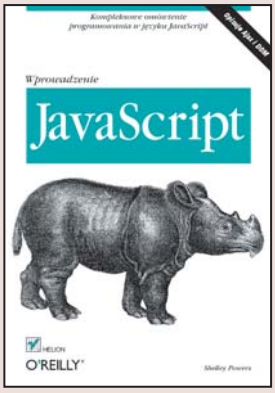
JavaScript. Wprowadzenie

Autor: Shelley Powers
Tłumaczenie: Anna Trojan
ISBN: 978-83-246-0942-0
Tytuł oryginału: Learning JavaScript
Format: B5, stron: 344



Kompleksowe omówienie programowania w JavaScript

- Jak weryfikować poprawność danych w formularzach HTML?
- Jak zabezpieczyć aplikacje internetowe?
- W jaki sposób korzystać z mechanizmów AJAX?

Zamierzasz wzbogacić tworzone przez siebie witryny internetowe o dodatkowe możliwości? A może zainteresowała Cię technologia AJAX, bazująca na języku JavaScript? Najwyższa pora poznać język, który na początku był jedynie skryptowym interfejsem pomiędzy stroną internetową otwartą w przeglądarce a aplikacją znajdującą się na serwerze. Dziś JavaScript jest jednym z najpopularniejszych narzędzi wykorzystywanych przez twórców stron i aplikacji internetowych. Rozpowszechnił się również poza światem internetu, służąc jako język skryptowy dla wielu aplikacji.

„JavaScript. Wprowadzenie" to podręcznik, dzięki któremu opanujesz możliwości JavaScriptu i nauczysz się wykorzystywać go w swoich projektach. Poznasz podstawowe elementy tego języka i dowiesz się, w jaki sposób łączyć kod JavaScript z kodem HTML. Nauczysz się przechwytywać zdarzenia zachodzące w oknie przeglądarki, sprawdzać dane wprowadzane przez użytkowników do formularzy na stronach WWW oraz korzystać z plików cookie. Poznasz obiektowy model dokumentu (DOM), technologię AJAX i dodatkowe biblioteki, dzięki którym JavaScript zyskuje nowe, niesamowite możliwości.

- Osadzanie JavaScriptu w kodzie strony
- Typy danych, operatory i instrukcje
- Obiekty JavaScriptu
- Obsługa formularzy
- Stosowanie plików cookie
- Obiektowy model dokumentu
- Tworzenie własnych obiektów
- Technologia AJAX

Wykorzystaj pełnię możliwości JavaScriptu, tworząc nowoczesne witryny internetowe

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

Spis treści

ROZDZIAŁ 12.

Tworzenie dynamicznych stron internetowych — dodawanie stylu do skryptu

W roku 1996 zostałam zaproszona na poufną prezentację nowych technologii, które Microsoft zamierzał opublikować w przeciągu roku. Przyjechałam z Portland w stanie Oregon do kampusu Microsoftu w Seattle, gdzie wraz z kilkoma innymi autorami oraz redaktorami z różnych wydawnictw spotkaliśmy się w dość miłym pokoju konferencyjnym (z doskonałym bufetem).

Jeden z menedżerów Microsoftu przedstawił projekcję obrazu strony internetowej, która nie była niczym specjalnym. Do momentu, kiedy kliknął nagłówek na stronie, a materiał umieszczony pod nim zjechał na dół jak ukryty akapit. Niewielka rzecz i w dzisiejszych czasach nic specjalnego, jednak wtedy zrobiło to na mnie ogromne wrażenie.

To było moje pierwsze spotkanie z koncepcją, która stała się znana pod nazwą dynamicznego HTML czy też DHTML. W końcu napisałam nawet książkę na ten temat oraz kilka artykułów omawiających DHTML zgodny z różnymi przeglądarkami. Przełomowym momentem dla tej koncepcji było wprowadzenie nowej specyfikacji konsorcjum W3C, Cascading Style Sheets, obok koncepcji Document Object Model, choć w ówczesnych czasach nie istniał jeden uniwersalny model.

To dzięki CSS można definiować wygląd elementów strony bez konieczności polegania na zewnętrznych aplikacjach, dodatkach bądź nadmiernego użycia obrazków. To również dzięki CSS i arkuszom stylów możliwe jest oddzielenie prezentacji elementów strony od ich organizacji.

Jednak to dzięki DOM możliwy stał się dostęp do właściwości arkuszy stylów z JavaScriptu i zmiana pojedynczych właściwości elementu nawet po zakończeniu ładowania strony. W połączeniu z CSS był to potężny środek umożliwiający uczynienie strony internetowej o wiele bardziej interaktywną niż wcześniej.

Jedynym problemem było to, że każda z firm, które wtedy oferowały liczącą się przeglądarkę — najpopularniejsze były wówczas Netscape Navigator oraz Internet Explorer z Microsoftu — implementowała odmienny DOM, co czyniło DHTML dość skomplikowanym.

