WILEY

Wydanie V

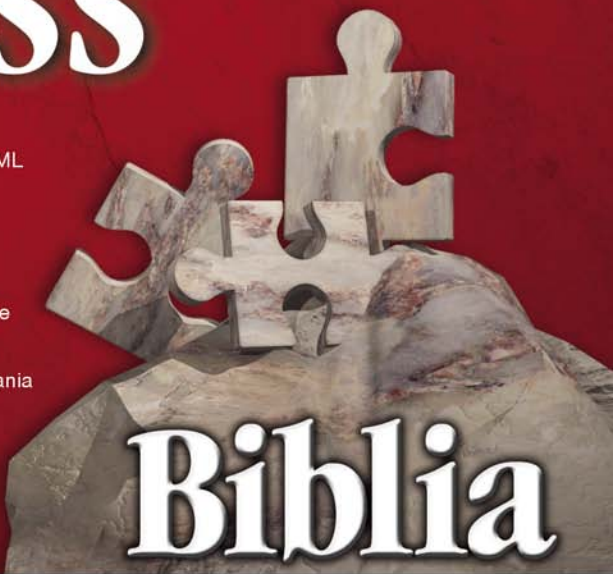
Steven M. Schafer

HTML, XHTML i CSS

Opanuj
możliwości języka HTML
i kaskadowych
arkuszy stylów

Twórz
strony statyczne,
dynamiczne
i na urządzenia mobilne

Odkryj
sposoby rozbudowywania
stron WWW



Helion

Biblia

Wiedza obiecana

HTML, XHTML i CSS. Biblia. Wydanie V

Autor: Steven M. Schafer
Tłumaczenie: Piotr Rajca
ISBN: 978-83-246-2742-4
Tytuł oryginału: HTML, XHTML, and CSS Bible
Format: 172×245, stron: 768

- Opanuj możliwości języka HTML i kaskadowych arkuszy stylów
- Twórz strony statyczne, dynamiczne i na urządzenia mobilne
- Odkryj sposoby rozbudowania stron WWW

Jeśli czytasz notkę tej książki, zapewne zdecydowałeś się na stworzenie własnej, profesjonalnej strony internetowej. Jedyne, czego Ci teraz trzeba, to wiedza, jak sprawnie wykorzystać niezbędny w tym celu zestaw technologii – języki HTML i XHTML oraz kaskadowe arkusze stylów CSS. To właśnie te narzędzia pozwalają zbudować dokument witryny, sformatować umieszczone na niej teksty, osadzać elementy multimedialne i nadawać jej atrakcyjny wizualnie wygląd oraz nowoczesny, dynamiczny charakter. A jeśli masz jeszcze wątpliwości, czy zadanie to leży w zasięgu Twoich umiejętności, ta książka z pewnością je rozwieje!

Podręcznik ten, adresowany zarówno do początkujących webmasterów, jak i zawodowców, dla pierwszych stanowi solidną podstawę do rozpoczęcia przygody z tworzeniem stron WWW, natomiast dla drugich wyczerpujące kompendium wiedzy o odpowiednich technologiach. Autor tak uporządkował zagadnienia – od prostych po złożone – aby w jak największym stopniu ułatwić ich opanowanie przez czytelnika. Naukę rozpoczniesz zatem od poznania znaczników HTML, struktury i atrybutów tego języka. Dowiesz się, jak używać multimediów i skryptów, oraz skorzystasz z programów wspomagających tworzenie czy testowanie dokumentów. Później przeczytasz o praktycznych rozszerzeniach HTML – XML i HTML Basic – oraz zgłębisz wszystkie kwestie związane ze stosowaniem CSS i publikacją strony. Ogarniesz w ten sposób wszystkie aspekty samodzielnego budowania nowoczesnej, zgodnej ze standardami, estetycznej witryny internetowej!

- Budowanie treści z wykorzystaniem języka HTML
- Tworzenie list, łączy, tabel i ramek
- Osadzanie elementów multimedialnych
- Skrypty serwerowe i skrypty klienckie
- Czym jest i czemu służy DHTML?
- Projektowanie serwisów wielojęzycznych i zasady lokalizacji serwisów
- Publikacja witryny w sieci
- Porządkowanie i walidacja dokumentów
- Zastosowanie języka XML
- Tworzenia stron dla urządzeń mobilnych
- Kontrola prezentacji za pomocą języka CSS
- Praktyczne sztuczki i triki w CSS

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

Spis treści

Rozdział 36.
Dynamiczny HTML i CSS

W tym rozdziale:

- Korzystanie z właściwości CSS w kodzie JavaScript
- Użyteczne operacje z użyciem CSS

Kaskadowe arkusze stylów mogą być potężnym narzędziem umożliwiającym tworzenie ładnie sformatowanych stron WWW. W tym rozdziale opisano, w jaki sposób w różnych przeglądarkach można manipulować wartościami właściwości CSS, by nadać dokumentom HTML dynamiczny charakter. Dowiesz się z niego, w jaki sposób można uzyskiwać dostęp do właściwości CSS oraz jak operować nimi w skryptach, aby wykonywać takie operacje jak zmiana koloru tekstu. Jak się okaże, można zmienić wartość każdej właściwości CSS.

Jak się przekonasz, niektóre przeglądarki (mamy tu na myśli głównie program Internet Explorer) udostępniają możliwość tworzenia dynamicznych efektów wizualnych, takich jak cienie i rozmycia, z wykorzystaniem składni zbliżonej do CSS.

