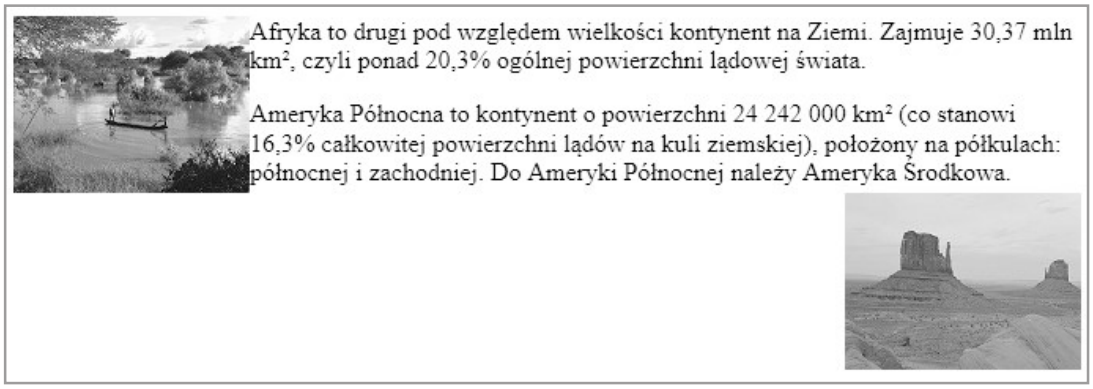
Rysunek 3.47. Błędy przy rozmieszczaniu tekstu wokół zdjęcia

Aby temu zapobiec, należy użyć atrybutu `clear`, który ustala pozycję kolejnych elementów względem elementu pływającego (rysunek 3.48).

```
selektor {clear: sposób;}
```

Wartość atrybutu `clear` określa sposób zachowania się kolejnych elementów względem elementu pływającego:

- `left` — lewy bok następnego elementu nie przylega do elementu pływającego;
- `right` — prawy bok następnego elementu nie przylega do elementu pływającego;
- `both` — żaden bok następnego elementu nie przylega do elementu pływającego;
- `none` — brak ograniczeń.



Rysunek 3.48. Efekt zastosowania atrybutu clear

Ćwiczenie 3.14

Popraw podany w przykładzie 3.91 kod tak, aby drugie zdjęcie było wyświetlane poniżej pierwszego zdjęcia niezależnie od długości tekstu występującego obok zdjęcia oraz bez względu na wielkość okna przeglądarki i rozdzielczość ekranu (rysunek 3.48).

Kolumny na stronie

Atrybut `float` może być wykorzystany do tworzenia kolumn w dokumencie. W tym celu musimy zawartość każdej kolumny zawrzeć w znaczniku `<div>`. Następnie, np. przy trzech kolumnach, należy określić szerokość każdego znacznika `<div>`. Po zdefiniowaniu dla każdego znacznika `<div>` atrybutu `float: left` uzyskamy „pływanie” kolejnych kolumn z lewej strony.

Przykład 3.92

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Kolumny na stronie</title>
  <meta charset="UTF-8">
  <style>
  div {width:32%; float:left; margin: 5px;}
  .blok {float:left; margin: 10px;}
  </style>
</head>
<body>
  <div>
    <img class="blok" src="pust2.jpg" alt="pustynia">
    <p>Wartość określa sposób pozycjonowania elementu i może być: static - przywraca normalne pozycjonowanie elementu. Przydatne wtedy, gdy wcześniej, np. w arkuszach stylów, została podana deklaracja pozycjonowania tego typu elementów. Użycie w definicji stylu inline dla wybranego elementu deklaracji position: static zniesie tę deklarację dla danego elementu.</p>
  </div>
  <div>
    <img class="blok" src="pust1.jpg" alt="pustynia">
    <p>Wartość określa sposób pozycjonowania elementu i może być: static - przywraca normalne pozycjonowanie elementu. Przydatne wtedy, gdy wcześniej, np. w arkuszach stylów, została podana deklaracja pozycjonowania tego typu elementów. Użycie w definicji stylu inline dla wybranego elementu deklaracji position: static zniesie tę deklarację dla danego elementu.</p>
  </div>
  <div>
    <img class="blok" src="pust3.jpg" alt="pustynia">
    <p>Wartość określa sposób pozycjonowania elementu i może być: static - przywraca normalne pozycjonowanie elementu. Przydatne wtedy, gdy wcześniej, np. w arkuszach stylów, została podana deklaracja pozycjonowania tego typu elementów. Użycie w definicji stylu inline dla wybranego elementu deklaracji position: static zniesie tę deklarację dla danego elementu.</p>
  </div>
</body>
</html>
```

Na rysunku 3.49 przedstawiono wynik wykonania kodu.

Rysunek 3.49. Dokument z trzema kolumnami

Ćwiczenie 3.15

Przygotuj krótkie teksty z informacją o zimowej olimpiadzie w Pjongczang oraz obrazy pokazujące zmagania sportowców. Za pomocą poznanych znaczników języka HTML zaprojektuj szablon strony internetowej na ten temat. Zaproponuj teksty, które znajdą się w nagłówku i stopce strony. Umieść przygotowane wcześniej teksty i obrazy w odpowiednich blokach. Wykorzystując style CSS i poznane narzędzia, zaprojektuj stronę internetową.

Ćwiczenie 3.16

Utwórz na stronie galerię zdjęć na temat samochodów przyszłości. Rozplanuj położenie obrazków tak, aby obok siebie wyświetlane były po trzy zdjęcia. Układ zdjęć nie powinien zostać zaburzony przy zmianie wielkości okna przeglądarki. Określ minimalny rozmiar każdego zdjęcia.

3.6.4. Sposoby wyświetlania elementów

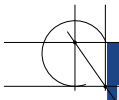
Wszystkie definiowane elementy są wyświetlane na stronie w sposób domyślny. Jeżeli chcemy zrezygnować z takiego pokazywania elementu, możemy zdefiniować atrybut `display`. Za jego pomocą można określić, jak dany element będzie wyświetlany na stronie.

```
selektor {display: sposób;}
```

Parametr atrybutu określa sposób pokazywania elementu na stronie i może przyjmować wartości:

- `block` — element wyświetlany jako element blokowy z odstępami od góry i od dołu;
- `inline` — element wyświetlany w linii z innymi elementami;
- `list-item` — element wyświetlany jako element wykazu listy `<li>`;
- `none` — element nie będzie wyświetlany;

- `inline-block` — element wyświetlany podobnie jak element zastępowany; mimo że element ma charakter blokowy (odstęp, obramowanie, np. `<div>`), jest wyświetlany jako liniowy;
- `run-in` — jeżeli po elemencie występuje element będący blokiem, to ten pierwszy element zostanie wyświetlony w jednej linii z blokiem, w przeciwnym razie element zostanie wyświetlony w bloku.



UWAGA

Element zastępowany to element, dla którego istnieje tylko wymiar wewnętrzny, np.: `<img>` (zawartość elementu jest zastępowana obrazem określonym przez atrybut `src`), `<input>`, `<textarea>`, `<select>`, `<object>`.

Atrybut `display` jest bardzo przydatny, jeżeli chcemy zrezygnować z domyślnego sposobu wyświetlania. Jeżeli np. element jest wyświetlany w bloku (czyli dodawane są linie odstępu między sąsiednimi elementami), to po zastosowaniu `display: inline` elementy zostaną ustawione w jednej linii.

Element blokowy

Elementy blokowe zawsze są wyświetlane od nowej linii i zajmują całą dostępną na stronie szerokość. Elementami blokowymi są m.in.:

- `<div>`,
- `<h1>` … `<h6>`,
- `<p>`,
- `<form>`,
- `<header>`,
- `<footer>`,
- `<section>`.

Element wbudowany (element InLine)

Element wbudowany (*InLine*) jest wyświetlany w miejscu wstawienia (nie rozpoczyna się od nowej linii) i zajmuje na szerokość tylko tyle miejsca, ile potrzeba na wyświetlenie jego zawartości. Elementami wbudowanymi są m.in.:

- `<span>`,
- `<a>`,
- `<img>`.

Atrybut `display` może zostać wykorzystany do zamiany sposobu wyświetlania elementów strony. I tak element *InLine* może zostać zmieniony na element blokowy, a element blokowy na element *InLine*.

Przykładem jest tworzenie za pomocą elementu <li> menu poziomego.

Domyślne wyświetlanie elementu <li> zostało pokazane na rysunku 3.50.

Rysunek 3.50. Domyślne wyświetlanie elementów

Podaną listę utworzoną za pomocą elementu blokowego przekształcimy na element *InLine*, aby uzyskać menu poziome.

Przykład 3.93

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Elementy wbudowane</title>
  <meta charset="UTF-8">
  <style>
  body {background-color: #d9d9d9;}
  li {display: inline;}
  </style>
</head>
<body>
  <h3>Lista kontynentów:</h3>
  <ul>
    <li><a href="strona_E.html">Europa</a></li>
    <li><a href="strona_Az.html">Azja</a></li>
    <li><a href="strona_Af.html">Afryka</a></li>
  </ul>
</body>
</html>
```

Rysunek 3.51 przedstawia wynik wykonania kodu.

Rysunek 3.51. Zastosowanie atrybutu display: InLine

W podobny sposób element typu *InLine*, np. element <a>, można przekształcić w element blokowy.

Przykład 3.94

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Zmiana sposobu wyświetlania elementu</title>
  <meta charset="UTF-8">
  <style>
  a {display: block;}
  </style>
</head>
<body>
  <h3>Lista kontynentów:</h3>
  <a href="strona_E.html">Europa</a>
  <a href="strona_Az.html">Azja</a>
  <a href="strona_Af.html">Afryka</a>
</body>
</html>
```

Wartość inline-block

Elementy, którym został nadany atrybut `display` z wartością `inline-block`, są elementami wbudowanymi, ale mogą posiadać szerokość (`width`) i wysokość (`height`).

Do ustawiania elementu względem innych elementów, które z nim sąsiadują w poziomie, służy atrybut `float`. Atrybut `clear` pozwala ustalić pozycję kolejnych elementów względem elementu pływającego.

Przykład 3.95

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Pływanie elementów</title>
  <meta charset="UTF-8">
  <style>
  .blok_p {
    float: left;
```

```
    width: 150px;
    height: 120px;
    margin: 10px;
    border: 3px solid #d9d9d9;
  }
  .blok_s {
    clear: left;
    border: 3px solid green;
  }
  </style>
</head>
<body>
  <h2>Zdjęcia z podróży</h2>
  <div class="blok_p"><img src="czad.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="utah.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="bejr3.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="czad.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="utah.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="bejr3.jpg" alt="obrazek"></div>
  <div class="blok_s"><img src="bruks.jpg" alt="obrazek"></div>
</body>
</html>
```

Podobny efekt można uzyskać za pomocą wartości `inline-block` atrybutu `display` i wtedy nie ma potrzeby użycia atrybutu `clear`.

Przykład 3.96

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Działanie wartości inline-block</title>
  <meta charset="UTF-8">
  <style>
  .blok_p {
    display: inline-block;
    width: 150px;
```

```
    height: 120px;
    margin: 10px;
    border: 3px solid #d9d9d9;
  }
  .blok_s {
    border: 3px solid green;
  }
  </style>
</head>
<body>
  <h2>Zdjęcia z podróży</h2>
  <div class="blok_p"><img src="czad.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="utah.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="bejr3.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="czad.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="utah.jpg" alt="obrazek"></div>
  <div class="blok_p"><img src="bejr3.jpg" alt="obrazek"></div>
  <div class="blok_s"><img src="bruks.jpg" alt="obrazek"></div>
</body>
</html>
```

Ukrywanie elementów

Ukrywanie elementów strony można uzyskać po ustawieniu atrybutu `display` na `none`. Innym atrybutem, za pomocą którego można sterować wyświetlaniem elementów na stronie, jest atrybut `visibility`.