Choć przeglądarki w wersji 4 były w stanie stworzyć zadziwiające efekty, wiązało się to z pewnymi kosztami. Strona musiała zawierać kod służący do utworzenia efektu, który będzie działał nie tylko w każdej nowej przeglądarce, ale również w starszych przeglądarkach, nieobsługujących DHTML. Przede wszystkim ze względu na te właśnie trudności DHTML dogorywał, nie będąc szeroko używanym, dopóki nie nadszedł czas bardziej nowoczesnych przeglądarek. Teraz DHTML na nowo wzbudza zainteresowanie dodatkowo wzmocnione przez wzrastającą popularność Ajaksa (omówionego w rozdziałach 13. oraz 14.).

Jak wspomniano w rozdziale 11., dysponuję zbiorem budowanych od 1998 roku obiektów oraz animacji dla wszystkich przeglądarek opartym na DHTML. Współczesną wersję tego zbioru wraz z kilkoma przykładami użycia można pobrać ze strony internetowej *Learning JavaScript* (*http://burningbird.net/tutorials/xobjsintro.htm*.).

DHTML — JavaScript, CSS oraz DOM

Kaskadowe arkusze stylów (CSS) miały trudne początki. Pomysł umieszczenia prezentacji elementów strony w osobnej specyfikacji znany był od początków Internetu, jednak został odłożony na bok przez pierwszych projektantów przeglądarek. Dopiero po 1996 roku — kiedy wydano pierwszą wersję CSS, a następnie przeglądarki w wersji 4.x — CSS w końcu stał się rzeczywistością. Nie za szybko, choć programiści stron internetowych byli dość sfrustrowani ich ograniczeniami.

W tamtych czasach większość stron internetowych składała się z tabel HTML, które początkowo nie były zaplanowane jako służące do zarządzania układem strony, a jedynie organizacją danych. Problemy z układem strony wiązały się na przykład z tym, że cała strona nie była wyświetlana, dopóki nie zostały załadowane wszystkie obrazki, nie wspominając o kwestiach wiążących się z różnymi przeglądarkami. Jeśli pracowało się ze stronami internetowymi z tamtych lat, z pewnością zna się elementy `font` oraz `blink`.

CSS udostępnił schludną alternatywę. Dzięki niej można było inicjalizować i modyfikować różne kategorie właściwości prezentacyjnych. Obejmują one tło dokumentu, czcionki, kolory, obramowania i rozmiary pól, marginesy oraz dopełnienia — jeśli mają one zastosowanie. Były one bardzo przyjemnym dodatkiem do zbioru narzędzi programisty strony internetowej, jednak czegoś w tym wszystkim brakowało — możliwości pozycjonowania elementów oraz kontrolowania ich układu, a także ich wyświetlania i widoczności. Te właściwości stylu zostały opublikowane dopiero kiedy firmy Netscape oraz Microsoft wspólnie wypracowały wczesne wydanie pozycyjnego CSS, zwanego CSS-P. W końcu stały się one częścią drugiego wydania CSS — CSS2.

W niniejszym rozdziale założono, że Czytelnik zna CSS i wie, jak do strony internetowej dodaje się arkusze stylów. Jeśli jest inaczej, może przed przejściem do lektury niniejszego rozdziału przeczytać jakiś przewodnik bądź książkę na ten temat. Polecam pozycję Erica A. Meyera *CSS. Kaskadowe arkusze stylów. Przewodnik encyklopedyczny* (Helion). Istnieje również wiele kursów online, które można znaleźć w Internecie. Jedną z popularnych stron jest W3 Schools, znajdująca się pod adresem: *http://www.w3schools.com/css/default.asp*.

Właściwość style

Właściwości stylu CSS są zazwyczaj odczytywane i ustawiane za pośrednictwem obiektu `style`. Koncepcja właściwości `style` narodziła się w Microsofcie, jednak została przyjęta przez konsorcjum W3C i zawarta w module DOM Level 2 CSS. Dzięki W3C DOM każdy węzeł posiada powiązany obiekt `style` jako właściwość, co oznacza, że możliwa jest zmiana właściwości stylu dowolnego elementu strony za pomocą JavaScriptu.

By zmienić dowolne ustawienia stylu za pomocą JavaScriptu, należy najpierw skorzystać z jednej z metod dostępu DOM, omówionej w rozdziałach 9. oraz 10., w celu uzyskania dostępu do pojedynczego elementu (bądź elementów). By zmienić jego atrybut `style`, należy skorzystać z prostego przypisania:

```
element.style.color="#fff";
```