Korzystanie z właściwości CSS w kodzie JavaScript

Przeglądarki fundacji Mozilla (Firefox) oraz Internet Explorer zapewniają możliwość dostępu do właściwości CSS poziomu 1. w kodzie JavaScript, za pośrednictwem obiektowego modelu dokumentu (DOM). Niestety, pomiędzy DOM stosowanym w przeglądarkach Firefox oraz Internet Explorer występują pewne różnice. W obu tych rozwiązaniach zostały częściowo zaimplementowane możliwości standardu CSS2, jednak nie są to te same możliwości, przez co wykorzystujący je skrypt, który działa w jednej z tych przeglądarek, może nie działać w drugiej. Należy zwrócić uwagę, że silnik Gecko (używany do przetwarzania i wyświetlania stron WWW w przeglądarkach Mozilli) obsługuje wszystkie właściwości standardu CSS poziomu 2.

Ogólnie rzecz biorąc, z właściwości CSS korzysta się w standardowy sposób — ich wartości są odczytywane w sposób typowy dla właściwości i ustawiane przy użyciu metod. Aby odwołać się do właściwości CSS w kodzie JavaScript, należy po prostu podać jej nazwę, o ile tylko nie zawiera ona łącznika. W przypadku nazw kilkuczłonowych znak

łącznika należy usunąć, a znak znajdujący się bezpośrednio za nim należy zapisać wielką literą. Wszystkie pozostałe znaki nazwy właściwości są zapisywane małymi literami. Na przykład właściwość:

```
font-size
```

staje się w kodzie JavaScript:

```
fontSize
```

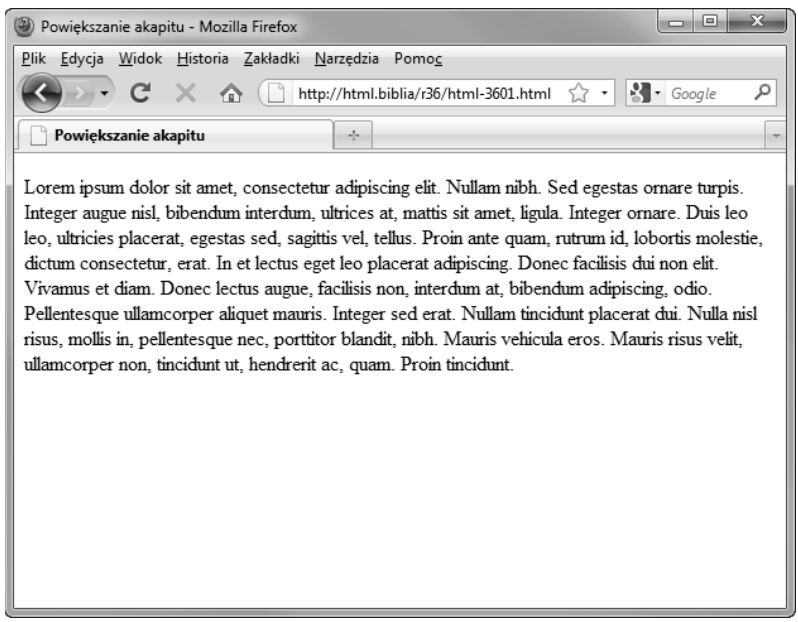
Tak uzyskaną nazwę właściwości dodaje się do nazwy (lub identyfikatora) obiektu zawierającego kolekcję stylów. Na przykład, aby odwołać się do właściwości `font-size` obiektu o nazwie `bigText`, należy użyć następującego wyrażenia:

```
bigText.style.fontSize
```

Powyższe wyrażenie może zostać użyte także do określenia nowej wartości właściwości. Na przykład poniższa instrukcja przypisuje właściwości `font-size` obiektu `bigText` wartość `xx-large`:

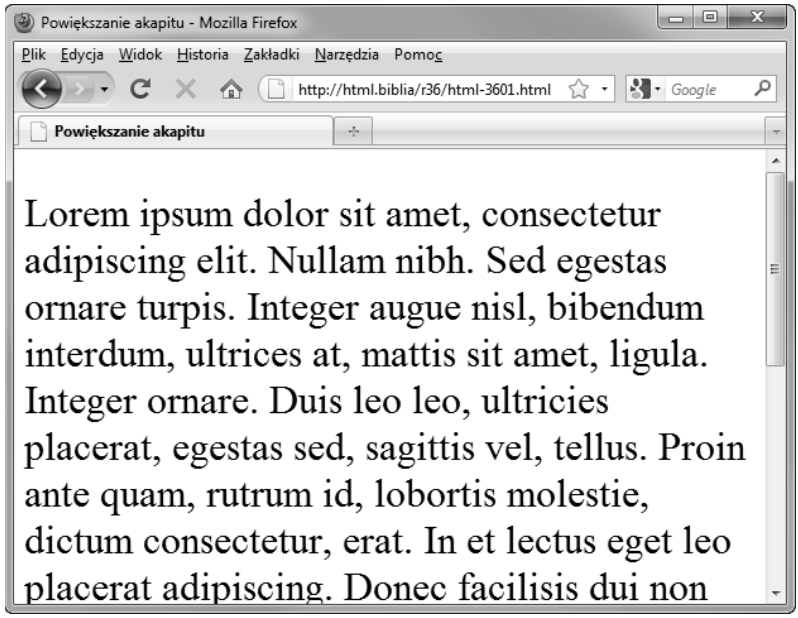
```
bigText.style.fontSize = "xx-large";
```

Przeanalizuj kod poniższego przykładu. Kiedy klikniemy wyświetlony na stronie akapit tekstu, procedura obsługi zdarzeń `onClick` wywoła funkcję `SuperSizeMe()`, która z kolei przypisuje właściwości `font-size` akapitu wartość `xx-large` (przez co tekst akapitu zostaje powiększony). Początkowy wygląd strony został przedstawiony na rysunku 36.1, natomiast rysunek 36.2 przedstawia tę samą stronę po kliknięciu akapitu.