```
selektor {visibility: typ;}
```

Jako wartość parametru *typ* należy podać:

- `visible` — element będzie widoczny na ekranie.
- `hidden` — element będzie ukryty, niewidoczny na ekranie. W jego miejscu pojawi się pusta przestrzeń.
- `collapse` — zastosowana dla tabeli ukrywa całą zawartość wiersza lub kolumny. Dla innych elementów działa jak `hidden`.

Podobny efekt jak w przypadku `visibility: hidden` można uzyskać, stosując atrybut `display: none`. Różnica polega na tym, że jeśli użyjemy `display: none`, element zostanie całkowicie usunięty ze strony.

3.7. Menu w języku CSS

Zaprojektowane na stronie internetowej menu umożliwia nawigację w jej obrębie. Paski nawigacyjne buduje się najczęściej na podstawie standardowej listy nieuporządkowanej. Lista jest zbiorem występujących po sobie pozycji, podobnie budowane jest menu. Za pomocą języka CSS można utworzoną listę zamienić w atrakcyjne menu w postaci pasków nawigacyjnych.

Lista menu

Budowanie menu zaczynamy od utworzenia listy w języku HTML. Lista taka powinna zawierać linki do odpowiednich miejsc wywoływanych przez menu.

Przykład 3.97

```
<ul>
  <li><a href="start.html">Start</a></li>
  <li><a href="nowe.html">Aktualności</a></li>
  <li><a href="szkolenia.html">Szkolenia</a></li>
  <li><a href="uslugi.html">Usługi</a></li>
</ul>
```

Po zdefiniowaniu zawartości menu można przystąpić do jego formatowania z użyciem stylów CSS. Dołączenie deklaracji stylów pozwoli na stworzenie elastycznego i funkcjonalnego menu nawigacyjnego. Sam wygląd menu zostanie zmieniony tylko przez dołączenie arkusza stylów z odpowiednimi deklaracjami.

Dobrym rozwiązaniem jest zdefiniowanie dla znacznika `<ul>` identyfikatora lub klasy, aby w wyniku formatowania tworzonego menu nie doprowadzić do przeformatowania zwykłych list, które mogą się znaleźć na stronie. Pozwoli to również na umieszczanie na stronie kilku różnie wyglądających zestawów menu.

Zanim przystąpimy do definiowania menu, na przygotowanej liście trzeba usunąć punktory oraz wyzerować marginesy. Należy również zadeklarować wyświetlanie listy w bloku oraz określić szerokość tego bloku.

Przykład 3.98

```
ul {display: block; list-style-type: none; margin: 0; padding: 0;}
```

Tak zapisana deklaracja w arkuszu stylów jest podstawą do tworzenia pionowych i poziomych pasków nawigacyjnych.

3.7.1. Menu pionowe

Aby utworzyć pionowe menu nawigacyjne, należy do podanych deklaracji dla list dołączyć deklarację dotyczącą znacznika `<a>` występującego w zdefiniowanej liście oraz znacznika `<li>`.

Przykład 3.99

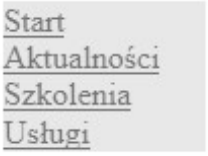
```
li a {display: block; width:100;}
```

Atrybut `display: block` wyświetli linki jako elementy blokowe, a `width:100` ustawi szerokość menu.

Przykład 3.100

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Menu pionowe</title>
  <meta charset="UTF-8">
  <style>
  ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    width: 100px;
  }
  li a {
    display: block;
    background-color: #dddddd;
  }
  </style>
</head>
<body>
  <ul>
    <li><a href="strona_n.html">Start</a></li>
    <li><a href="strona_a.html">Aktualności</a></li>
    <li><a href="strona_k.html">Szkolenia</a></li>
    <li><a href="strona_m.html">Usługi</a></li>
  </ul>
</body>
</html>
```

W wyniku wykonania kodu powstanie menu pionowe (rysunek 3.52).



Rysunek 3.52. Podstawowe menu pionowe

Utworzoną listę można za pomocą kaskadowych arkuszy stylów zmienić w atrakcyjne menu.

Przykład 3.101

```
ul a:link, ul a:visited {
  display: block;
  width: 168px;
  text-decoration: none;
  background-color: #c0c0c0;
  color: black;
  padding: 4px;
  border: 2px outset #c0c0c0;
}
ul a:hover {
  border-style: inset;
  color: blue;
  padding: 6px 3px 3px 6px;
}
```

W tej części definicji stylów został zdefiniowany wygląd przycisków menu.

Każdy z elementów listy zostanie wyświetlony w bloku, w związku z tym nie tylko tekst, ale cały blok stanie się odnośnikiem.

Deklaracje `a:link` oraz `a:visited` określają wygląd przycisków przed odwiedzeniem strony lub po jej odwiedzeniu (zniesione podkreślenie, ustalony kolor tła i kolor tekstu, margines wewnętrzny `4px` i ramka `2px` z ustawionym stylem ramki).

Deklaracja `a:hover` określa wygląd przycisku, gdy znajdzie się nad nim kursor myszy (zmiana stylu obramowania oraz przesunięcie marginesu wewnętrznego).

Przykładowe menu pionowe zostało pokazane na rysunku 3.53.

Rysunek 3.53. Wygląd menu zdefiniowany za pomocą arkuszy stylów

Rozwiązaniem przykładu pokazanego na rysunku 3.53 może być kod z przykładu 3.102.

Przykład 3.102

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Menu pionowe</title>
  <meta charset="UTF-8">
  <style>
  ul, ul li {
    display: block;
    list-style: none;
    margin: 0;
    padding: 0;
  }
  ul {
    width: 200px;
  }
  ul a:link, ul a:visited {
    display: block;
    width: 166px;
    text-decoration: none;
    background-color: #c0c0c0;
    color: black;
    padding: 4px;
    border: 2px outset #c0c0c0;
  }
  ul a:hover {
    border-style: inset;
    color: blue;
    padding: 6px 3px 3px 6px;
  }
  </style>
</head>
<body>
  <ul>
    <li><a href="start.html">Start</a></li>
```

```
    <li><a href="nowe.html">Aktualności</a></li>
    <li><a href="szkolenia.html">Szkolenia</a></li>
    <li><a href="uslugi.html">Usługi</a></li>
  </ul>
</body>
</html>
```

Ćwiczenie 3.17

Wykorzystując poznane narzędzia, utwórz na stronie internetowej menu pionowe. Zmieniając deklaracje stylów w arkuszu stylów, nadaj elementom menu atrakcyjny wygląd.

3.7.2. Menu poziome

Przy tworzeniu menu poziomego również korzystamy ze zdefiniowanej wcześniej listy nieuporządkowanej.

Domyślnie lista jest tworzona w układzie pionowym. Ułożenie pozycji listy obok siebie można uzyskać na dwa sposoby:

- wyświetlając kolejne elementy listy w linii (atrybut `display: inline;`);
- ustawiając pływanie dla każdej pozycji listy (atrybut `float`).

Podobnie jak w przypadku menu pionowego, w przygotowanym menu należy usunąć punktory oraz wyzerować marginesy.

Przykład 3.103

```
ul, li {display: block; list-style: none; margin: 0; padding: 0;}
```

Wyświetlanie kolejnych elementów listy w linii

Tak jak przy menu pionowym, utworzoną listę można sformatować za pomocą kaskadowych arkuszy stylów. Wyświetlanie kolejnych elementów listy w linii zostało zrealizowane za pomocą atrybutu `display: inline`.

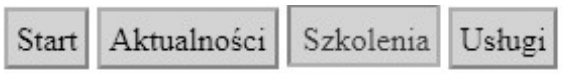
Przykład 3.104

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Menu poziome_1</title>
  <style>
  ul, ul li {
```

```
    display: block;
    list-style: none;
    margin: 0;
    padding: 0;
  }
  ul li {
    display: inline;
    white-space: nowrap;
  }
  ul a:link, ul a:visited {
    text-decoration: none;
    background-color: #c0c0c0;
    color: black;
    padding: 4px;
    border: 2px outset #c0c0c0;
  }
  ul a:hover {
    border-style: inset;
    color: blue;
    padding: 6px 3px 3px 6px;
  }
  </style>
</head>
<body>
  <ul>
    <li><a href="start.html">Start</a></li>
    <li><a href="nowe.html">Aktualności</a></li>
    <li><a href="szkolenia.html">Szkolenia</a></li>
    <li><a href="uslugi.html">Usługi</a></li>
  </ul>
</body>
</html>
```

Dodane polecenie `white-space: nowrap;` blokuje zawijanie tekstu w każdej pozycji menu. Jeśli wszystkie pozycje menu nie zmieszczą się w jednej linii, podział nastąpi między elementami listy, a nie wewnątrz elementu. Elementy zmieniają swoje położenie

w odpowiedzi na `hover`. Po najechaniu na wybrany element menu zmienia on swoje położenie (rysunek 3.54).



Rysunek 3.54. Menu poziome — kolejne elementy listy w linii

Metoda ta daje poprawne efekty i nie wpływa na elementy znajdujące się poniżej menu. Nie można jednak za jej pomocą zdefiniować jednakowej szerokości pozycji menu.

Float dla każdej pozycji listy

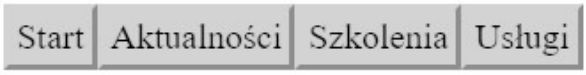
Wyświetlanie kolejnych elementów listy w linii zostało zrealizowane za pomocą atrybutu `float`.

Przykład 3.105

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Menu poziome</title>
  <meta charset="UTF-8">
  <style>
  ul li {
    display: inline;
    white-space: nowrap;
  }
  ul, li {
    float: left;
  }
  ul a:link, ul a:visited {
    text-decoration: none;
    background-color: #c0c0c0;
    color: black;
    padding: 4px;
    border: 2px outset #c0c0c0;
  }
  ul a:hover {
    border-style: inset;
```

```
    color: blue;
    padding: 6px 3px 3px 6px;
  }
  </style>
</head>
<body>
  <ul>
    <li><a href="start.html">Start</a></li>
    <li><a href="nowe.html">Aktualności</a></li>
    <li><a href="szkolenia.html">Szkolenia</a></li>
    <li><a href="uslugi.html">Usługi</a></li>
  </ul>
</body>
</html>
```

Efekt zastosowania tej metody ilustruje rysunek 3.55.



Rysunek 3.55. Menu poziome z zastosowaniem atrybutu float

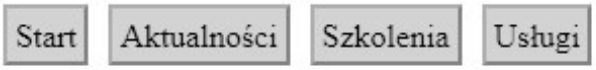
Wygląd menu można poprawić, umieszczając elementy w blokach. Jeżeli elementy zostaną umieszczone w bloku, będzie można zdefiniować dla nich stałą szerokość.

Przykład 3.106

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Menu poziome_1</title>
  <meta charset="UTF-8">
  <style>
  ul, ul li {
    display: block;
    list-style: none;
    margin: 0;
    padding: 0;
  }
  ul li {
```

```
    float: left;
    margin-right: 10px;
  }
  ul a:link, ul a:visited {
    display: block;
    position: relative;
    text-decoration: none;
    background-color: #c0c0c0;
    color: black;
    padding: 4px;
    border: 2px outset #c0c0c0;
  }
  ul a:hover {
    border-style: inset;
    color: blue;
    padding: 6px 3px 3px 6px;
  }
  </style>
</head>
<body>
  <ul>
    <li><a href="start.html">Start</a></li>
    <li><a href="nowe.html">Aktualności</a></li>
    <li><a href="szkolenia.html">Szkolenia</a></li>
    <li><a href="uslugi.html">Usługi</a></li>
  </ul>
</body>
</html>
```

Po umieszczeniu elementów menu w blokach wynik w przeglądarce jest prawidłowy (rysunek 3.56).



Rysunek 3.56. Elementy menu umieszczone w blokach

Dodatkowo można określić stałą szerokość bloków i wyśrodkowanie tekstu menu.

Przykład 3.107

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Menu poziome_2</title>
  <meta charset="UTF-8">
  <style>
  ul, ul li {
    display: block;
    list-style: none;
    margin:0;
    padding:0;
  }
  ul li {
    float: left;
    margin-right: 5px;
  }
  ul a:link, ul a:visited {
    display: block;
    width: 100px;
    text-align: center;
    position: relative;
    text-decoration: none;
    background-color: #c0c0c0;
    color: black;
    padding: 4px;
    border: 2px outset #c0c0c0;
  }
  ul a:hover {
    border-style: inset;
    color: blue;
    padding: 6px 3px 3px 6px;
  }
  </style>
</head>
```

```
<body>
  <ul>
    <li><a href="start.html">Start</a></li>
    <li><a href="nowe.html">Aktualności</a></li>
    <li><a href="szkolenia.html">Szkolenia</a></li>
    <li><a href="uslugi.html">Usługi</a></li>
  </ul>
</body>
</html>
```

W wyniku wykonania kodu zostanie utworzone menu poziome jak na rysunku 3.57.

Rysunek 3.57. Elementy menu z określoną stałą szerokością

Umieszczenie elementów w blokach daje możliwość zdefiniowania jednakowej szerokości wszystkich pozycji menu oraz ustalenia odległości między pozycjami. Mogą się natomiast pojawić problemy z prawidłowym wyświetlaniem w niektórych przeglądarkach. Trzeba również pamiętać o przypisaniu właściwości `clear` następnemu elementowi strony w celu uniknięcia „przyklejenia" tego elementu do menu.

Ćwiczenie 3.18

Zdefiniuj menu poziome, które będzie zawierało poziomy pasek nawigacyjny w wybranym kolorze tła. Gdy wskaźnik myszy zostanie ustawiony nad linkiem, kolor tła wybranego linku ulegnie zmianie (rysunek 3.58).

Rysunek 3.58. Przykład menu poziomego

Rozwiązanie

Rozwiązaniem ćwiczenia 3.18 może być następujący kod:

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Menu poziome</title>
  <meta charset="UTF-8">
```

```
  <style>
  ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #33415d;
  }
  li {
    float: left;
  }
  li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
  }
  li a:hover {
    background-color: #0d6ba1;
  }
  </style>
</head>
<body>
  <ul>
    <li><a href="start.html">Start</a></li>
    <li><a href="nowe.html">Aktualności</a></li>
    <li><a href="szkolenia.html">Szkolenia</a></li>
    <li><a href="uslugi.html">Usługi</a></li>
  </ul>
</body>
</html>
```

3.7.3. Menu zagnieżdżone

Podobnie jak w przypadku menu pionowego lub poziomego, budując menu zagnieżdżone, należy zacząć od utworzenia listy. Lista powinna zawierać linki do odpowiednich miejsc wywoływanych przez menu.