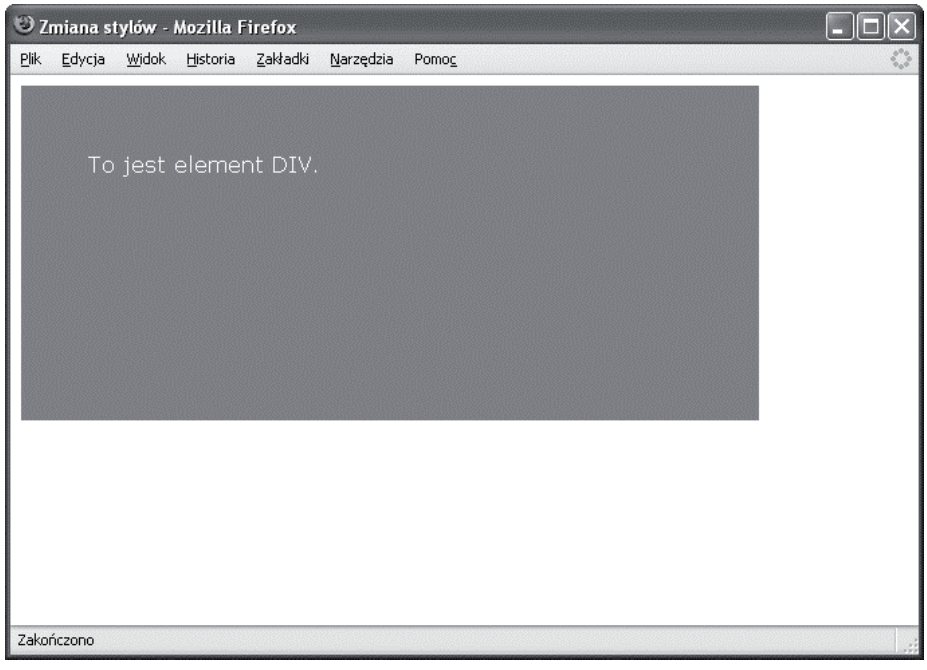
Działa to dla każdego poprawnego atrybutu CSS2 i na każdym poprawnym obiekcie XHTML. Na listingu 12.1 zaprezentowano sposób modyfikacji kilku atrybutów CSS dzięki wykorzystaniu znajomej metody `getElementById` w celu uzyskania dostępu do elementu `DIV` oraz obiektu `style`.

Listing 12.1. Zastosowanie kilku zmian właściwości style na obiekcie DIV

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Zmiana stylów</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <script type="text/javascript">
         //<![CDATA[

         function changeElement() {
            var div = document.getElementById("div1");
            div.style.backgroundColor="#f00";
            div.style.width="500px";
            div.style.color="#fff";
            div.style.height="200px";
            div.style.paddingLeft="50px";
            div.style.paddingTop="50px";
            div.style.fontFamily="Verdana";
            div.style.borderColor="#000";
         }

         //]]>
      </script>

   </head>
   <body onload="changeElement();">
      <div id="div1">
         To jest element DIV.
      </div>
   </body>
</html>
```

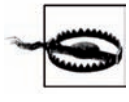
Warto zwrócić uwagę na konwencję nazewnictwa wykorzystywaną we właściwościach CSS. Jeśli właściwość ma w środku myślnik, tak jak `border-color`, myślnik ten jest usuwany, a pierwsza litera drugiego wyrazu zastępowana jest dużą literą — `border-color` z CSS staje się

borderColor w JavaScripcie. Poza tym nazwy właściwości CSS wykorzystywane w JavaScripcie są dokładnie takie same jak nazwy właściwości z arkusza stylów. Na rysunku 12.1 zaprezentowano wygląd elementu DIV i jego zawartości po dokonaniu zmian stylu.



Rysunek 12.1. Zastosowanie kilku zmian stylu

Choć modyfikowanie atrybutu style jest proste, to już jego odczytywanie jest nieco trudniejsze. Jeśli właściwość style nie zostanie ustawiona za pomocą JavaScriptu bądź atrybutu style wstawionego do elementu, to wartość tej właściwości będzie albo pusta, albo niezdefiniowana, nawet jeśli w rzeczywistości wartość ta została wcześniej ustawiona w arkuszu stylu. Ważne, by o tym pamiętać, ponieważ przeszkoda ta pojawia się częściej niż jakakolwiek inna, kiedy pracuje się z DHTML. Ustawienia style wykorzystywane do początkowego generowania obiektu są wewnętrzne dla przeglądarki i oparte na połączeniu ustawień arkusza stylów oraz dziedziczeniu elementów.



Należy to wyraźnie powtórzyć: jeśli właściwość style nie zostanie ustawiona za pomocą JavaScriptu bądź bezpośrednio za pomocą atrybutu style elementu, jej wartość pozostanie pusta bądź niezdefiniowana, kiedy uzyskuje się do niej dostęp ze skryptu, nawet jeśli wartość tę ustawi się wcześniej za pośrednictwem arkusza stylów.

By uzyskać dostęp do stylu, należy skorzystać z różnych właściwości, specyficznych dla każdej przeglądarki. Microsoft oraz Opera obsługują właściwość currentStyle elementu, natomiast Firefox, Mozilla oraz Netscape obsługują window.getComputedStyle. Niestety, obydwa te sposoby nie działają spójnie w różnych przeglądarkach.

Do metody `getComputedStyle` należy przekazać atrybut CSS za pomocą tej samej składni, którą wykorzystuje się przy ustawianiu stylu w arkuszu stylu. W metodzie `currentStyle` należy jednak wykorzystywać notację z JavaScriptu. W przypadku przeglądarki Safari nie ma to znaczenia, ponieważ nie obsługuje ona żadnej z wymienionych metod.

Na listingu 12.2 zademonstrowano wariant funkcji, która zwraca ustawienia stylu dla obiektu i określonej właściwości CSS. Najpierw sprawdza ona, czy obsługiwana jest metoda `currentStyle`, a jeśli tak nie jest, sprawdza istnienie `window.getComputedStyle`. Jeśli żadna z metod nie jest obsługiwana, po prostu zwraca `null`. Uzyskiwany jest dostęp do właściwości `style`, która jest wyświetlana zarówno przed jej ustawieniem, jak i po.

Listing 12.2. Próba otrzymania informacji o stylu CSS

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Nieśmiały styl</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <style type="text/css">
         #div1 { background-color: #ff0 }
      </style>
      <script type="text/javascript">
         //<![CDATA[

         document.onclick=changeElement;

         function getStyle(obj,jsprop,cssprop) {
            if (obj.currentStyle) {
               return obj.currentStyle[jsprop];
            } else if (window.getComputedStyle) {
               return document.defaultView.getComputedStyle(obj,null).
               getPropertyValue(cssprop);
            } else {
               return null;
            }
         }

         function changeElement() {
            var obj = document.getElementById("div1");
            alert(obj.style.backgroundColor);
            alert(getStyle(obj,"backgroundColor","background-color"));
            obj.style.backgroundColor="#ff0000";
            alert(getStyle(obj,"backgroundColor","background-color"));
            alert(obj.style.backgroundColor);
         }

         //]]>
      </script>
   </head>
   <body>
      <div id="div1">
         <p>To jest element DIV.</p>
      </div>
   </body>
</html>
```