Rysunek 36.1. *Początkowo, przed kliknięciem, tekst jest wyświetlony czcionką o średniej wielkości (medium)*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
```



Rysunek 36.2. *Po kliknięciu wielkość czcionki zostaje zmieniona na bardzo dużą (xx-large)*

```
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Powiększanie akapitu</title>
  <style type="text/css">
    #bigText { font-size: medium; }
  </style>

  <script type="text/javascript">

    function SuperSizeMe(obj) {
      obj.style.fontSize = "xx-large";
    }
  </script>
</head>

<body>

  <p id="bigText" onClick="SuperSizeMe(this);">Lorem ipsum dolor sit amet, consectetur
  adipiscing elit. Nullam nibh. Sed egestas ornare turpis. Integer augue nisl,
  bibendum interdum, ultrices at, mattis sit amet, ligula. Integer ornare. Duis leo
  leo, ultricies placerat, egestas sed, sagittis vel, tellus. Proin ante quam,
  rutrum id, lobortis molestie, dictum consectetur, erat. In et lectus eget leo
  placerat adipiscing. Donec facilisis dui non elit. Vivamus et diam. Donec lectus
  augue, facilisis non, interdum at, bibendum adipiscing, odio. Pellentesque
  ullamcorper aliquet mauris. Integer sed erat. Nullam tincidunt placerat dui. Nulla
  nisl risus, mollis in, pellentesque nec, porttitor blandit, nibh. Mauris vehicula
  eros. Mauris risus velit, ullamcorper non, tincidunt ut, hendrerit ac, quam. Proin
  tincidunt.</p>
</body>
</html>
```

A teraz musimy poświęcić trochę czasu na to, by dobrze przeanalizować i zrozumieć, jak działa powyższy przykład. Można sądzić, że kolekcja `style` zapewnia dostęp do stylów

przypisanych danemu elementowi, niezależnie od tego, skąd one pochodzą. Okazuje się jednak, że tak nie jest. Kolekcja `style` zapewnia możliwość odwoływania się i operowania na atrybucie `style` umieszczonym w znaczniku obiektu. Oznacza to, że wykonanie poniższego fragmentu kodu JavaScript bezpośrednio po wyświetleniu naszej przykładowej strony w przeglądarce spowodowałoby wyświetlenie okienka informacyjnego z napisem „null":

```
alert(document.getElementById("bigText").style.fontSize);
```

Właściwość `style.fontSize` ma wartość `null`, ponieważ w elemencie `bigText` nie został określony atrybut `style`. W jaki sposób zatem działa powyższy przykład, skoro nie zmienia on wartości `medium` atrybutu `font-size` określonego w arkuszu stylów umieszczonym w elemencie `<style>` w sekcji nagłówka strony? Odpowiedź jest prosta — wcale nie musi on zmieniać wartości właściwości określonych w elemencie `<style>`, zmienia on wartości przechowywane w *atrybucie* `style` elementu. Wartości te mają bowiem wyższy priorytet niż właściwości zdefiniowane w arkuszach stylów w sekcji nagłówka strony.

Oczywiście, gdyby wartości przechowywane w atrybucie `style` zostały podane, to można by odczytać ich wartości przy użyciu atrybutu `style`.

Aby odczytać właściwości podane w sekcji `<style>`, należy użyć jednej z dwóch metod: jedna z nich działa w przeglądarkach Internet Explorer, a druga w przeglądarkach Firefox. W pierwszym przypadku korzystamy z udostępnianej przez Internet Explorera właściwości `currentStyle`; z kolei w przeglądarce Firefox obiekt `window` udostępnia metodę `getComputedStyle`.

Korzystanie z właściwości `currentStyle` w Internet Explorerze jest wyjątkowo proste — wystarczy odszukać interesujący element strony, używając w tym celu jego identyfikatora, a następnie skorzystać z właściwości, by pobrać wartość odpowiedniego stylu. Pokazano to na poniższym przykładzie:

```
obj = document.getElementById(id);
value = obj.currentStyle['fontSize'];
```

Należy zwrócić uwagę, że nazwa stylu jest zapisana bez łącznika, zatem ma ona postać `fontSize`, a nie `font-size`.

W przeglądarce Firefox należy wykonać dodatkowy, pośredni krok, gdyż wywołanie metody `getComputedStyle()` zwraca kolekcję, z której dopiero można odczytać interesującą nas wartość stylu. Ten etap pośredni wymaga użycia metody `getPropertyValue()`. Cały proces wygląda następująco:

```
obj = document.getElementById(id);
objstyles = window.getComputedStyle(obj,null);
value = objstyles.getProperty('font-size');
```

Warto zauważyć, że w tym rozwiązaniu stosowane są standardowe nazwy właściwości CSS, a nie ich przekształcone wersje bez łączników. Niemniej jednak w obu przedstawionych przypadkach efekt będzie taki sam — w zmiennej `value` zostanie zapisana wartość właściwości `font-size`.

Dzięki wykorzystaniu niezbyt skomplikowanego kodu określającego rodzaj używanej przeglądarki istnieje możliwość połączenia obu przedstawionych wcześniej rozwiązań

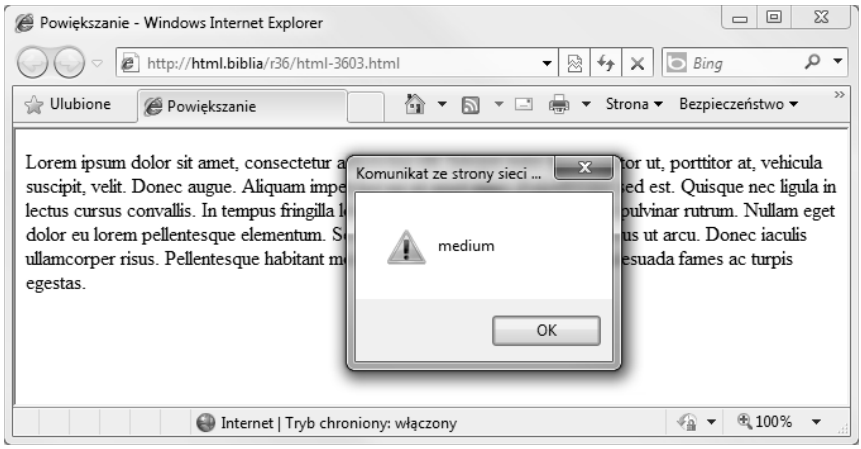
i zaimplementowania ich w postaci jednej funkcji. Poniższy listing przedstawia kod funkcji, która zwraca wartość stylu na podstawie przekazanej w wywołaniu nazwy elementu oraz nazwy właściwości CSS (przy czym używana jest tu prawidłowa nazwa właściwości, a nie nazwa zapisywana bez łącznika):

```
// Funkcja zwraca wartość właściwości CSS o nazwie propName
// odczytanej z elementu określonego przy użyciu identyfikatora id

function getStyleVal (id, propName) {
  // Czy w ogóle możemy cokolwiek zrobić [czy uda się nam
  // pobrać obiekt elementu przy użyciu metody getElementById()]?
  if (obj = document.getElementById(id)) {
    // Czy dostępna jest właściwość currentStyle (IE)?
    if (obj.currentStyle) {
   // Konwertujemy nazwę właściwości na format używany w IE
      if (propName.indexOf("-") != -1) {
        hyp = propName.indexOf("-");
        propName = propName.substr(0,hyp) +
                   propName.charAt(hyp+1).toUpperCase() +
                   propName.substr(hyp+2);
      }
      return obj.currentStyle[propName];
    }

  // Czy jest dostępna metoda getComputedStyle (Mozilla)?
    if (window.getComputedStyle) {
      compStyle = window.getComputedStyle(obj,null);
      return compStyle.getPropertyValue(propName);
    }
  } // Koniec instrukcji if (obj == document.getElementById(id))

 // Nie udało się odszukać elementu — zwracamy pusty łańcuch znaków
  return "";
}
```

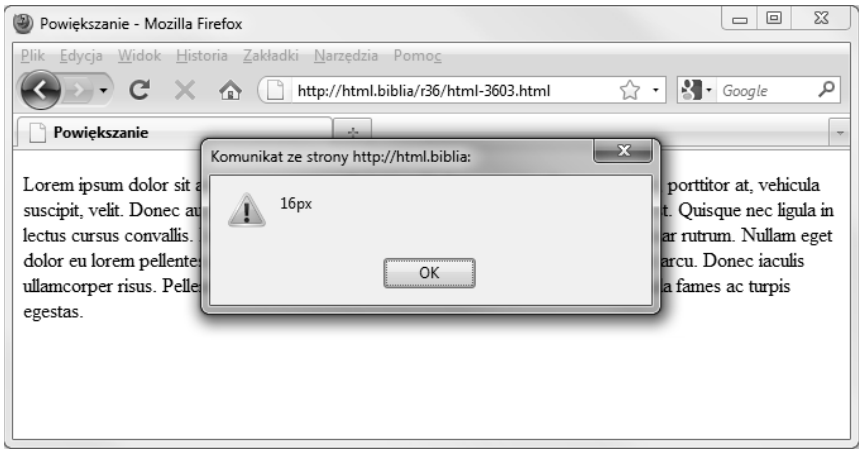
Zwróć uwagę, że w pierwszej kolejności funkcja sprawdza, czy używana przeglądarka udostępnia metodę `document.getElementById()` (wszystkie nowoczesne przeglądarki ją udostępniają). Jeśli metoda ta nie jest dostępna, funkcja kończy działanie, zwracając pusty łańcuch znaków. Następnie funkcja sprawdza, z jakiego sposobu odczytu wartości stylu może skorzystać — tego stosowanego w przeglądarkach Internet Explorer czy tego z przeglądarek Mozilli. Korzystając z jednego z tych dwóch sposobów, funkcja odczytuje wartość podanej właściwości stylu i zwraca ją. Dodatkowo, w razie potrzeby, funkcja konwertuje nazwę właściwości na format stosowany w przeglądarkach IE. Praktyczne wyniki wykorzystania tej funkcji zostały przedstawione na rysunkach: 36.3 (w Internet Explorerze) oraz 36.4 (w przeglądarce Firefox). W obu przypadkach użyto tej samej strony, zawierającej następujący fragment kodu:

```
  <style type="text/css">
    #bigText { font-size: medium; }
  </style>
...
<p id="bigText" onClick="alert(getStyleVal('bigText','font-size'));">Lorem
ipsum dolor sit amet, consecterur adipisicing elit, ...
```

Po kliknięciu tekstu wyświetlonego na stronie zostanie wyświetlone informacyjne okienko dialogowe, a w nim informacje o wartości właściwości `font-size` we wskazanym akapicie.



Rysunek 36.3. *Funkcja getStyleVal() została tu zastosowana do wyświetlenia wartości właściwości font-size w przeglądarce Internet Explorer*



Rysunek 36.4. *Funkcja getStyleVal() została tu zastosowana do wyświetlenia wartości właściwości font-size w przeglądarce Firefox*

Warto zwrócić uwagę, że w przypadku wykonywania funkcji `getStyleVal()` w przeglądarce Firefox zwraca ona wartość bezwzględną wyrażoną w pikselach. Ze względu na sposób działania tej funkcji może się okazać, że zwracane przez nią wartości będą zapisane w innym formacie niż ten, jakiego użyto do określania danej właściwości stylu. Na przykład, mimo że kolor jakiegoś elementu określisz w stylu przy użyciu wartości szesnastkowej `#FFA500`, funkcja zwróci nazwę koloru — `orange`. Albo, jak w zaprezentowanym wcześniej przykładzie, zamiast wielkości względnych (`medium`) — bezwzględne wartości wyrażone w pikselach.

A zatem dlaczego nie można by używać dwóch przedstawionych wcześniej rozwiązań do manipulowania stylami? Wytłumaczenie jest proste — obie pozwalają jedynie na odczyt wartości stylów. Wartości stylów można podawać bezpośrednio, w sposób przedstawiony na początku tego podrozdziału.

Użyteczne operacje z użyciem CSS

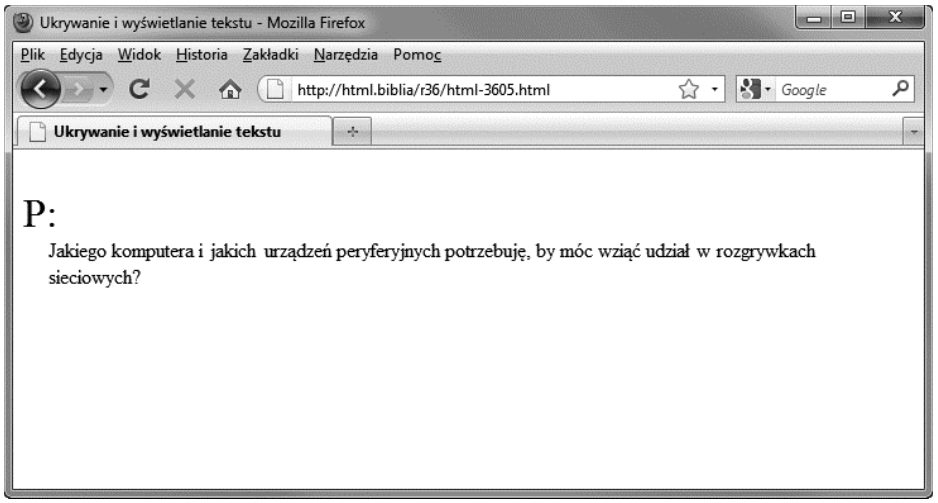
Przykłady przedstawione we wcześniejszej części rozdziału pokazały, w jaki sposób można operować na właściwościach związanych z postacią czcionki. Choć są to całkiem przydatne operacje, to jednak bardziej złożone manipulacje właściwościami CSS z poziomu kodu JavaScript mogą pozwolić nam tworzyć jeszcze bardziej atrakcyjne efekty wizualne. W tej części rozdziału znajdziesz kilka przykładów, które możesz wykorzystać jako punkt startowy do dalszych, własnych eksperymentów.

Ukrywanie i wyświetlanie tekstu

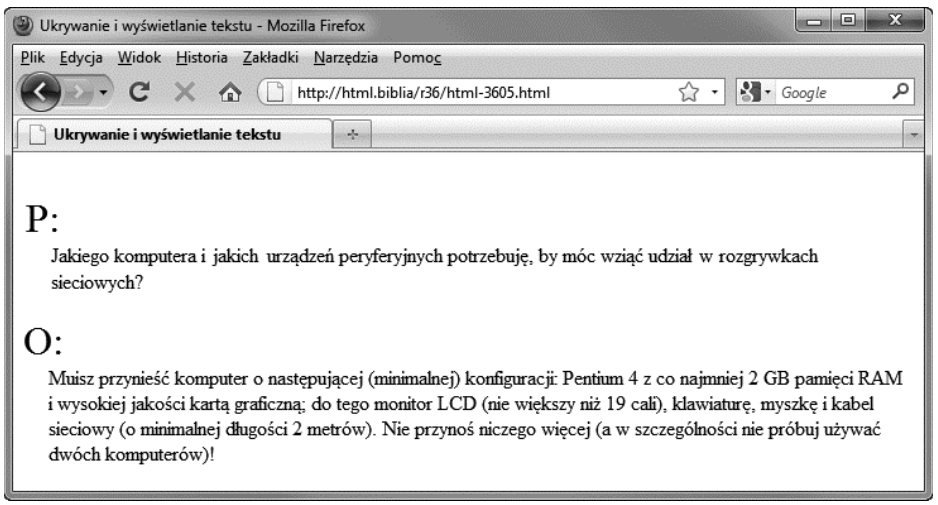
Dzięki wykorzystaniu kaskadowych arkuszy stylów i JavaScriptu bez większych problemów można ukrywać i wyświetlać teksty oraz inne elementy stron. Rozwiązanie takie może być wykorzystane w przeróżnych sytuacjach, takich jak tworzenie rozwijanego menu bądź ukrywanie fragmentu strony do momentu, gdy użytkownik zdecyduje się go wyświetlić. Na przykład przeanalizujmy stronę zawierającą listę pytań i odpowiedzi. Można sądzić, że użytkownik nie będzie chciał oglądać całej listy odpowiedzi, już bezpośrednio po wyświetleniu strony. Jest bardziej prawdopodobne, że będzie chciał selektywnie wyświetlać odpowiedzi tylko na te pytania, które go interesują. Dzięki zastosowaniu właściwości CSS `display` napisanie skryptu zapewniającego takie możliwości funkcjonalne nie nastręcza żadnych problemów.

Przeanalizuj poniższy kod strony, w którym odpowiedzi na pytania są początkowo ukryte, a przeglądarka wyświetli je dopiero po kliknięciu dużej litery „P” lub tekstu pytania. Rysunek 36.5 przedstawia stronę bezpośrednio po wyświetleniu (z ukrytą odpowiedzią), natomiast rysunek 36.6 — stronę po wyświetleniu odpowiedzi.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Ukrywanie i wyświetlanie tekstu</title>
  <style type="text/css">

  /* Początkowo ukrywamy wszystkie odpowiedzi */
    .hideseek { display: none; }

  /* Różne stany widoczności pytań i odpowiedzi */
    .Q { font-size: xx-large;
         padding-bottom: 0;
         margin-bottom:0;
         cursor: pointer; }
    .Qtext { margin-left: 20px;
             margin-top: 0;
             padding-top: 0; }
    .A { font-size: xx-large;
         padding-bottom: 0;
         margin-bottom: 0;
         clear: left; }
```