Przykład 3.108

```
<ul id="blok">
  <li><a href="start.html">Start</a></li>
  <li><a href="aktualnosci.html">Aktualności</a></li>
  <li><a href="szkolenia.html">Szkolenia</a>
    <ul>
      <li><a href="kurs1.html">HTML</a></li>
      <li><a href="kurs2.html">CSS</a></li>
      <li><a href="kurs3.html">JavaScript</a></li>
    </ul>
  </li>
  <li><a href="uslugi.html">Usługi</a></li>
</ul>
```

Po zdefiniowaniu zawartości menu można przystąpić do tworzenia menu zagnieżdżonego. Ponieważ zaproponowana lista ma podlistę tylko na drugiej pozycji, nasze menu będzie się rozwijało jedynie na tej właśnie pozycji.

Tak jak w przypadku menu pionowego i poziomego, w przygotowanym menu trzeba usunąć punktory oraz wyzerować marginesy.

Przykład 3.109

```
ul, ul li {margin: 0; padding: 0;}
#blok li {list-style: none;}
```

W definicji stylów został zdefiniowany identyfikator `#blok`, który w treści dokumentu HTML został przypisany do znacznika listy `<ul>`.

Następnym etapem jest zdefiniowanie wyglądu menu.

Przykład 3.110

```
#blok, #blok ul {
  width: 140px;
}
#blok li {
  padding-bottom: 1px;
```

```
  position: relative;
}
#blok ul {
  position: absolute;
  top: 0;
  left: 160px;
  padding-left: 1px;
  visibility: hidden;
}
#blok li:hover ul {
  visibility: visible;
  width: auto;
  height: auto;
}
```

Listom została nadana szerokość `140px`, elementy listy zostały rozdzielone przerwą (`padding-bottom: 1px`). Zdefiniowano położenie względne dla listy głównej w stosunku do strony internetowej oraz położenie podlisty w stosunku do listy głównej.

Dla wewnętrznej listy (`#blok ul`) zdefiniowano pozycjonowanie absolutne w stosunku do listy głównej oraz jej ukrycie (`visibility: hidden;`).

Gdy ustawimy kursor myszy na elemencie menu, wyświetli się podmenu.

Pozostało jeszcze zdefiniowanie, jak ma wyglądać element listy, gdy ustawimy nad nim kursor myszy. Aby element menu głównego był nadal podświetlony, gdy zostanie wybrany element podmenu, należy odpowiednio wykorzystać zasady dziedziczenia.

Przykład 3.111

```
#blok a, #blok li:hover li a {
  display: block;
  font: 12px/30px verdana, sans-serif;
  text-decoration: none;
  padding: 0 10px;
  width: 140px;
  background-color: #c0c0cf;
}

#blok li:hover a, #blok li:hover li:hover a {
  background-color: #333033;
  color: white;
}
```

Pierwsza linia definiuje wygląd każdego elementu menu, druga — wygląd elementu wskazanego kursorem myszy. Jeżeli będzie to element podmenu, element z menu głównego też pozostanie podświetlony.

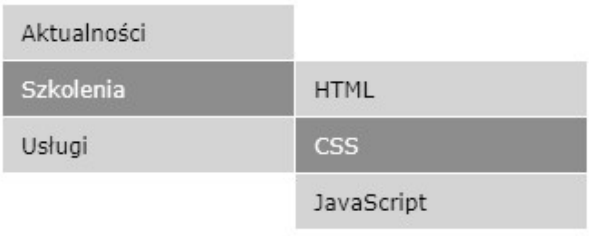
Efekt pracy pokazano w przykładzie poniżej.

Przykład 3.112

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Menu zagnieżdżone</title>
  <meta charset="UTF-8">
  <style>
  ul, ul li {
    margin: 0;
    padding: 0;
  }
  #blok li {
    list-style: none;
  }
  #blok, #blok ul {
    width: 140px;
  }
  #blok ul {
    position: absolute;
    top: 0;
    left: 160px;
    padding-left: 1px;
    visibility: hidden;
  }
  #blok li {
    padding-bottom: 1px;
    position: relative;
  }
  #blok li:hover ul {
    visibility: visible;
```

```
    width: auto;
    height: auto;
  }
  #blok a, #blok li:hover li a {
    display: block;
    font: 12px/30px verdana, sans-serif;
    text-decoration: none;
    padding: 0 10px;
    width: 140px;
    color: black;
    background-color: #c0c0cf;
  }
  #blok li:hover a, #blok li:hover li:hover a {
    background-color: #5f699c;
    color: white;
  }
  </style>
</head>
<body>
  <ul id="blok">
    <li><a href="aktualnosci.html">Aktualności</a></li>
    <li><a href="szkolenia.html">Szkolenia</a>
      <ul>
        <li><a href="kurs1.html">HTML</a></li>
        <li><a href="kurs2.html">CSS</a></li>
        <li><a href="kurs3.html">JavaScript</a></li>
      </ul>
    </li>
    <li><a href="uslugi.html">Usługi</a></li>
  </ul>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.59.



Rysunek 3.59.
Menu zagnieżdżone

Ćwiczenie 3.19

Korzystając z powyższego kodu, zdefiniuj menu zagnieżdżone dla pozostałych dwóch opcji menu głównego. Samodzielnie zaprojektuj dla tworzonej strony internetowej wygląd menu z zastosowaniem arkusza stylów i poleceń języka CSS.

3.8. Zasady projektowania szablonu strony internetowej

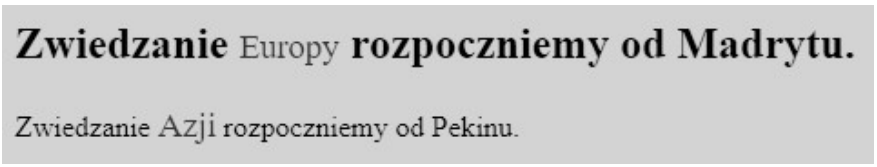
3.8.1. Definiowanie reguł

Reguły definiowane w arkuszu CSS dla znacznika dotyczą wszystkich użytych w dokumencie znaczników tego typu. Oznacza to, że jeżeli zdefiniujemy w arkuszu stylów wygląd znacznika `<span>`, to w każdym miejscu na stronie tekst otoczony znacznikiem `<span>` będzie wyglądał tak samo, niezależnie od tego, czy znajdzie się w nagłówku strony, w treści dokumentu, czy w menu.

Przykład 3.113

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Szablon strony - definiowanie reguł</title>
  <meta charset="UTF-8">
  <style>
  body {
    background-color: #c0c0c0;
  }
  span {
    font-size: 14pt;
    color: blue;
    font-weight: normal;
  }
  </style>
</head>
<body>
  <h2>Zwiedzanie <span>Europy</span> rozpoczniemy od Madrytu.</h2>
  <p>Zwiedzanie <span>Azji</span> rozpoczniemy od Pekinu.</p>
</body>
</html>
```

W podanym przykładzie znacznikiem został otoczony fragment tekstu umieszczony w nagłówku <h2> oraz fragment akapitu. Wynik wykonania kodu widać na rysunku 3.60. Teksty sformatowane z użyciem znacznika wyglądają tak samo.



Rysunek 3.60. Formatowanie tekstów otoczonych znacznikiem

Aby odróżnić przy formatowaniu podobne elementy umieszczane w różnych miejscach dokumentu, możemy zastosować jedną z trzech technik:

- reguły kontekstowe,
- identyfikatory,
- klasy.

Reguły kontekstowe

Reguły kontekstowe pozwalają na zdefiniowanie w arkuszu stylów wyglądu każdego elementu w zależności od jego położenia.

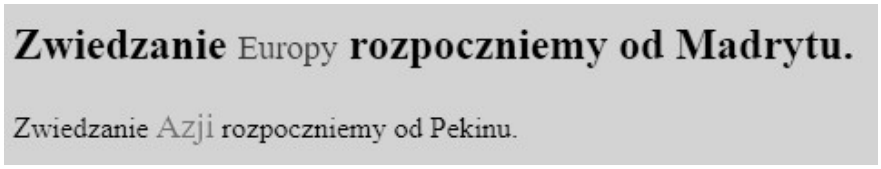
W podanym przykładzie teksty sformatowane z użyciem znacznika wyglądają tak samo, mimo że znajdują się w różnych miejscach i byłoby wskazane, aby wyglądały inaczej. Jeżeli w trakcie definiowania reguł określimy kontekst znacznika , będzie możliwe sformatowanie w inny sposób obu fragmentów tekstu.

Przykład 3.114

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Szablon strony - reguły kontekstowe</title>
  <meta charset="UTF-8">
  <style>
  body {
    background-color: #c0c0c0;
  }
  h2 span {
    font-size: 14pt;
    color: blue;
    font-weight: normal;
```

```
  }
  p span {
    font-size: 14pt;
    color: red;
    font-weight: normal;
  }
  </style>
</head>
<body>
  <h2>Zwiedzanie <span>Europy</span> rozpoczniemy od Madrytu.</h2>
  <p>Zwiedzanie <span>Azji</span> rozpoczniemy od Pekinu.</p>
</body>
</html>
```

W podanym przykładzie znacznik został zdefiniowany w kontekście nagłówka <h2> oraz akapitu <p>. Efekt jest taki, że każdy fragment otoczony znacznikiem wygląda inaczej (rysunek 3.61).



Rysunek 3.61. Zastosowanie reguł kontekstowych

Identyfikatory

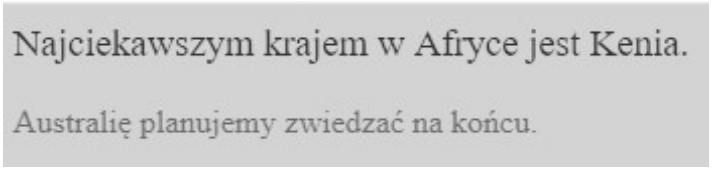
Nie zawsze możliwe jest definiowanie wyglądu elementów w zależności od kontekstu. Na przykład próba zdefiniowania różnego wyglądu dla dwóch akapitów znajdujących się w tym samym obszarze dokumentu przy użyciu reguł kontekstowych nie powiedzie się, ponieważ obydwa akapity występują w tym samym kontekście. W takiej sytuacji można zastosować identyfikatory. Identyfikator jednoznacznie określa wybrany element. Dzięki niemu każdy, nawet najmniejszy element strony może wyglądać inaczej.