Warto zwrócić uwagę, że wykorzystana w skrypcie składnia, służąca do otrzymania obliczonej wartości, to `document.defaultView.getComputedStyle`, a nie `window.getComputedStyle`. Przyczyną takiego postępowania jest fakt, iż `document.defaultView` zwraca obiekt DOM

`AbstractView`, który jest podstawowym interfejsem, z którego pochodzą wszystkie widoki. Można go zmienić na obiekt `window`, jednak nie ma żadnej gwarancji, że to zachowanie nie zmieni się z przeglądarki na przeglądarkę oraz z wersji na wersję. Dlatego też dla uzyskania właściwości `style` lepiej będzie skorzystać z `document.defaultView.getComputedStyle`.

Nawet kiedy właściwość `style` jest dostępna, to to, co naprawdę jest zwracane, jest różne w różnych przeglądarkach — tak jak zwykły kolor. Opera zwraca kolor w formacie szesnastkowym:

```
#ff0000
```

natomiast Firefox zwraca ustawienia RGB:

```
RGB(255,0,0)
```

Dlatego też, jeśli chce się otrzymać spójne wyniki, należy przeprowadzić konwersję pomiędzy tymi dwoma formatami.

Pobranie ustawień stylu ze strony wiąże się z interesującymi wyzwaniami. Być może nawet w większym stopniu niż byłoby to zabawne bądź użyteczne. Dobrą zasadą podczas pracy z DHTML jest unikanie pobierania informacji bezpośrednio z ustawień stylu strony. Zamiast tego, kiedy tylko jest to możliwe, należy wykorzystywać zmienne programu do przechowywania wartości i do ustawiania atrybutów używać jedynie `style`.

Właściwości `style` CSS dzielą się na grupy podobnych właściwości: związanych z czcionkami, obramowaniami, pojemnikami na elementy, pozycjonowaniem, wyświetlaniem i tak dalej. W pozostałej części rozdziału omówionych zostanie kilka atrybutów i zademonstrowany będzie sposób, w jaki można pracować z każdym z nich z użyciem JavaScriptu. Z pewnością warto poświęcić nieco czasu na zatrzymanie się i wykonanie kilku modyfikacji wszystkich z przykładów.

Nie omówiono tutaj otrzymywania informacji o stylu za pośrednictwem zbioru arkuszy stylów `stylesheets`. Jest to nowy zbiór, który nie jest częścią oryginalnego Browser Object Model. Wykorzystywanie tego rozwiązania pozwala na obejście niektórych problemów związanych ze zgodnością przeglądarek oraz ustawianiem atrybutów przedstawionych w niniejszym rozdziale. By przyjrzeć się przykładowi i omówieniu tego rozwiązania, warto przeczytać artykuł „Modifying Styles" autorstwa Stevena Champeona, znajdujący się pod adresem: *http://developer.apple.com/internet/webcontent/styles.html*.