Rysunek 36.5. *Odpowiedzi na pytania są początkowo ukryte*



Rysunek 36.6. *Odpowiedź jest wyświetlana po kliknięciu dużej litery P (bądź dowolnego innego fragmentu pytania). Ponowne kliknięcie powoduje ukrycie odpowiedzi*

```
    .Atext { margin-left: 20px;
             margin-top: 0;
             padding-top: 0; }

</style>

<script type="text/javascript">

  // Naprzemiennie ukrywamy i wyświetlamy element
  function hideseek(id) {
    obj = document.getElementById(id);
 // Jeśli styl nie jest określony, to możemy założyć,
 // że element nie był jeszcze modyfikowany i jest niewidoczny
```

```
      if ((obj.style.display == "") ||
          (obj.style.display == "none")) {
        obj.style.display = "block";
      } else {
        obj.style.display = "none";
      }
    }

  </script>
</head>
<body>
  <div onClick="hideseek('A1');">
    <p class="Q" >P:</p>
    <p class="Qtext">Jakiego komputera i urządzeń peryferyjnych potrzebuję, by móc
    wziąć udział w rozgrywkach sieciowych?</p>
  </div>
  <div id="A1" class="hideseek">
    <p class="A" >O:</p>
    <p class="Atext">
      Musisz przynieść komputer o następującej (minimalnej) konfiguracji: Pentium 4
      z co najmniej 2 GB pamięci RAM i wysokiej jakości kartą graficzną; do tego
      monitor LCD (nie większy niż 19 cali), klawiaturę, myszkę i kabel sieciowy
      (o minimalnej długości 2 metrów). Nie przynoś niczego więcej
      (a w szczególności nie próbuj używać dwóch komputerów)!
    </p>
  </div>
</body>
</html>
```

Powyższy przykład wykorzystuje funkcję JavaScript, która odczytuje bieżącą wartość właściwości `display` i ustawia ją na wartość przeciwną, co powoduje naprzemienne wyświetlanie i ukrywanie elementu. Funkcja ta jest wywoływana przez procedurę obsługi zdarzeń `onClick`, zdefiniowaną w elemencie `<div>` zawierającym pytanie. W wywołaniu funkcji umieszczony jest identyfikator elementu `<div>` zawierającego odpowiedź, dzięki czemu wie ona, na jakim elemencie ma operować.

Jak już wcześniej wspomniano, takie rozwiązanie może być stosowane w przeróżnych celach. Wystarczy ukryć element, który początkowo ma być niewidoczny (wykorzystując do tego celu właściwość `display` z wartością `none`), a następnie wyświetlać go przy użyciu funkcji wywoływanej po kliknięciu przycisku lub zajściu jakiegokolwiek innego zdarzenia.

Powiększanie obrazków

Innym często spotykanym zastosowaniem możliwości CSS do tworzenia ciekawych efektów wizualnych jest powiększanie obrazków, które początkowo są wyświetlane w postaci miniaturek. Technika ta jest powszechnie wykorzystywana we wszelkiego typu galeriach internetowych lub w innych witrynach, na których wyświetlanie obrazków w pełnej wielkości jest pożądane, lecz kosztowne.

Prezentowana tu technika jest podobna do rozwiązania przedstawionego w poprzednim punkcie rozdziału — także w tym przypadku generowane przez użytkownika zdarzenie wywołuje skrypt, który odpowiednio modyfikuje właściwości CSS danego elementu.