Przykład 3.115

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Szablon strony - identyfikatory</title>
  <meta charset="UTF-8">
  <style>
```

```
  body {
    background-color: #c0c0c0;
  }
  p#ak1 {
    font-size: 14pt;
    color: blue;
  }
  p#ak2 {
    font-size: 12pt;
    color: green;
  }
  </style>
</head>
<body>
  <p id="ak1">Najciekawszym krajem w Afryce jest Kenia.</p>
  <p id="ak2">Australię planujemy zwiedzać na końcu.</p>
</body>
</html>
```

Każdy akapit będzie wyświetlony w inny sposób, ponieważ zostały im przypisane identyfikatory, a w arkuszu stylów dla każdego identyfikatora zdefiniowano inny sposób wyświetlania (rysunek 3.62).



Rysunek 3.62. Zastosowanie identyfikatorów

Klasy

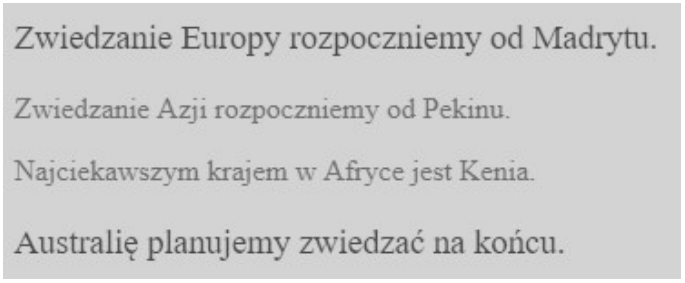
Czasami na stronie znajduje się grupa elementów, które powinny wyglądać tak samo. W takiej sytuacji nie trzeba każdemu elementowi przypisywać identyfikatora. Można te elementy połączyć w grupy i dla każdej grupy zdefiniować klasę, a następnie, używając pojedynczych reguł w CSS, za jej pomocą określić wygląd całej grupy elementów.

Przykład 3.116

```
<!DOCTYPE html>
<html lang="pl-PL">
```

```
<head>
  <title>Klasy</title>
  <meta charset="UTF-8">
  <style>
  body {
    background-color: #c0c0c0;
  }
  p.t1 {
    font-size: 14pt;
    color: #5e1138;
  }
  p.t2 {
    font-size: 12pt;
    color: #186d28;
  }
  </style>
</head>
<body>
  <p class="t1">Zwiedzanie Europy rozpoczniemy od Madrytu.</p>
  <p class="t2">Zwiedzanie Azji rozpoczniemy od Pekinu.</p>
  <p class="t2">Najciekawszym krajem w Afryce jest Kenia.</p>
  <p class="t1">Australię planujemy zwiedzać na końcu.</p>
</body>
</html>
```

Wynik zdefiniowania dwóch klas dla akapitu przedstawiono na rysunku 3.63.



Rysunek 3.63. Zastosowanie różnych klas do formatowania akapitów

Jeżeli wybrany element jest definiowany za pomocą kilku reguł, to obowiązują następujące priorytety:

1. Identyfikator.
2. Klasa.
3. Znacznik kontekstowy.
4. Znacznik.

Znacznik *

W modelu blokowym CSS wszystkie elementy można traktować jak prostokątne bloki. Dla każdego z tych bloków można definiować marginesy zewnętrzne i wewnętrzne oraz obramowania. Jeżeli dla elementu umieszczanego na stronie te parametry nie zostaną zdefiniowane, to przyjmą wartości domyślne. Różne przeglądarki w odmienny sposób interpretują wartości domyślne elementu, co może prowadzić do nieprawidłowego wyświetlania zawartości strony. Aby temu zapobiec, należy jako jedną z pierwszych deklaracji strony umieścić deklarację zerowania marginesów zewnętrznych i wewnętrznych dla wszystkich elementów występujących na stronie.

Specjalny znacznik, który możemy do tego wykorzystać, to selektor uniwersalny `*`. Zdefiniowanie dla niego reguły oznacza, że powstała reguła odnosi się do wszystkich znaczników na stronie.

Przykład 3.117

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

Bloki danych

Do rozmieszczania elementów na stronie stosuje się najczęściej omówione już elementy grupujące: `<div>`, `<header>`, `<nav>`, `<section>` i `<footer>`. Są to elementy, które grupują elementy strony, ale nie w celu ich formatowania, tylko w celu ich odpowiedniego rozmieszczenia. Znaczniki te są odpowiednikami ramek, w których zgrupowane elementy można dowolnie przemieszczać na stronie. Znacznikom grupującym można przypisywać identyfikatory lub klasy. Poprzez deklarację reguły dla identyfikatorów i klas każdy znacznik może być formatowany. Wewnątrz znacznika grupującego mogą być umieszczane inne elementy, np.: obrazy, teksty lub kolejne znaczniki.

Elementy `<div>`, `<header>`, `<nav>`, `<section>` i `<footer>` tworzą bloki, w których są umieszczane inne elementy, może więc się zdarzyć, że element umieszczony wewnątrz znacznika grupującego jest szerszy niż wielkość tego znacznika i nie mieści się w nim. W takim przypadku zbyt szerokie elementy mogą zachodzić na sąsiednie bloki, zasłaniając ich zawartość, lub blok może zostać przeniesiony niżej. Takie zachowanie bloków niszczy układ całej strony.

Aby zachować układ strony, można dla podanych znaczników umieszczanych na stronie zdefiniować atrybut `overflow` jako `hidden`. Atrybut `overflow` definiuje sposób wyświetlania, gdy zawartość nie mieści się w zdefiniowanym rozmiarze bloku; wartość `hidden` wymusi obcięcie niemieszczącego się fragmentu. Fragment zawartości przestanie być widoczny na stronie, ale zostanie zachowany układ strony.

Przykład 3.118

```
div {overflow: hidden;}
```

WSKAZÓWKA

Na początku pracy nad nową stroną wskazane jest zdefiniowanie dla wszystkich znaczników grupujących atrybutu `overflow: hidden`.

Kolory

Dobrym zwyczajem jest również zdefiniowanie koloru tła i koloru tekstu. Domyślny kolor tła to biały, a tekstu — czarny. Nie można jednak zakładać, że są to kolory obowiązujące u wszystkich użytkowników naszej strony. Jeżeli użytkownik zmieni kolory domyślne w systemie operacyjnym swojego komputera, to wyświetlając zaprojektowaną przez nas stronę, może uzyskać nieoczekiwane efekty kolorystyczne.

Przykład 3.119

```
body {background-color: #fff; color: #000;}
```

Wymiary strony

Przystępując do projektowania szablonu strony o stałej szerokości, należy ustalić tę szerokość. Zwykle przyjmuje się, że strona internetowa jest przygotowana dla ekranu o rozdzielczości 1024 pikseli. Oczywiście można wybrać dowolną szerokość, ale trzymając się standardów obowiązujących w internecie, uzyskamy lepsze wyniki wyświetlania strony w dowolnej przeglądarce.

Przyjmując, że na stronie pojawia się suwak do przewijania treści, przy zerowych marginesach strony szerokość szablonu powinna wynosić 1000 px. Jeżeli dla projektowanego szablonu strony zostanie zdefiniowany zewnętrzny element, np. `<div>`, w którym będzie umieszczona cała zawartość strony, i nadamy mu identyfikator `ramka`, to właśnie ten element powinien mieć szerokość 1000 px.

Przykład 3.120

```
#ramka {
  width: 1000px;
  margin-right: auto;
  margin-left: auto;
}
```

Aby strona była zawsze widoczna na środku przeglądarki, niezależnie od rozmiaru jej okna, należy ustawić wartości marginesów prawego i lewego na `auto`. Oznacza to, że

obydwa marginesy będą miały taką samą wartość i że będzie ona automatycznie dostosowywana do szerokości okna przeglądarki.

3.8.2. Szablon strony

Pamiętajmy, że używając znaczników `<div>`, `<header>`, `<nav>`, `<section>` i `<footer>`, możemy grupować w bloki elementy strony i formatować je za pomocą stylów. Bloki te mogą zawierać akapity, listy, obrazy oraz inne bloki. Umożliwiają one wydzielanie na stronie większych, logicznych fragmentów w celu nadania im specyficznego formatowania za pomocą stylów CSS. Z tych powodów wymienione bloki świetnie nadają się do budowy szablonu strony internetowej. Ponieważ tak zbudowana strona może zawierać wiele różnych znaczników, w celu ich odróżnienia warto każdemu z nich przypisać identyfikator (atrybut `id`). Jeżeli definiujemy znaczniki, które mogą się powtórzyć na stronie, należy przypisać je do klasy (atrybut `class`).

W praktyce najczęściej mamy do czynienia z szablonami typu:

- nagłówek, jedna kolumna, stopka;
- nagłówek, dwie kolumny, stopka;
- nagłówek, trzy kolumny, stopka.

Jeżeli zajrzymy na najpopularniejsze portale internetowe, najczęściej spotkamy wersję z dwiema lub trzema kolumnami. Witryny tematyczne często mają układ dwukolumnowy: pierwsza kolumna zawiera menu tematyczne, druga — treść. Mogą wystąpić różne modyfikacje tych układów, ale podstawowa struktura w postaci kolumn zostaje zachowana. Można planować układ z większą liczbą kolumn, ale przy wielu kolumnach strona staje się nieczytelna.

Szablon oparty na stałej liczbie kolumn jest najczęściej stosowaną formą układu strony. Jest on niezależny od rozdzielczości ekranu, a elementy rozplanowane na stronie zawsze pozostają w miejscach, w których zostały zaplanowane.

Szablon z trzema kolumnami

Na nasze potrzeby przygotujemy szablon strony z trzema kolumnami.

W pierwszym etapie musimy w języku HTML zdefiniować szablon strony przy użyciu znaczników blokowych i semantycznych.

Przykład 3.121

```
<body>
  <header id="naglowek">
    Tutaj znajdzie się zawartość nagłówka strony
    <nav>Tutaj będzie NAWIGACJA</nav>
  </header>
```

```
<div id="ramka">
  <section class="kolumny" id="kol1">
    <h3>Kolumna 1</h3>
    Tutaj znajdzie się zawartość kolumny1
    <article class="artyk" id="ar1_1">
      <h4>Artykuł 1_1</h4>
      Artykuł1 w kolumnie1
    </article>
    <article class="artyk" id="ar2_1">
      <h4>Artykuł 2_1</h4>
      Artykuł2 w kolumnie1
    </article>
  </section>
  <section class="kolumny" id="kol2">
    <h3>Kolumna 2</h3>
    Tutaj znajdzie się zawartość kolumny2
    <article class="artyk" id="ar1_2">
      <h4>Artykuł 1_2</h4>
      Artykuł1 w kolumnie2
    </article>
    <article class="artyk" id="ar2_2">
      <h4>Artykuł 2_2</h4>
      Artykuł2 w kolumnie2
    </article>
    <article class="artyk" id="ar3_2">
      <h4>Artykuł 3_2</h4>
      Artykuł3 w kolumnie2
    </article>
  </section>
  <section class="kolumny" id="kol3">
    <h3>Kolumna 3</h3>
    Tutaj znajdzie się zawartość kolumny3
    <article class="artyk" id="ar1_3">
      <h4>Artykuł 1_3</h4>
      Artykuł1 w kolumnie3
```

```
      </article>
    </section>
    <footer id="stopka">
      Tutaj znajdzie się zawartość stopki strony
    </footer>
  </div>
</body>
```

Tak przygotowany szablon stanowi strukturę strony, którą za pomocą stylów CSS można w dowolny sposób formatować. Projektując kolejne arkusze CSS dla tego samego kodu HTML, możemy zmieniać wygląd strony. Natomiast wypełniając go odpowiednią treścią, możemy tworzyć kolejne strony oparte na tym samym szablonie.

W przygotowanym szablonie wszystkie bloki zostały ułożone jeden pod drugim. Aby kolumny szablonu ułożyć pionowo obok siebie, należy użyć atrybutu `float`.

Atrybut ten był już wykorzystywany do oblewania ilustracji tekstem. Jeżeli dla kilku sąsiednich elementów nadamy atrybutowi `float` wartość `left`, to elementy te ustawią się obok siebie, pod warunkiem że pozwoli na to szerokość bloku, w którym się znajdują.

W związku z tym wystarczy dla kolumn zdefiniować regułę zawierającą atrybut `float`: `left`. Musimy pamiętać, aby suma szerokości wszystkich kolumn nie przekroczyła całkowitej szerokości strony.

Przykład 3.122

```
#kol1 {float: left; width: 30%;}
#kol2 {float: left; width: 30%;}
#kol3 {float: left; width: 30%;}
```

Lub

Przykład 3.123

```
.kolumny {float: left; width: 30%;}
```

Należy pamiętać również o tym, aby stopce nadać odpowiednie atrybuty uniemożliwiające jej wyświetlenie z lewej strony kolumn. Posłuży do tego atrybut `clear` z wartością `both`.

Przykład 3.124

```
#stopka {clear: both; width: 100%;}
```

Jeśli do projektowania szablonu strony wykorzystamy wcześniej poznane reguły projektowania, to kod szablonu strony może być podobny do podanego w poniższym przykładzie.

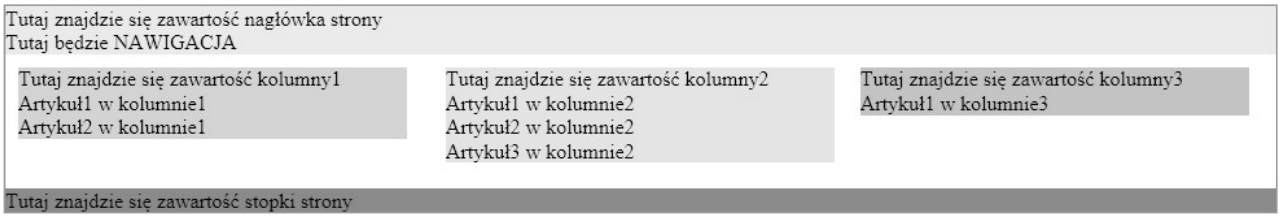
Przykład 3.125

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <meta charset="UTF-8">
  <title>Trzy kolumny</title>
  <style>
  * {
    margin: 0px;
    padding: 0px;
  }
  section {
    overflow: hidden;
  }
  #ramka {
    width: 1000px;
    margin-right: auto;
    margin-left: auto;
  }
  #naglowek {
    width: 100%;
    background-color: #FCF;
  }
  #stopka {
    width: 100%;
    background-color: #93F;
    clear: both;
  }
  .kolumny {
    width: 30%;
    margin-top: 10px;
    margin-right: 2%;
    margin-bottom: 2%;
    margin-left: 10px;
  }
  #kol1 {
```

```
    float: left;
    background-color: #3FF;
  }
  #kol2 {
    float: left;
    background-color: #CF3;
  }
  #kol3 {
    background-color: #F93;
  }
  </style>
</head>
<body>
  <header id="naglowek">
    Tutaj znajdzie się zawartość nagłówka strony
    <nav>Tutaj będzie NAWIGACJA</nav>
  </header>
  <div id="ramka">
    <section class="kolumny" id="kol1">
      <h3>Kolumna 1</h3>
      Tutaj znajdzie się zawartość kolumny1
      <article class="artyk" id="ar1_1">
        <h4>Artykuł 1_1</h4>
        Artykuł1 w kolumnie1
      </article>
      <article class="artyk" id="ar2_1">
        <h4>Artykuł 2_1</h4>
        Artykuł2 w kolumnie1
      </article>
    </section>
    <section class="kolumny" id="kol2">
      <h3>Kolumna 2</h3>
      Tutaj znajdzie się zawartość kolumny2
      <article class="artyk" id="ar1_2">
        <h4>Artykuł 1_2</h4>
        Artykuł1 w kolumnie2
```