Czcionki oraz tekst

Jednym z pierwszych elementów HTML charakterystycznych dla prezentacji był element `font`. Jest on również jednym ze starszych elementów HTML, które nadal zbyt często można odnaleźć na stronach internetowych. Nie powinno być zaskoczeniem, że właśnie właściwości `font` oraz `text` były tak interesujące dla budowania stron internetowych. Niewiele zmian, które można wprowadzić w atrybutach `style` elementu, może przynieść taki efekt jak zmiany `text` bądź `font`.

Warto zauważyć, że mowa jest o właściwościach `font` oraz `text`. Właściwość `font` wiąże się z samymi znakami: ich rodziną, rozmiarem, typem i innymi cechami ich wyglądu. Atrybuty `text` mają więcej wspólnego z dekoracją dołączoną do tekstu, jego położeniem i tak dalej.

Właściwości stylu czcionki

Istnieje kilka atrybutów stylu dla czcionek. Ich nazwy z CSS oraz powiązane atrybuty `style` dostępne w JavaScripcie podane są na poniższej liście:

`font-family`
: W JavaScripcie dostępny jako `fontFamily`. Określa rodzinę czcionek (taką jak Serif, Arial, Verdana) dla czcionki. Kiedy podaje się rodzinę czcionek składającą się z kilku słów, należy je wpisać dokładnie, wraz ze spacjami.

`font-size`
: W JavaScripcie dostępny jako `fontSize`. Ustawia rozmiar czcionki. Przy ustawianiu rozmiaru możliwe jest korzystanie z kilku różnych jednostek. Jeśli wykorzystuje się `em` bądź `pt` (jak na przykład `12pt` lub `2.5em`), rozmiar czcionki zmieniany jest zgodnie z ustawieniami użytkownika strony. Jeśli korzysta się z jednostki `px` (piksel), czcionka ma stały rozmiar bez względu na ustawienia użytkownika. Można albo określić konkretną jednostkę za pomocą JavaScriptu, albo skorzystać z jednego z predefiniowanych rozmiarów czcionki — `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` oraz `xx-large`. Można również korzystać z rozmiarów względnych, poza wykorzystywaniem wartości procentowej opartej na rozmiarze czcionki elementu rodzica.

`font-size-adjust`
: W JavaScripcie dostępny jako `fontSizeAdjust`. Jest to stosunek pomiędzy wysokością litery `x` a wysokością określoną w `font-size`. Ustawienie to przechowuje ten stosunek, choć rzadko jest on podawany.

`font-stretch`
: W JavaScripcie dostępny jako `fontStretch`. Rozszerza bądź zwęża czcionkę. Można skorzystać z jednej z następujących wartości: `normal`, `wider`, `narrower`, `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `semi-expanded`, `expanded`, `ultra-expanded` bądź `extra-expanded`.

`font-style`
: W JavaScripcie dostępny jako `fontStyle`. Można korzystać z wartości `normal` (`default`), `italic` bądź `oblique`.

`font-variant`
: W JavaScripcie dostępny jako `fontVariant`. Jeśli chce się użyć kapitalików, należy skorzystać z wartości `small-caps`.

`font-weight`
: W JavaScripcie dostępny jako `fontWeight`. Należy ustawić wagę czcionki (grubość). Można wykorzystać wartości `normal`, `bold`, `bolder`, `lighter` bądź też wartości liczbowe — `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800` lub `900`.

Jak pokazano na listingu 12.1, zmiana czcionki elementów zmienia czcionkę dla całego tekstu, który mieści się w tym elemencie, o ile nie zostanie to nadpisane przez ustawienia stylu elementu zawartego — na tym polega kaskadowość CSS. Jest to zachowanie charakterystyczne dla CSS; wykorzystywanie JavaScriptu do dynamicznej zmiany czcionki nie ma wpływu na ten efekt.

Można zmienić wiele atrybutów czcionki za jednym razem, używając do tego samego `font`. W poniższym kodzie:

```
div.style.font="italic small-caps 400 14px verdana";
```

wykorzystano atrybut `font` bez żadnej podrzędnej właściwości do ustawienia wartości `font-style`, `font-variant`, `font-weight`, `font-size` oraz `font-family`. Wiele z właściwości CSS posiada podobne skrótowe metody. W JavaScripcie są one przypisywane w taki sam sposób, w jaki byłyby przypisane w samym CSS. W CSS wszystkie ustawienia znajdują się po prawej stronie dwukropka. W JS wszystko, co znajduje się po prawej stronie dwukropka, zawarte jest w cudzysłowie po prawej stronie instrukcji przypisania.

Właściwości stylu tekstu

Poniżej zaprezentowano kilka atrybutów, które wpływają na wygląd tekstu, choć w przeciwieństwie do `font` nie stanowią jednej rodziny. Najczęściej ustawiane atrybuty CSS dla `text` są następujące:

`color`
: W JavaScripcie dostępny jako `color`. Określa kolor tekstu.

`line-height`
: W JavaScripcie dostępny jako `lineHeight`. Przestrzeń pomiędzy dołem poprzedniego wiersza, a górą następnego, czyli odstęp między dwoma wierszami. Należy określić wartość w sposób podobny do określania rozmiaru czcionki bądź skorzystać z `normal`.