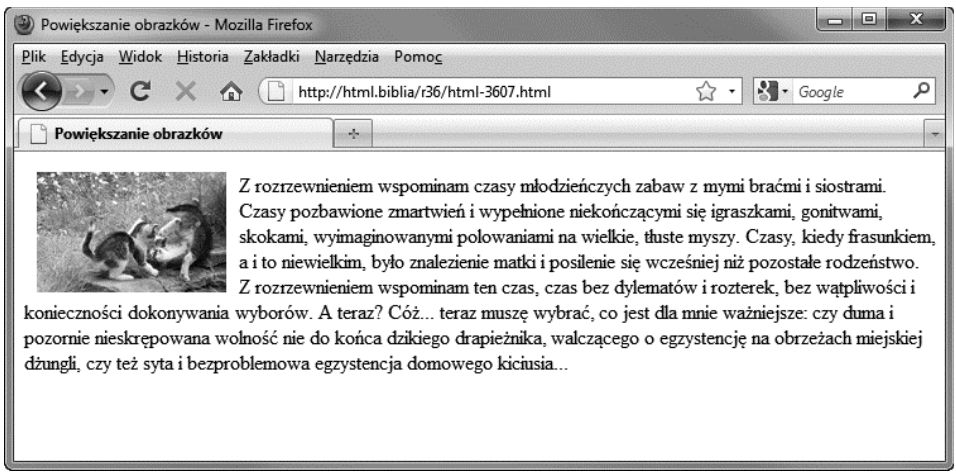
W prezentowanym przykładzie skrypt operuje na właściwości `display` miniatury oraz pełnowymiarowego obrazka. Poniżej przedstawiony został kod przykładowej strony WWW:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Powiększanie obrazków</title>
  <style type="text/css">
    .zoom { display: none; margin: 0px 10px;
            float: left;}
    .zoom p {margin: 0;}
    .thumb { display: block; margin: 0px 10px;
             float: left; }
    .thumb p {margin: 0;}
  </style>
  <script type="text/javascript">
    function PicZoom(id) {
      pic = document.getElementById(id);
      thum = document.getElementById("T"+id);
      if ((pic.style.display == "") ||
          (pic.style.display == "none")) {
        pic.style.display = "block";
        thum.style.display = "none";
      } else {
        pic.style.display = "none";
        thum.style.display = "block";
      }
    }
  </script>
</head>

<body>

  <div id="1" class="zoom"><p><img src="cats.jpg" alt="Kotki"
  onMouseOut="PicZoom(1)"/> </p></div>
  <div id="T1" class="thumb"><p><img src="cats_thumb.jpg" alt="Kotki"
  onMouseOver="PicZoom(1)" /></p></div>
  <div class="text"><p>
    Z rozrzewnieniem wspominam czasy młodzieńczych zabaw z mymi braćmi i siostrami.
    Czasy pozbawione zmartwień i wypełnione niekończącymi się igraszkami, gonitwami,
    skokami, wyimaginowanymi polowaniami na wielkie, tłuste myszy. Czasy, kiedy
    frasunkiem, a i to niewielkim, było znalezienie matki i posilenie się wcześniej
    niż pozostałe rodzeństwo. Z rozrzewnieniem wspominam ten czas, czas bez
    dylematów i rozterek, bez wątpliwości i konieczności dokonywania wyborów.
    A teraz? Cóż... teraz muszę wybrać, co jest dla mnie ważniejsze: czy duma
    i pozornie nieskrępowana wolność nie do końca dzikiego drapieżnika, walczącego
    o egzystencję na obrzeżach miejskiej dżungli, czy też syta i bezproblemowa
    egzystencja domowego kiciusia...
  </p></div>
</body>
</html>
```

Aby ułatwić formatowanie strony i zapewnić większą elastyczność, wszystkie obrazki, zarówno miniaturka, jak i obrazek pełnowymiarowy, zostały umieszczone w elementach `<div>`. Właściwość `display` elementu `<div>` zawierającego miniaturkę ma początkowo wartość `block`, co sprawia, że element ten jest widoczny. Natomiast w elemencie `<div>`

zawierającym pełnowymiarowy obrazek właściwość `display` ma początkowo wartość `none`, a zatem element ten jest początkowo ukryty. W elemencie `img` prezentującym miniaturkę zdefiniowano procedurę obsługi zdarzeń `onMouseOver`, w której wywoływana jest funkcja `PicZoom()`. Dzięki temu funkcja ta zostanie wywołana w momencie umieszczenia wskaźnika myszy na miniaturce. Działanie funkcji `PicZoom()` polega na zamianie wartości właściwości `display` elementów `<div>` zawierających obrazki; innymi słowy, pojawi się pełnowymiarowy obrazek, a miniaturka zniknie. Wskaźnik myszy pozostanie w tym samym położeniu — w obszarze pełnowymiarowego zdjęcia. Kiedy użytkownik przesunie go poza obrazek, zostanie zgłoszone zdarzenie `onMouseOut`, a w efekcie ponownie zostanie wywołana funkcja `PicZoom()`, która tym razem ukryje pełnowymiarowy obrazek i wyświetli miniaturkę.

Działanie tej strony zostało zilustrowane na rysunkach 36.7 oraz 36.8. Pierwszy z nich przedstawia stronę w jej początkowej postaci, z widoczną miniaturką; natomiast na rysunku 36.8 widoczna jest strona po wyświetleniu obrazka pełnowymiarowego.



Rysunek 36.7. *Początkowo, po wyświetleniu strony, widoczna jest miniaturka, natomiast pełnowymiarowy obrazek jest ukryty*

Podobnie jak w pozostałych przykładach przedstawionych w tym rozdziale, podobny efekt można uzyskać na wiele sposobów. Jeden z nich polega na umieszczeniu wszystkich obrazków na jednym dużym i przesuwanie go w taki sposób, by w danym momencie był widoczny jego odpowiedni fragment. Inne rozwiązanie polega na modyfikowaniu atrybutu `src` znacznika `<img>` tak, by wyświetlany był w nim odpowiedni obrazek. Istnieje także bardziej złożona technika, bazująca na wykorzystaniu warstw i odpowiednich modyfikacjach właściwości `z-index` wybranego elementu. Nic nie stoi na przeszkodzie, byś wymyślił także swoje własne rozwiązanie.

Podmieniane menu

We wszystkich przykładach przedstawionych we wcześniejszej części rozdziału uzyskiwaliśmy zamierzony efekt wizualny, stosując kod JavaScript do modyfikowania właściwości CSS. Jak na razie nie skorzystaliśmy z pseudoklas CSS dostępnych w elementach łączy, które pozwalają na uzyskanie podobnych efektów.

Rysunek 36.8. *Po umieszczeniu wskaźnika myszy na miniaturce, zamiast niej w przeglądarce jest wyświetlany obrazek w pełnych wymiarach*

Pseudoklasy elementów `<a>` przedstawione zostały w tabeli 36.1.

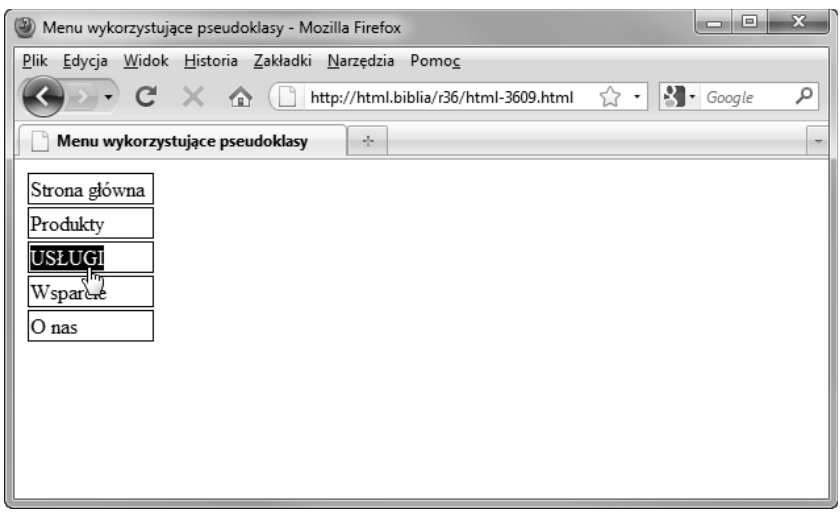
Tabela 36.1. *Pseudoklasy elementu <a>*

Pseudoklasa	Zastosowanie lub efekt
`:link`	Formatuje elementy wybrane przez selektor jako łącza, które jeszcze nie zostały odwiedzone.
`:visited`	Formatuje elementy wybrane przez selektor jako odwiedzone łącza.
`:hover`	Formatuje elementy wybrane przez selektor w momencie umieszczenia na nich wskaźnika myszy.
`:active`	Formatuje elementy wybrane przez selektor jako aktywne łącza.

Pseudoklasy wymienione w tabeli 36.1 są zazwyczaj stosowane do określania wyglądu elementów traktowanych jak dynamiczne łącza. Na przykład, korzystając z pseudoklasy `:hover`, można dynamicznie zmieniać wygląd elementu w momencie umieszczenia nad nim wskaźnika myszy — czyli dokładnie tak samo, jak zachowują się łącza.

Na przykład przedstawiony poniżej dokument HTML wykorzystuje pseudoklasę `:hover`, by zmieniać wygląd łączy po wskazaniu ich myszą. Uzyskany efekt przypomina dynamicznie modyfikowane menu tworzone przy użyciu kodu JavaScript. Efekty jego działania zostały przedstawione na rysunku 36.9.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Menu wykorzystujące pseudoklasy</title>
  <style type="text/css">
    .nav tr td {border: 1px solid black;}
    .menu { color: black;
            background-color: white;
            text-transform: none;
            text-decoration: none; }
    .menu:hover { color: white;
                  background-color: black;
                  text-transform: uppercase;
                  text-decoration: none; }
    .menucase { width: 100px; }
  </style>
</head>

<body>
  <div class="menucase">
    <table border="0" width="100%" class="nav">
      <tr><td><a class="menu" href="index.html">Strona główna</a></td></tr>
      <tr><td><a class="menu" href="produkty.html">Produkty</a></td></tr>
      <tr><td><a class="menu" href="uslugi.html">Usługi</a></td></tr>
      <tr><td><a class="menu" href="wsparcie.html">Wsparcie</a></td></tr>
      <tr><td><a class="menu" href="onas.html">O nas</a></td></tr>
    </table>
  </div>
</body>
</html>
```