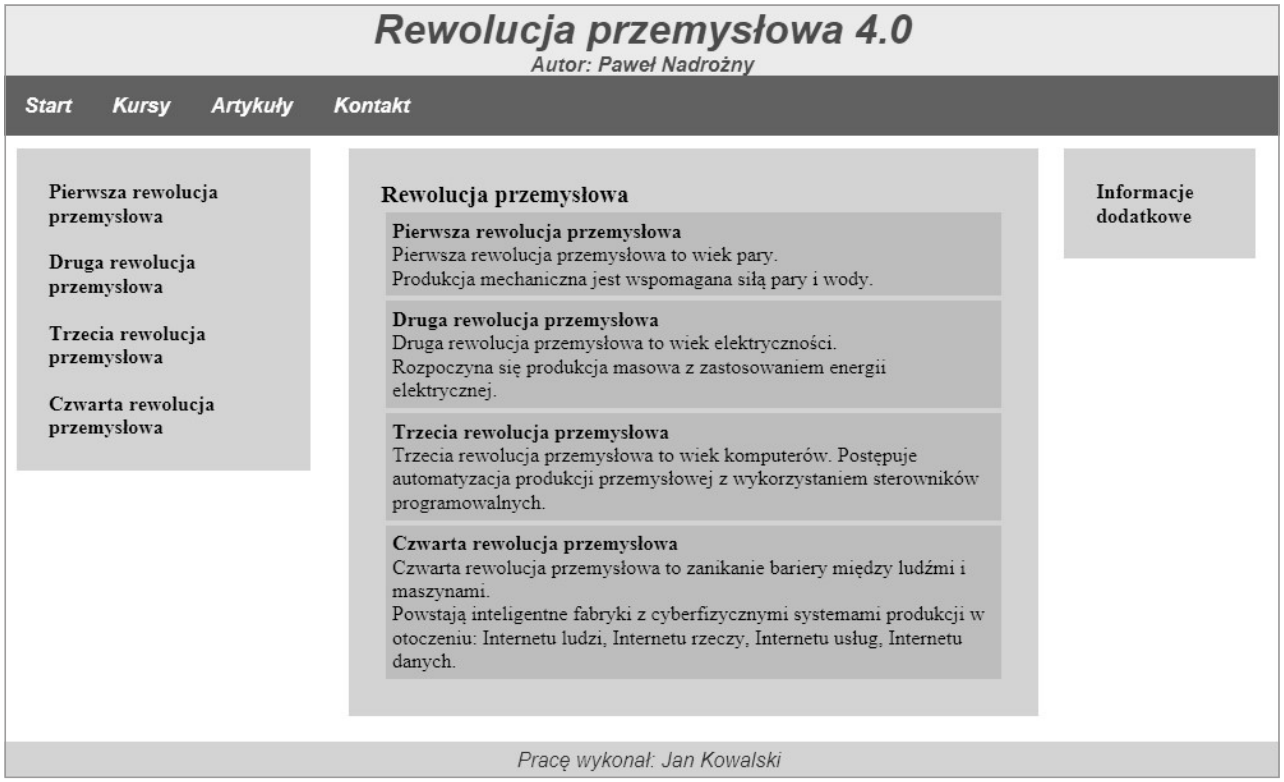
```
      </article>
      <article class="artyk" id="ar2_2">
        <h4>Artykuł 2_2</h4>
        Artykuł2 w kolumnie2
      </article>
      <article class="artyk" id="ar3_2">
        <h4>Artykuł 3_2</h4>
        Artykuł3 w kolumnie2
      </article>
    </section>
    <section class="kolumny" id="kol3">
      <h3>Kolumna 3</h3>
      Tutaj znajdzie się zawartość kolumny3
      <article class="artyk" id="ar1_3">
        <h4>Artykuł 1_3</h4>
        Artykuł1 w kolumnie3
      </article>
    </section>
    <footer id="stopka">
      Tutaj znajdzie się zawartość stopki strony
    </footer>
  </div>
</body>
</html>
```

Wynik wykonania kodu pokazuje rysunek 3.64.



Rysunek 3.64. Szablon strony z trzema kolumnami

Po określeniu przeznaczenia strony szablon może zostać zindywidualizowany, co pokazuje rysunek 3.65.



Rysunek 3.65. Zmodyfikowany szablon strony z trzema kolumnami

Realizacją podanego szablonu strony może być przedstawiony niżej kod.

Przykład 3.126

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <meta charset="utf-8">
  <title>Szablon strony</title>
  <style>
  * {
    margin: 0px;
    padding: 0px;
  }
  section {
    overflow: hidden;
    padding: 25px;
  }
  #ramka {
    width: 999px;
```

```
  margin-right: auto;
  margin-left: auto;
}
#naglowek {
  width: 100%;
  background-color: #fcf;
  font-size: 12pt;
  color: #5e1138;
  font-family: Arial;
  font-style: italic;
  text-align: center;
  font-weight: bold;
}
#stopka {
  width: 100%;
  background-color: #c0c0c0;
  color: blue;
  font-family: Arial;
  font-style: italic;
  padding: 5px;
  text-align: center;
  clear: both;
}
.kolumny {
  margin-top: 10px;
  margin-right: 2%;
  margin-bottom: 2%;
  margin-left: 10px;
  background-color: #c0c0c0;
}
.art {
  background-color: #9ca0c9;
  padding: 5px;
  margin: 4px;
}
```

```
  #kol1 {
    float: left;
    width: 180px;
  }
  #kol2 {
    float: left;
    width: 490px;
  }
  #kol3 {
    width: 100px;
  }
  ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #33415d;
  }
  li {
    float: left;
  }
  li a {
    display: block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;
  }
  li a:hover {
    background-color: #0d6ba1;
  }
  </style>
</head>
<body>
  <div id="ramka">
```

```
    <header id="naglowek">
      <h1>Rewolucja przemysłowa 4.0</h1>
      <p>Autor: Paweł Nadrożny</p>
      <nav>
        <ul>
          <li><a href="start.html">Start</a></li>
          <li><a href="nowe.html">Kursy</a></li>
          <li><a href="serwis.html">Artykuły</a></li>
          <li><a href="uslugi.html">Kontakt</a></li>
        </ul>
      </nav>
    </header>
    <section class="kolumny" id="kol1">
      <h4>Pierwsza rewolucja przemysłowa</h4><br>
      <h4>Druga rewolucja przemysłowa</h4><br>
      <h4>Trzecia rewolucja przemysłowa</h4><br>
      <h4>Czwarta rewolucja przemysłowa</h4>
    </section>
    <section class="kolumny" id="kol2">
      <h3>Rewolucja przemysłowa</h3>
      <article class="art">
        <h4>Pierwsza rewolucja przemysłowa</h4>
        <p>Pierwsza rewolucja przemysłowa to wiek pary.<br> Produkcja mecha-
niczna jest wspomagana siłą pary i wody.</p>
      </article>
      <article class="art">
        <h4>Druga rewolucja przemysłowa</h4>
        <p>Druga rewolucja przemysłowa to wiek elektryczności.</p>
        <p>Rozpoczyna się produkcja masowa z zastosowaniem energii
elektrycznej.</p>
      </article>
      <article class="art">
        <h4>Trzecia rewolucja przemysłowa</h4>
        <p>Trzecia rewolucja przemysłowa to wiek komputerów. Postępuje automa-
tyzacja produkcji przemysłowej z wykorzystaniem sterowników programowalnych.
</p>
```

```
      </article>
      <article class="art">
        <h4>Czwarta rewolucja przemysłowa</h4>
        <p>Czwarta rewolucja przemysłowa to zanikanie bariery między ludźmi
i maszynami.</p>
        <p>Powstają inteligentne fabryki z cyberfizycznymi systemami produk-
cji w otoczeniu: Internetu ludzi, Internetu rzeczy, Internetu usług, Internetu
danych.</p>
      </article>
    </section>
    <section class="kolumny" id="kol3">
      <h4>Informacje dodatkowe</h4>
    </section>
    <footer id="stopka">
      <p>Pracę wykonał: Jan Kowalski</p>
    </footer>
  </div>
</body>
</html>
```

Ćwiczenie 3.20

Wykorzystując poznane narzędzia, przygotuj własny szablon strony internetowej. Utwórz menu. Zmieniając deklaracje stylów w arkuszu stylów, nadaj elementom strony atrakcyjny wygląd.

3.8.3. Płynny szablon strony

Płynny szablon pozwala, aby przy zmianie szerokości okna przeglądarki zmieniała się szerokość kolumn strony. Najczęściej zmiana szerokości dotyczy tylko wybranych kolumn, np. kolumna z menu pozostaje niezmieniona, płynnie zmienia się natomiast szerokość kolumny z treścią.

Na nasze potrzeby przygotujemy szablon strony z dwiema kolumnami.

Przykład 3.127

```
<body>
  <div id="ramka">
    <header id="naglowek">
      Tutaj znajdzie się zawartość nagłówka strony
```

```
    </header>
    <section class="kolumny" id="kol1">
      <h2>Kolumna 1</h2>
      Tutaj znajdzie
      się zawartość kolumny1
    </section>
    <section class="kolumny" id="kol2">
      <h2>Kolumna 2</h2>
      Tutaj znajdzie się
      zawartość kolumny2
    </section>
    <footer id="stopka">
      Tutaj znajdzie się zawartość stopki strony
    </footer>
  </div>
</body>
```

Dla tak przygotowanego szablonu można, przy wykorzystaniu reguł CSS, określić sposób wyświetlania kolumn.

Przykład 3.128

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <meta charset="UTF-8">
  <title>Płynny szablon</title>
  <style>
  * {
    margin: 0px;
    padding: 0px;
  }
  section {
    overflow: hidden;
    padding: 25px;
  }
  #ramka {
```

```
    margin-right: auto;
    margin-left: auto;
    min-width: 400px;
    max-width: 900px;
  }
  #naglowek {
    background-color: #FCF;
    font-size: 16pt;
    color: #5e1138;
    font-family: Arial;
    font-style: italic;
    text-align: center;
    font-weight: bold;
  }
  #stopka {
    background-color: #c0c0c0;
    font-size: 14pt;
    color: blue;
    font-family: Arial;
    font-style: italic;
    clear: both;
  }
  .kolumny {
    margin-top: 10px;
    margin-right: 2%;
    margin-bottom: 2%;
    margin-left: 10px;
    background-color: #c0c0c0;
  }
  #kol1 {
    float: left;
    width: 100px;
  }
  </style>
</head>
```

```
<body>
<div id="ramka">
  <header id="naglowek">Nagłówek mojej strony</header>
  <section class="kolumny" id="kol1">
    <h3>Menu</h3>
  </section>
  <section class="kolumny" id="kol2">
    <h3>Treść</h3>
  </section>
  <footer id="stopka">Pracę wykonał: Jan Kowalski</footer>
</div>
</body>
</html>
```

Jeżeli nie zostaną określone szerokości kolumn ani szerokość szablonu, to szablon zajmie całe okno przeglądarki. Zmiana szerokości okna przeglądarki spowoduje automatyczną modyfikację zawartości strony i jej dostosowanie do aktualnych rozmiarów okna.