`text-decoration`
: W JavaScripcie dostępny jako `textDecoration`. Można wykorzystać `none`, `underline`, `overline` bądź `line-through`.

`text-indent`
: W JavaScripcie dostępny jako `textIndent`. Wskazuje, o ile wcięty ma być pierwszy wiersz tekstu.

`text-transform`
: W JavaScripcie dostępny jako `textTransform`. Można wykorzystać `none`, `capitalize`, `uppercase` bądź `lowercase`.

`white-space`
: W JavaScripcie dostępny jako `whiteSpace`. Można wykorzystać `normal`, `pre` lub `nowrap`.

`direction`
: W JavaScripcie dostępny jako `direction`. Można użyć `ltr` (od lewa do prawa) oraz `rtl` (od prawa do lewa).

`text-align`
: W JavaScripcie dostępny jako `textAlign`. Sposób wyrównania treści tekstu. Można wykorzystać `left`, `right`, `center` bądź `justify`.

`word-spacing`
: W JavaScripcie dostępny jako `wordSpacing`. Liczba odstępów pomiędzy słowami. Można wykorzystać `normal` bądź podać długość.

Jakie są typowe zastosowania modyfikacji właściwości czcionki bądź tekstu? Można rozszerzyć blok tekstu, by był bardziej czytelny, bądź z jakiegoś powodu podkreślić niektóre dane. Na listingu 12.3 kliknięcie jednego z dwóch łączy albo bardzo powiększy tekst i go wyjustuje, albo przywróci go do poprzedniej postaci.

Listing 12.3. Modyfikacja bloku tekstu

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Czytaj TO</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <script type="text/javascript">
         //<![CDATA[

         function makeMore() {
            var div = document.getElementById("div1");
            div.style.fontSize="larger";
            div.style.letterSpacing="10px";
            div.style.textAlign="justify";
            div.style.textTransform="uppercase";
            div.style.fontSize="xx-large";
            div.style.fontWeight="900";
            div.style.lineHeight="40px";
         }

         function makeLess() {
            var div = document.getElementById("div1");
            div.style.fontSize="smaller";
            div.style.letterSpacing="normal";
            div.style.textAlign="left";
            div.style.textTransform="none";
            div.style.fontSize="medium";
            div.style.fontWeight="normal";
            div.style.lineHeight="normal";
         }
         //]]>
      </script>