Rysunek 36.9. *Pseudoklasy mogą być stosowane do tworzenia dynamicznego menu, w tym przypadku wskazany element menu jest podświetlany*

Należy zauważyć, że choć technika polegająca na zastosowaniu pseudoklas w celu uzyskania ciekawych efektów wizualnych jest bardzo popularna, to jednak nie jest ona zgodna z zasadą separacji działań od prezentacji. Dlatego też znacznie lepszym rozwiązaniem jest tworzenie podobnych efektów przy wykorzystaniu kodu JavaScript.

Zwróć uwagę, że opisana tu technika może być stosowana w połączeniu niemal ze wszystkimi elementami HTML. Jednak pseudoklas można używać wyłącznie do formatowania znaczników łączy, a zatem by odpowiednio formatować łącza w dokumencie, należy korzystać z innych znaczników.

Podsumowanie

W tym rozdziale przedstawione zostały technologie DHTML i CSS. Dowiedziałeś się, jak używać JavaScriptu, by modyfikować style CSS elementów i uzyskiwać ciekawe efekty wizualne, oraz jak tworzyć takie efekty przy użyciu pseudoklas CSS. W następnym rozdziale zamieszczone zostały informacje dotyczące stosowania kaskadowych arkuszy stylów do określania postaci stron przeznaczonych do wydruku. W kolejnych rozdziałach, od 38. do 41., przedstawione zostały bardziej specjalistyczne zagadnienia związane ze stosowaniem kaskadowych arkuszy stylów.