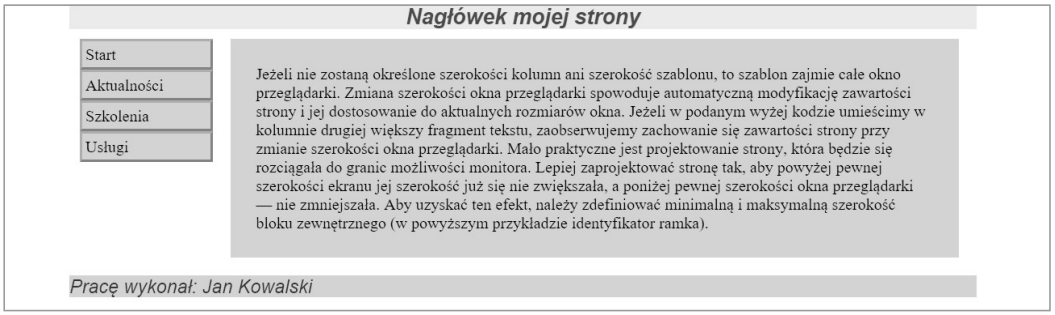
Jeżeli w podanym wyżej kodzie umieścimy w kolumnie drugiej większy fragment tekstu, zaobserwujemy zachowanie się zawartości strony przy zmianie szerokości okna przeglądarki.

Mało praktyczne jest projektowanie strony, która będzie się rozciągała do granic możliwości monitora. Lepiej zaprojektować stronę tak, aby powyżej pewnej szerokości ekranu jej szerokość już się nie zwiększała, a poniżej pewnej szerokości okna przeglądarki — nie zmniejszała. Aby uzyskać ten efekt, należy zdefiniować minimalną i maksymalną szerokość bloku zewnętrznego (w powyższym przykładzie identyfikator `ramka`).

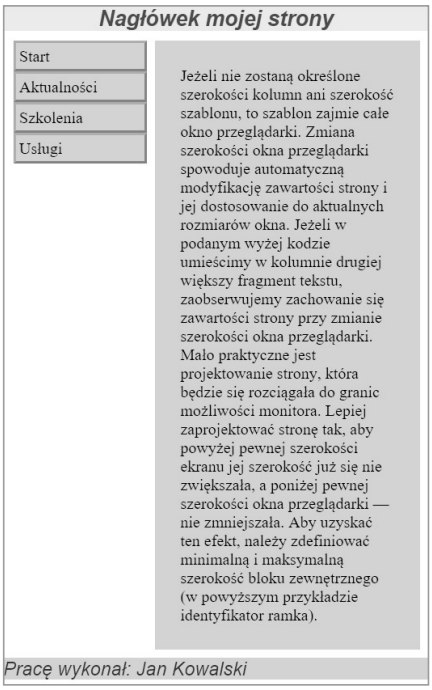
Przykład 3.129

```
#ramka {
  margin-right: auto;
  margin-left: auto;
  min-width: 400px;
  max-width: 900px;
}
```

Po ustawieniu tych parametrów zawartość strony będzie skalowana w zakresie podanych wartości. Dalsze zwiększanie szerokości strony spowoduje pokazanie tła strony (rysunek 3.66), a zmniejszanie szerokości poniżej zadeklarowanej spowoduje ukrycie części zawartości i wyświetlenie suwaków, za pomocą których będzie można obejrzeć ukryte fragmenty strony (rysunek 3.67).



Rysunek 3.66. Płynny szablon strony. Okno przeglądarki zmaksymalizowane



Rysunek 3.67. Płynny szablon strony. Okno przeglądarki zminimalizowane

Jeżeli planujemy umieszczenie menu w pierwszej kolumnie, to kolumna ta powinna mieć stałą szerokość, a dynamicznie zmieniać się powinna tylko szerokość drugiej kolumny.

Przykład 3.130

```
#kol1 {float: left; width: 100px;}
```

Jeśli taką deklarację wprowadzimy dla pierwszej kolumny, to dla drugiej kolumny nie musimy deklarować szerokości, ponieważ automatycznie wypełni ona pozostałą część strony. Należy pamiętać o jeszcze jednej zależności, mianowicie w przypadku układów o płynnej szerokości nie ma potrzeby ustawiania atrybutu `float` dla kolumny położonej z prawej strony. Dlatego w kodzie dokumentu trzeba usunąć ten parametr.

Ćwiczenie 3.21

Na podstawie szablonu przygotowanego w ćwiczeniu 3.20 utwórz stronę internetową, na której zaprezentujesz osiągnięcia piłkarzy nożnych.

3.9. Responsywny układ strony

Responsywny układ strony to taki układ, który zmienia swoje atrybuty zdefiniowane za pomocą stylów CSS w zależności od różnych zewnętrznych czynników. Jest wiele czynników, pod których wpływem elementy strony mogą otrzymywać nowe atrybuty. Jednak najczęściej to szerokość okna przeglądarki internetowej wymusza konieczność definiowania dodatkowych reguł CSS. Zadziałają one, gdy dokument HTML zostanie wyświetlony w oknie przeglądarki internetowej o określonej szerokości (komputer, tablet, telefon).

Responsywny projekt strony internetowej pozwala za pomocą kodu HTML i stylów CSS zmienić rozmiar zawartości, przesunąć określoną zawartość lub ukryć ją, tak aby strona dobrze wyglądała na dowolnym urządzeniu.

W HTML5 za pomocą znacznika `<meta>` możliwe jest dołączenie metatagu `viewport` i przejęcie kontroli nad widocznym dla użytkownika obszarem wyświetlania strony internetowej. Do strony internetowej należy dołączyć następujący zapis znacznika `<meta>`:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Spowoduje on poprawne wyświetlanie strony na urządzeniach mobilnych.

Zdefiniowany w znaczniku `<meta>` element `"viewport"` udostępnia przeglądarce internetowej polecenia dotyczące kontrolowania wymiarów strony i skalowania.

Składnia `width=device-width` określa szerokość strony, która powinna odpowiadać szerokości ekranu urządzenia, gdzie będzie wyświetlana strona.

Część `initial-scale=1.0` ustawia początkowy poziom powiększenia w momencie, gdy strona jest otwierana przez przeglądarkę.

W specyfikacji CSS3 rozbudowane zostały *media queries* (zapytania o media), co pozwala na rozpoznawanie rozdzielczości urządzeń, na których wyświetlana jest strona internetowa. W zależności od wykrytych rozdzielczości wczytywane są różne arkusze stylów. Wybrane elementy strony są przenoszone w inne miejsce lub są ukrywane. W wyniku tego strony o tej samej strukturze HTML są wyświetlane w różny sposób.

Dodając regułę `@media`, spowodujemy włączanie określonych atrybutów stylów CSS tylko wtedy, gdy spełnione zostaną wymagane warunki.

Przykład 3.131

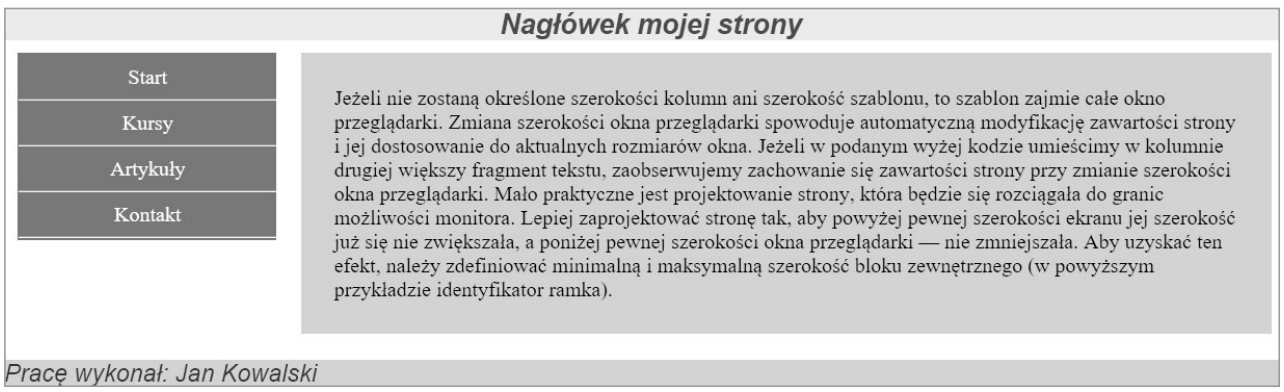
```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <title>Strona responsywna</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <style>
  body {
    background-color: #c0c0c0;
  }
  @media only screen and (max-width: 500px) {
    body {
      background-color: lightblue;
    }
  }
  </style>
</head>
<body>
  <p>Zmiana koloru tła, gdy szerokość strony mniejsza od 500 pikseli.</p>
</body>
</html>
```

W podanym przykładzie za pomocą reguły `@media` zdefiniowany został warunek narzucony na szerokość strony równy `500px`. Efekt zmiany koloru tła strony można zaobserwować po zmniejszeniu rozmiaru okna przeglądarki.

Aby tworzone strony internetowe dobrze wyglądały na małym ekranie, można definiować tzw. punkty przerwania. Po ich zdefiniowaniu wybrane części projektowanej strony będą się zachowywały inaczej po obu stronach punktów przerwania.

Strona, która będzie wyświetlana na urządzeniach mobilnych, powinna być zbudowana na bazie jednej kolumny. Zapewni to poprawne skalowanie strony i dopasowanie szerokości kolumny do ekranu urządzenia wyświetlającego. Natomiast oryginalna strona w najprostszym przypadku może zawierać dwie kolumny, np. kolumnę z menu oraz kolumnę informacyjną (rysunek 3.68). Można się jeszcze zastanowić, które elementy strony mogą zostać ukryte dla urządzeń mobilnych.



Rysunek 3.68. Wygląd oryginalnej strony z dwiema kolumnami

Przykład 3.132

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Strona responsywna</title>
  <style>
  * {
    margin: 0px;
    padding: 0px;
  }
  section {
    overflow: hidden;
  }
  #ramka {
    margin-right: auto;
    margin-left: auto;
    min-width: 300px;
    max-width: 1000px;
  }
  #naglowek {
    background-color: #FCF;
    font-size: 16pt;
    color: #5e1138;
    font-family: Arial;
    font-style: italic;
    text-align: center;
    font-weight: bold;
  }
  #stopka {
    background-color: #c0c0c0;
    font-size: 14pt;
    color: blue;
    font-family: Arial;
```

```
  font-style: Italic;
  clear: both;
}
.kolumny {
  margin-top: 10px;
  margin-right: 2%;
  margin-bottom: 2%;
  margin-left: 10px;
  background-color: #c0c0c0;
}
#kol1 {
  float: left;
  width: 20%;
}
#kol2 {
  padding: 25px;
}
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  width: 100%;
  background-color: #555;
  border: 1px solid #555;
}
li a {
  display: block;
  color: #fff;
  padding: 8px 16px;
  text-decoration: none;
}
li {
  text-align: center;
  border-bottom: 1px solid #f1f1f1;
}
```

```
  li a.active {
    background-color: #4caf50;
    color: white;
  }
  li a:hover:not(.active) {
    background-color: #f1f1f1;
    color: black;
  }
  @media only screen and (max-width:620px) {
    #kol1 {
      width: 100%;
    }
  }
  </style>
</head>
<body>
  <div id="ramka">
    <header id="naglowek">Nagłówek mojej strony</header>
    <section class="kolumny" id="kol1">
      <h3>Menu</h3>
      <ul>
        <li><a href="start.html">Start</a></li>
        <li><a href="nowe.html">Kursy</a></li>
        <li><a href="serwis.html">Artykuły</a></li>
        <li><a href="uslugi.html">Kontakt</a></li>
      </ul>
    </section>
    <section class="kolumny" id="kol2">
      <h3>Opis</h3>
```

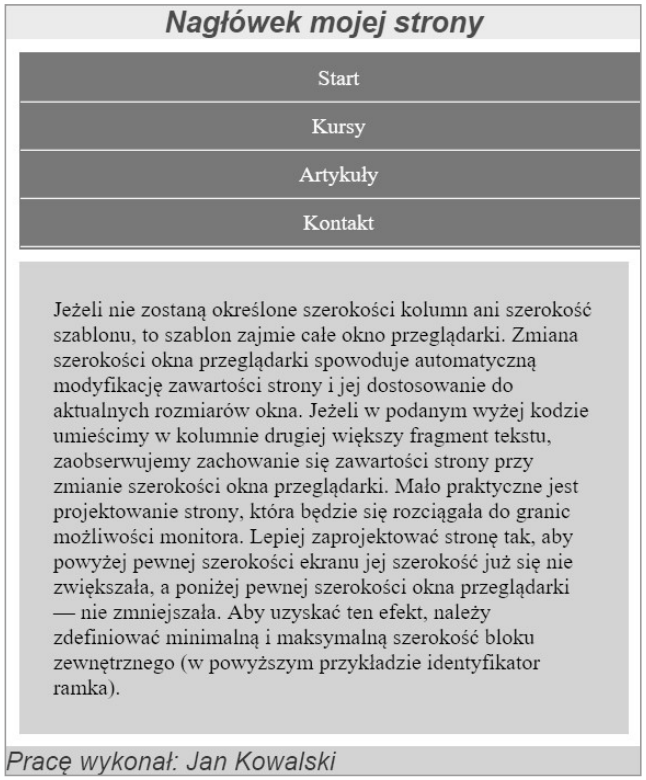
Jeżeli nie zostaną określone szerokości kolumn ani szerokość szablonu, to szablon zajmie całe okno przeglądarki. Zmiana szerokości okna przeglądarki spowoduje automatyczną modyfikację zawartości strony i jej dostosowanie do aktualnych rozmiarów okna.

Jeżeli w podanym wyżej kodzie umieścimy w kolumnie drugiej większy fragment tekstu, zaobserwujemy zachowanie się zawartości strony przy zmianie szerokości okna przeglądarki.