   </head>
   <body>
      <p>
         <a href="" onclick="makeMore(); return false;">Powiększ wszystko</a>
         <a href="" onclick="makeLess(); return false;">Pomniejsz wszystko</a>
      </p>
      <div id="div1">
         <p>Jednym z pierwszych elementów HTML charakterystycznych dla prezentacji był
         element font. Jest on również jednym ze starszych elementów HTML, które nadal
         zbyt często można odnaleźć na stronach internetowych. Nie powinno być
         zaskoczeniem, że właśnie właściwości font oraz text były tak interesujące
         dla budowania stron internetowych. Niewiele zmian, które można wprowadzić
         w atrybutach style elementu, może przynieść taki efekt jak zmiany text bądź
         font.</p>
         <p>Warto zauważyć, że mowa jest o właściwościach font oraz text. Właściwość
         font wiąże się z samymi znakami: ich rodziną, rozmiarem, typem i innymi
         cechami ich wyglądu. Atrybuty text mają więcej wspólnego z dekoracją
         dołączoną do tekstu, jego położeniem i tak dalej.</p>
      </div>
   </body>
</html>
```

Istnieją szanse, że rzadko zdarzy się konieczność aż tak dużego powiększania tekstu, jednak powyższy przykład pokazuje, jaki rodzaj transformacji można utworzyć za pomocą JavaScriptu oraz CSS. Innym typowym zastosowaniem jest zmiana koloru czcionki w polu tekstowym powiązanym z elementem `form` bądź w bloku tekstu, by pokazać, że nie ma on zastosowania w danym przypadku — czyli dosłowne „wyszarzenie" tekstu.

Pozycja oraz ruch

Przed pojawieniem się CSS, kiedy chciało się kontrolować układ strony w spójny sposób, trzeba było skorzystać z tabel HTML. Jeśli chodzi o jakiś rodzaj animacji, należało wykorzystać albo coś w stylu animowanego obrazka GIF, albo też dodatek taki jak Flash.

Firmy Netscape oraz Microsoft położyły temu kres poprzez wspólne wprowadzenie specyfikacji zwanej CSS-P bądź CSS Positioning. Rozważmy stronę jako wykres, ze współrzędnymi „x" oraz „y". Dzięki CSS-P można ustawić pozycję elementu wewnątrz tego systemu współrzędnych. Jeśli doda się do tego JavaScript, będzie można przemieszczać elementy na stronie.

Zaproponowane atrybuty CSS-P zostały w końcu włączone do specyfikacji CSS2. Właściwości CSS2 związane z pozycjonowaniem są następujące:

`position`
: Właściwość `position` przyjmuje jedną z pięciu wartości: `relative`, `absolute`, `static`, `inherit` bądź `fixed`. Pozycjonowanie `static` jest domyślnym ustawieniem dla większości elementów. Oznacza, że element taki jest częścią układu strony, inne elementy na stronie wpływają na jego pozycję, a on wpływa na wszystkie elementy następujące po nim. Pozycjonowanie `relative` jest podobne, jednak element jest przesunięty względem swojej normalnej pozycji. Pozycjonowanie ustawione na `absolute` wyjmuje element z układu strony i pozwala na bezwzględne ustawienie go na stronie. Pozwala również na układanie elementów warstwami, jeden na drugim, poprzez umieszczenie ich w tej samej lokalizacji. Pozycja `fixed` jest podobna do pozycjonowania bezwzględnego, jednak element jest pozycjonowany względem jakiegoś okna na ekranie. Dla potrzeb większości prac z DHTML wykorzystuje się głównie pozycjonowanie `absolute` bądź `relative`.

`top`
: W układzie współrzędnych strony internetowej wartość „y" rozpoczyna się od góry i jest równa zero. Zwiększa się ona w miarę przechodzenia w dół pojemnika bez względu na to, czy pojemnik jest stroną czy innym elementem. Ustawienie właściwości `top` elementu ustawia jego pozycję względem góry pojemnika.

`left`
: W układzie współrzędnych strony internetowej wartość „x" rozpoczyna się od lewej strony i równa jest zero. Zwiększa się ona w miarę przechodzenia przez kontener z lewej na prawo. Ustawienie właściwości `left` elementu ustawia jego pozycję względem prawej strony pojemnika.

`bottom`
: Punktem odniesienia dla właściwości `bottom` jest dół strony. Wyższe wartości przesuwają element w górę strony.

`right`
: Punktem odniesienia dla właściwości `right` jest prawa strona strony. Wyższe wartości przesuwają element w lewo.

`z-index`
: Czasami przyda się możliwość dodania `z-index`. Jeśli narysuje się linię prostopadłą do strony, będzie to właśnie `z-index`. Jak wspomniano wcześniej, w pozycjonowaniu `absolute` elementy mogą być układane warstwami jeden na drugim. Ich pozycja wewnątrz stosu kontrolowana jest przez jedną z dwóch rzeczy: pierwszą z nich jest ich pozycja na