```
      Mało praktyczne jest projektowanie strony, która będzie się rozciągała
do granic możliwości monitora. Lepiej zaprojektować stronę tak, aby powyżej
pewnej szerokości ekranu jej szerokość już się nie zwiększała, a poniżej pewnej
szerokości okna przeglądarki — nie zmniejszała. Aby uzyskać ten efekt, należy
zdefiniować minimalną i maksymalną szerokość bloku zewnętrznego (w powyższym
przykładzie identyfikator ramka).
    </section>
    <footer id="stopka">Pracę wykonał: Jan Kowalski</footer>
  </div>
</body>
</html>
```

W podanym przykładzie, jeżeli szerokość okna, w którym jest wyświetlana zawartość strony, będzie mniejsza od `620px`, nastąpi zmiana w sposobie wyświetlania zawartości strony (rysunek 3.69). Menu, które było wyświetlane z lewej, zostanie wyświetlone nad panelem głównym strony. Wynika to z zastosowania w kodzie strony reguły:

```
@media only screen and (max-width:620px) {
  #kol1 {width:100%;}
}
```



Rysunek 3.69. Układ strony po zmianie szerokości wyświetlanej strony

Liczba reguł `@media` zależy od rodzaju strony internetowej. Przy bardzo rozbudowanych stronach zdefiniowanie 3–5 reguł nie jest rzadkością i mogą one znacząco zmieniać wygląd strony nawet przy niewielkich zmianach rozdzielczości.

Jedną z zasad podczas projektowania responsywnej strony internetowej jest ukrywanie wybranych elementów przy mniejszych rozdzielczościach ekranu. Do ukrywania określonych elementów strony używamy atrybutu `display: none`.

Przykład 3.133

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Strona responsywna</title>
  <style>
  * {
    margin: 0px;
    padding: 0px;
  }
  body {
    font: 16px Verdana;
    line-height: 1.6;
    background: #e0dfe1;
  }
  #blok {
    width: 90%;
    overflow: hidden;
    margin: 0 auto;
  }
  a {
    color: #354458;
  }
  #header {
    height: 100px;
    background: #33415d;
    margin: 0 0 10px 0;
  }
  #header h1 {
```

```
  padding: 20px;
  color: #e9e0d6;
}
#kon {
  width: 75%;
  background: #c0c0c0;
  float: left;
  padding: 1%;
  overflow: hidden;
}
#menu {
  background: #bbc0cb;
  width: 20%;
  float: right;
  padding: 1%;
  overflow: hidden;
}
#menu h4 {
  padding: 3px 3px 15px 3px;
}
#menu ul li {
  list-style-type: none;
  padding: 0 0 0 10px;
}
li a {
  text-decoration: none;
}
@media only screen and (max-width: 680px) {
  #menu {
    width: 88%;
    position: absolute;
    height: 30px;
  }
  #menu h4 {
    display: none;
  }
  #menu ul li {
```

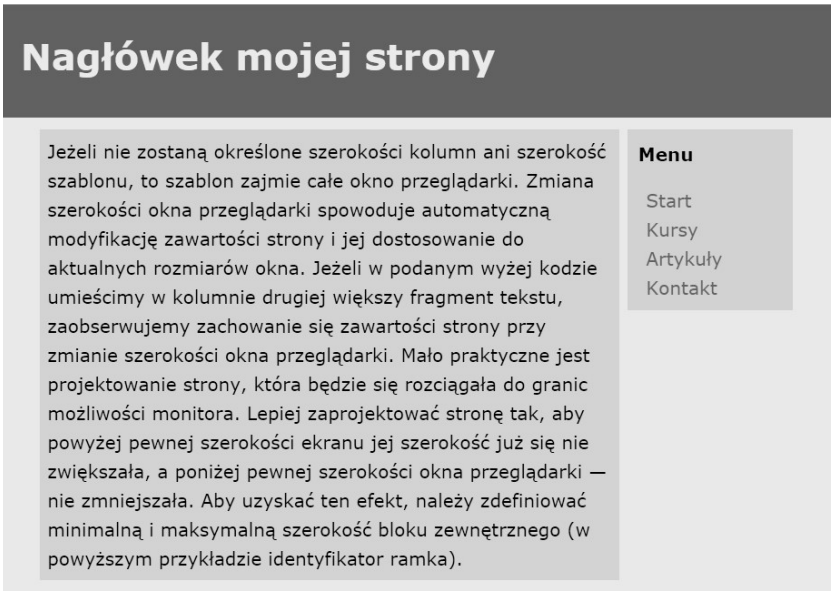
```
      display: inline;
    }
    #kon {
      width: 98%;
      position: relative;
      top: 30px;
      margin-top: 3%;
    }
  }
  </style>
</head>
<body>
  <header id="header">
    <h1>Nagłówek mojej strony</h1>
  </header>
  <div id="blok">
    <section id="kon">
      <h3>Opis</h3>
      <p>
      Jeżeli nie zostaną określone szerokości kolumn ani szerokość szablonu,
to szablon zajmie całe okno przeglądarki. Zmiana szerokości okna przeglądarki
spowoduje automatyczną modyfikację zawartości strony i jej dostosowanie do aktu-
alnych rozmiarów okna.
      Jeżeli w podanym wyżej kodzie umieścimy w kolumnie drugiej większy frag-
ment tekstu, zaobserwujemy zachowanie się zawartości strony przy zmianie szero-
kości okna przeglądarki.
      Mało praktyczne jest projektowanie strony, która będzie się rozciągała
do granic możliwości monitora. Lepiej zaprojektować stronę tak, aby powyżej
pewnej szerokości ekranu jej szerokość już się nie zwiększała, a poniżej pewnej
szerokości okna przeglądarki — nie zmniejszała. Aby uzyskać ten efekt, należy
zdefiniować minimalną i maksymalną szerokość bloku zewnętrznego (w powyższym
przykładzie identyfikator ramka).
      </p>
    </section>
    <section id="menu">
      <h4>Menu</h4>
      <ul>
        <li><a href="start.html">Start</a></li>
```

```
        <li><a href="nowe.html">Kursy</a></li>
        <li><a href="serwis.html">Artykuły</a></li>
        <li><a href="uslugi.html">Kontakt</a></li>
      </ul>
    </section>
  </div>
</body>
</html>
```

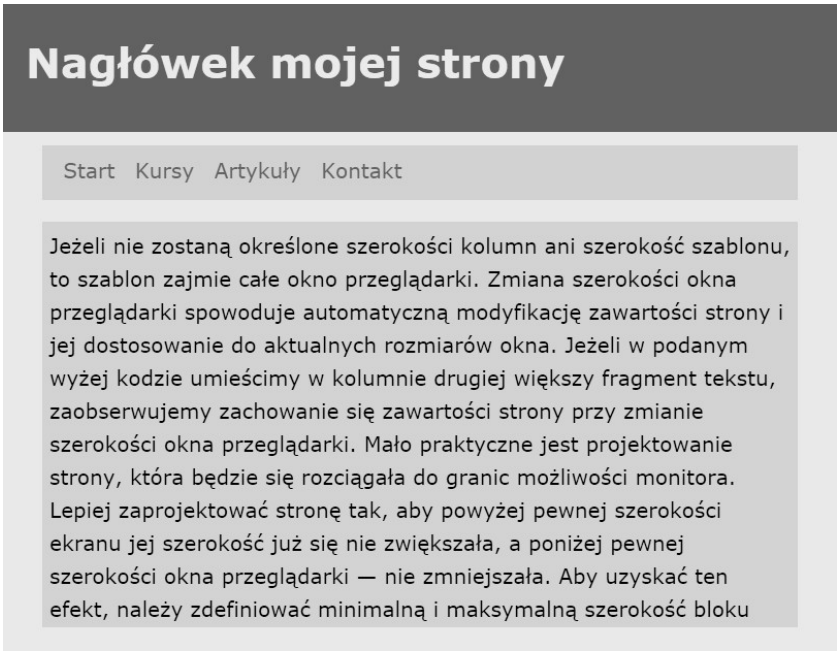
W podanym przykładzie strona składa się z dwóch kolumn. Prawa kolumna zawiera menu (rysunek 3.70). Gdy szerokość okna zmniejszy się do `680px`, menu przesunie się w górę i zmieni w menu poziome (rysunek 3.71).

```
@media only screen and (max-width: 680px) {
   #menu {width:90%; position: absolute; height: 30px;}
   #menu h4 {display: none;}
   #menu ul li {display: inline;}
   #kon {width: 98%; position: relative; top: 30px; margin-top: 3%;}
}
```

Aby menu zostało przeniesione do góry, nadano mu atrybut `position: absolute`. Z kolei aby menu zmieniło się na poziome, ustawiono atrybut `width: 90%`. Zwiększono również szerokość panelu głównego (`width: 98%`). Żeby kolejne elementy menu układały się poziomo, zmieniono atrybut `display: inline`. Dodatkowo przy zmianie menu na poziome ukryty został jego tytuł (tekst *Menu*) — atrybut `display: none`.



Rysunek 3.70. Wygląd strony z dwiema kolumnami



Rysunek 3.71. Nowy układ menu po zmianie szerokości wyświetlanej strony

Obrazki

Jeżeli na responsywnej stronie internetowej występują obrazki (rysunek 3.72), powinny być automatycznie skalowane wraz z szerokością strony. W tym celu można do arkusza stylów dodać atrybuty, które umożliwią ich skalowanie (rysunek 3.73).

Rysunek 3.72. Strona responsywna z umieszczonym obrazkiem

Przykład 3.134

```
img {
  max-width: 100%;
  height: auto;
  width: auto;
}
```

Rysunek 3.73. Przeskalowanie obrazka na stronie responsywnej

Zawsze, gdy tworzona jest responsywna strona internetowa, szerokość obrazków i innych grafik powinna być podawana w wartościach procentowych. Gdy nie jest to możliwe, należy taki element umieścić w kontenerze i jemu nadać wartości procentowe.

W przypadku gdy na stronie pojawiają się złożone elementy graficzne, które w wyniku skalowania stają się niewyraźne, jedynym rozwiązaniem jest ich ukrywanie i zastępowanie mniejszymi odpowiednikami.

Element <picture>

Zastosowanie znacznika `<picture>` do wyświetlania obrazków pozwala na większą elastyczność i wybór z dostępnych obrazków tego, który najlepiej zaprezentuje się na stronie. Zamiast jednego obrazka, który jest skalowany w zależności od sposobu jego wyświetlania na stronie, można użyć wielu obrazków. Ten, który najlepiej odpowiada parametrom ekranu, zostanie wyświetlony.

W elemencie <picture> występują dwa rodzaje znaczników: <source> i <img>.

Znacznik <source> posiada następujące atrybuty:

- srcset — określa URL obrazka do wyświetlania (wymagany);
- media — definiuje zapytanie o media, które są definiowane w regule CSS @media;
- sizes — określa szerokość (width), występuje w zapytaniu o media i określa ograniczenia narzucone na szerokość ekranu;
- type — określa typ danych.

Przykład 3.135

```
<picture>
  <source srcset="obrazek1.jpg" media="(max-width: 600px)">
  <source srcset="obrazek2.jpg">
  <img src="obrazek2.jpg" alt="obrazek">
</picture>
```

Przeglądarka internetowa wybierze pierwszy obrazek z listy elementów (<source>), który pasuje do zdefiniowanego zapytania o media, i go wyświetli.

Znacznik <img> jest wymagany jako ostatni element w bloku deklaracji <picture>. Służy również do osiągnięcia kompatybilności wstecznej i pozwala na wyświetlenie obrazka w przeglądarkach, które nie obsługują elementu <picture>.

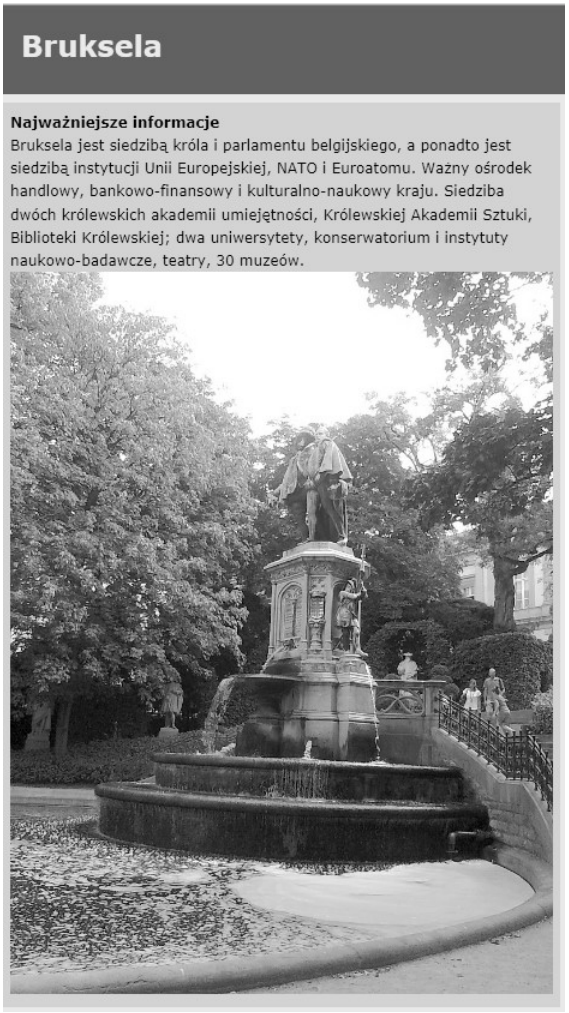
Przykład 3.136

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Obrazek responsywny</title>
  <style>
  * {
    margin: 0px;
    padding: 0px;
  }
  body {
    font: 16px Verdana;
    line-height: 1.6;
    background: #e0dfe1;
```

```
  }
  #header {
    height: 100px;
    background: #33415d;
    margin: 0 0 10px 0;
  }
  #header h1 {
    padding: 20px;
    color: #e9e0d6;
  }
  #kon {
    width: 75%;
    background: #c0c0c0;
    padding: 1%;
  }
  img {
    max-width: 100%;
    height: auto;
  }
  </style>
</head>
<body>
<header id="header">
  <h1>Bruksela</h1>
</header>
<section id="kon">
  <h4>Najważniejsze informacje</h4>
  <p>
  Bruksela jest siedzibą króla i parlamentu belgijskiego, a ponadto jest siedzibą instytucji Unii Europejskiej, NATO i Euroatomu. Ważny ośrodek handlowy, bankowo-finansowy i kulturalno-naukowy kraju. Siedziba dwóch królewskich akademii umiejętności, Królewskiej Akademii Sztuki, Biblioteki Królewskiej; dwa uniwersytety, konserwatorium i instytuty naukowo-badawcze, teatry, 30 muzeów.
  </p>
  <picture>
```

```
    <source media="(min-width: 800px)" srcset="pom1.jpg">
    <source media="(min-width: 600px)" srcset="pom2.jpg">
    <img src="pom3.jpg" alt="zdjęcie z Brukseli">
  </picture>
</section>
</body>
</html>
```

Wynik działania kodu został przedstawiony na rysunkach. Jeżeli szerokość ekranu jest większa niż `800px`, zostanie wyświetlony pierwszy obrazek (rysunek 3.74). Kolejny obrazek zawiera wykadrowaną część pierwszego zdjęcia i zostanie wyświetlony, gdy ekran będzie miał szerokość mniejszą od `800px`. Gdy ekran będzie miał szerokość mniejszą od `600px`, zostanie wyświetlony jeszcze mniejszy wykadrowany fragment zdjęcia (rysunek 3.75).



Rysunek 3.74.
Rysunek dodany do strony internetowej

Rysunek 3.75.
Zmiana rysunku przy zmienionej szerokości ekranu

Tabele

Dla tabel wyświetlanych na responsywnej stronie internetowej można zdefiniować pasek przewijania, który zostanie wyświetlony, gdy szerokość ekranu będzie zbyt mała, aby wyświetlić całą zawartość tabeli. Żeby uzyskać podany efekt, tabelę należy umieścić w kontenerze (np. blok `<div>`) i nadać mu atrybut `overflow-x: auto`.

Przykład 3.137

```
<!DOCTYPE html>
<html lang="pl-PL">
<head>
  <meta charset="UTF-8">
  <title>Tabela responsywna</title>
  <style>
  table {
    width: 100%;
    border-collapse: collapse;
  }
  th, td {
    text-align: left;
    padding: 8px;
  }
  div {
    overflow-x: auto;
  }
  tr:nth-child(even) {
    background-color: #f2f2f2
  }
  </style>
</head>
<body>
  <div>
    <table>
      <tr>
        <th>Europa</th>
        <th>Azja</th>
        <th>Afryka</th>
```

```
        <th>Ameryka Północna</th>
        <th>Ameryka Południowa</th>
      </tr>
      <tr>
        <td>Francja</td>
        <td>Indie</td>
        <td>Egipt</td>
        <td>Meksyk</td>
        <td>Peru</td>
      </tr>
      <tr>
        <td>Hiszpania</td>
        <td>Chiny</td>
        <td>Nigeria</td>
        <td>Kanada</td>
        <td>Brazylia</td>
      </tr>
      <tr>
        <td>Włochy</td>
        <td>Japonia</td>
        <td>Kongo</td>
        <td>USA</td>
        <td>Argentyna</td>
      </tr>
    </table>
  </div>
</body>
</html>
```

Wynik wykonania kodu został pokazany na rysunku 3.76

Europa	Azja	Afryka	Ameryka Północna	Ameryka Południowa
Francja	Indie	Egipt	Meksyk	Peru
Hiszpania	Chiny	Nigeria	Kanada	Brazylia
Włochy	Japonia	Kongo	USA	Argentyna

Rysunek 3.76.
Tabela responsywna

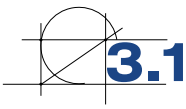
Po zmianie rozmiaru okna na dole bloku pojawił się pasek przewijania. Należy zwrócić uwagę na to, że również teksty umieszczone w nagłówkach kolumn zostały „złamane", a sposób ich wyświetlania został dostosowany do szerokości okna.

W podanym przykładzie zastosowano selektor `:nth-child` w postaci:

```
tr:nth-child(even) {background-color: #f2f2f2;}
```

Posłużył on do pokolorowania naprzemiennie parzystych wierszy tabeli. Atrybut `even` określa wiersze parzyste. Dla wierszy nieparzystych powinien zostać zastosowany atrybut `odd`.

```
tr:nth-child(odd) {background-color: #c0c0c0;}
```



3.10. Pytania i zadania

3.10.1. Pytania

1. Jaką rolę w tworzeniu strony internetowej odgrywają arkusze stylów CSS?
2. W jaki sposób i gdzie definiowany jest styl lokalny?
3. Do czego służy znacznik `<span>`?
4. Podaj różnice w stosowaniu zewnętrznego i wewnętrznego arkusza stylów CSS.
5. Podaj ogólną postać selektora klasy.
6. Kiedy stosujemy selektor identyfikatora?
7. Co oznacza zapis `a:link {color: green; background: yellow;}` występujący w arkuszu stylów?
8. Jaką rolę w modelu blokowym CSS odgrywają `Border` i `Padding`?
9. Na czym polega pozycjonowanie elementów na stronie internetowej?
10. Jaką rolę podczas definiowania stylów odgrywa selektor uniwersalny `*`?

Pytania egzaminacyjne

Zadanie 1.

W języku CSS wcięcie pierwszej linii akapitu na 30 pikseli uzyska się za pomocą zapisu:

A. `p {text-indent: 30px;}.`
B. `p {text-spacing: 30px;}.`
C. `p {line-height: 30px;}.`
D. `p {line-indent: 30px;}.`

Zadanie 2.

W języku CSS zdefiniowano następujące formatowanie:

```
h1 i {color: red;}
```

Kolorem czerwonym zostanie zapisany:

A. tylko tekst pochylony nagłówka pierwszego stopnia.
B. tylko tekst pochylony we wszystkich poziomach nagłówków.
C. cały tekst nagłówka pierwszego stopnia oraz pochylony tekst akapitu.
D. cały tekst nagłówka pierwszego stopnia oraz cały tekst pochylony, niezależnie od tego, w którym miejscu strony się znajduje.

Zadanie 3.

W języku CSS, aby sformatować dowolny element języka HTML w ten sposób, że po najechaniu na niego kursorem zmienia on kolor czcionki, należy zastosować pseudoklasę:

A. `:active`.
B. `:hover`.
C. `:visited`.
D. `:coursor`.

Zadanie 4.

Który zapis stylu CSS ustawi tło bloku na kolor niebieski?

A. `div {background-color: blue;}`.
B. `div {border-color: blue.}`.
C. `div {shadow: bIue;}`.
D. `div {color: bIue;}`.

Zadanie 5.

W zamieszczonym przykładzie pseudoklasa `hover` sprawi, że styl pogrubiony będzie przypisany:

```
a:hover {font-weight: bold;}
```

A. wszystkim odnośnikom odwiedzonym.
B. wszystkim odnośnikom nieodwiedzonym.
C. każdemu odnośnikowi niezależnie od aktualnego stanu.
D. odnośnikowi, w momencie kiedy najechał na niego kursor myszy.

Zadanie 6.

W celu określenia wysokości obrazka wyświetlonego na stronie WWW należy wykorzystać właściwość CSS o nazwie:

A. `padding`.
B. `margin`.
C. `height`.
D. `width`.

Zadanie 7.

W podanej regule CSS `h1 {color: blue;}` oznacza:

A. klasę.
B. wartość.
C. selektor.
D. deklarację.

Zadanie 8.

W języku CSS zapis w postaci:

```
p {background-image: url("rysunek.png");}
```

sprawi, że *rysunek.png* będzie:

A. tłem całej strony.
B. tłem każdego paragrafu.
C. wyświetlony obok każdego paragrafu.
D. wyświetlony, jeżeli w kodzie zostanie zastosowany znacznik `img`.

Zadanie 9.

W języku CSS zdefiniowano formatowanie:

```
p > i {color: blue;}
```

Oznacza to, że kolorem niebieskim zostanie zapisany:

A. cały tekst paragrafu niezależnie od jego formatowania.
B. pochylony tekst paragrafu.
C. cały tekst nagłówków niezależnie od ich formatowania.
D. pogrubiony tekst paragrafu.

Zadanie 10.

Chcąc zdefiniować formatowanie tabeli w języku CSS w taki sposób, aby wiersz, który jest aktualnie wskazywany kursorem myszy, został wyróżniony np. innym kolorem, należy zastosować:

A. pseudoklasę `:visited`.
B. pseudoklasę `:hover`.
C. pseudoelement `:first-line`.
D. nowy selektor klasy dla wierszy tabeli.

3.10.2. Zadania

Zadanie 3.1

Dla witryny biura podróży GLOB zdefiniuj style wyświetlanych w niej informacji. Witryna powinna być wyświetlana zgodnie ze schematem podanym na rysunku 2.44. Podczas tworzenia witryny i definiowania dla niej arkuszy stylów uwzględnij to, że może być ona wyświetlana na różnych urządzeniach (komputerze, tablecie, telefonie komórkowym).

Zadanie 3.2

Dla utworzonej witryny internetowej na temat konkursów przedmiotowych odbywających się w szkole należy ustalić styl wyświetlanych na stronach informacji. Uczestnicy projektu powinni określić wygląd tytułów stron, tekstów zawierających informacje, różnego rodzaju list, zachowanie się odsyłaczy do innych stron internetowych oraz sposób rozmieszczenia zdjęć na stronie. Po tych ustaleniach należy utworzyć zewnętrzny arkusz stylów zawierający definicję stylów dla wszystkich elementów witryny internetowej. Przygotowany arkusz stylów powinien zostać dołączony do wszystkich dokumentów HTML utworzonych przez uczestników projektu.

Zadanie egzaminacyjne — cz. II

Styl CSS witryny internetowej

Styl elementów witryny zdefiniuj przy pomocy języka CSS, w osobnym pliku o nazwie *styl.css*, plik ten zapisz w podfolderze WWW oraz prawidłowo dołącz do pliku z kodem strony.

Wymagania odnośnie do stylu CSS:

- kolor czcionki odnośników: `#ffd87e`,
- kolor tła banera, stopki oraz panelu lewego: `#af8c4b`,
- kolor tła panelu prawego: `#ffd87e`,
- wyrównanie tekstu banera i stopki: do środka,
- wyrównanie tekstu panelu lewego: do prawej,
- krój czcionki dla całej strony: Tahoma,
- szerokość panelu lewego: 30%,
- szerokość panelu prawego: 70%,
- wysokość paneli lewego i prawego: 500 px,
- wysokość banera: 60 px,
- wysokość stopki: 25 px,
- kolor poziomej linii w lewym panelu: `#ffd87e`,
- punktor list w panelu lewym: okrąg,
- wszystkie komórki tabeli obramowane ramką czarną kropkowaną o szerokości 1 px,
- włączone paski przewijania dla panelu prawego.