stronie. Elementy definiowane w stronie internetowej później umieszczane są wyżej na stosie, natomiast elementy wcześniejsze znajdują się na dole stosu. To zachowanie można nadpisać dzięki użyciu `z-index`. Możliwe jest używanie liczb dodatnich oraz ujemnych z wartością `0` będącą normalną warstwą renderowania (w pozycjonowaniu względnym). Wartość `negative` spycha element poniżej, natomiast `positive` — wypycha go wyżej.

Atrybut `display` również ma wpływ zarówno na pozycjonowanie, jak i na układ strony, jednak zostanie omówiony później, w podrozdziale „Wyświetlanie, widoczność i przezroczystość". Także atrybut `float` związany jest z pozycjonowaniem, jednak nie działa zbyt dobrze z DHTML, zatem nie zostanie przedstawiony.

Właściwości `top`, `right`, `bottom` oraz `left`, jak również `z-index`, działają tylko wtedy, gdy atrybut `position` ustawiony jest na `absolute`. Elementy mogą być ustawiane poza stroną poprzez ustawienie dowolnej z właściwości na wartość ujemną. Elementy mogą również być przemieszczane w oparciu o zdarzenia takie jak kliknięcia myszą.

Jeden z efektów DHTML zwany jest ***fly-in*** i polega na tym, że elementy dosłownie wydają się „wlatywać" spoza brzegów dokumentu. To dobre rozwiązanie dla sytuacji, w których chce się wprowadzić jedno zagadnienie po drugim, w oparciu o kliknięcie myszą czy wpis z klawiatury użytkownika strony.

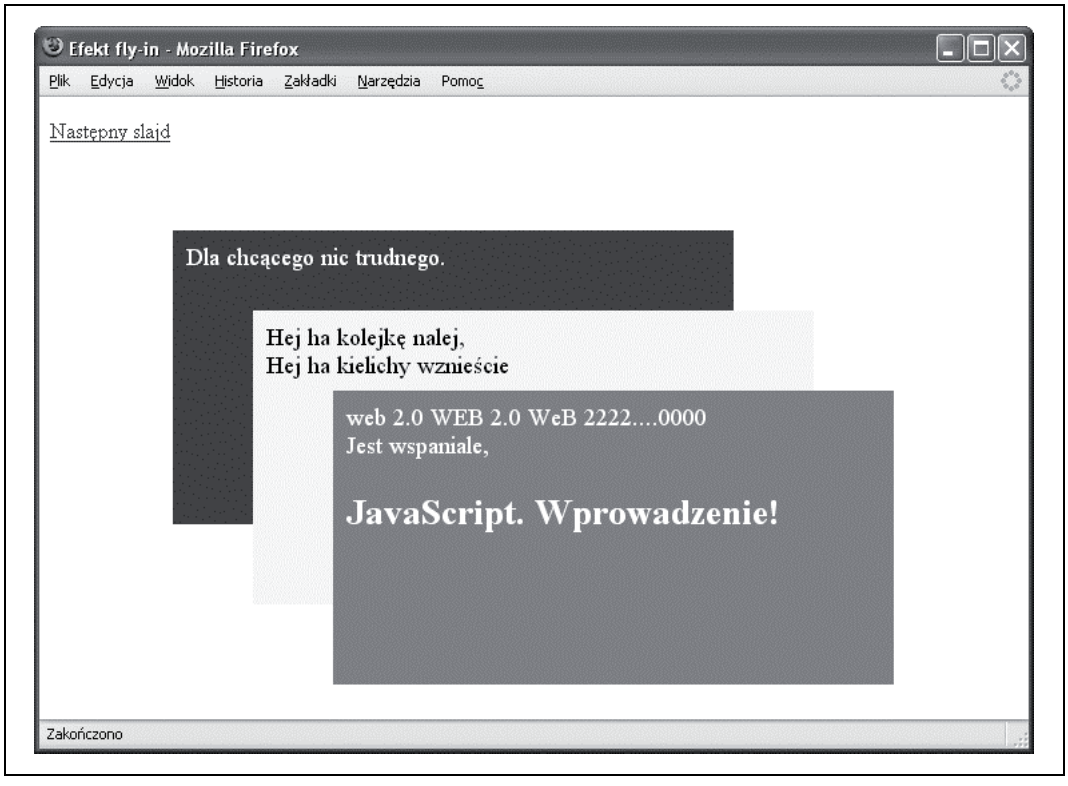
Listing 12.4 demonstruje taki efekt w oparciu o trzy elementy, które zlatują z góry po lewej stronie. Wykorzystano licznik czasu, który tworzy ruch i jest ustawiany ponownie, dopóki `x`, górna wartość, nie będzie większa niż `value` (200 + `value` × numer elementu, by utworzyć nakładanie się na siebie). Elementy są ukryte, kiedy znajdują się w pozycji początkowej poza stroną, na lewo i u góry, ponieważ ustawienie elementów poza stroną po prawej bądź na dole powoduje, że do strony dodawane są paski przewijania.

Listing 12.4. Pozycjonowanie elementów oraz ich ruch w oparciu o efekt fly-in

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Efekt fly-in</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <style type="text/css">

         div { padding: 10px; }

         #div1 { background-color: #00f;
            color: #fff;
            font-size: larger;
            position: absolute;
            width: 400px;
            height: 200px;
            left: -410px;
            top: -400px;
         }
         #div2 { background-color: #ff0;
            color: #;
            font-size: larger;
            position: absolute;
            width: 400px;
            height: 200px;
            left: -410px;
            top: -400px;
```

```
        }
        #div3 { background-color: #f00;
           color: #fff;
           font-size: larger;
           position: absolute;
           width: 400px;
           height: 200px;
           left: -410px;
           top: -400px;
        }
     </style>
     <script type="text/javascript">
        //<![CDATA[

        var element = ["div1","div2","div3"];

        function next() {
           setTimeout("moveBlock()",1000);
        }

        var x = 0;
        var y = 0;
        var elem = 0;
        function moveBlock() {
           x+=20;
           y+=20;
           var obj = document.getElementById(element[elem]);
           obj.style.top = x + "px";
           obj.style.left = y + "px";
           if (x < (100 + elem * 60)) {
              setTimeout("moveBlock()", 100);
           } else {
              elem++;
              x = 0; y = 0;
           }
        }
        //]]>
     </script>

  </head>
  <body>
     <p>
        <a href="javascript:next();">Następny slajd</a>
     </p>
     <div id="div1">
        Dla chcącego nic trudnego.
     </div>
     <div id="div2">
        Hej ha kolejkę nalej,<br />
        Hej ha kielichy wznieście
     </div>
     <div id="div3">
        web 2.0 WEB 2.0 WeB 2222....0000<br />
        Jest wspaniale,
        <h2>JavaScript. Wprowadzenie!</h2>
     </div>
  </body>
</html>
```

Tekst z przykładów jest nieco bezsensowny, jednak po wprowadzeniu drobnych ulepszeń do projektu i wykorzystaniu nieco bardziej odpowiedniej treści może stanowić efektywną technikę prezentacyjną. Na rysunku 12.2 zaprezentowano zrzut ekranu przedstawiający tę stronę otwartą w przeglądarce Firefox.



Rysunek 12.2. Strona z efektem fly-in

By strona była bardziej dostępna, łącze można zmienić tak, by otwierało strony z informacjami z fly-in. Alternatywnie, wszystkie trzy bloki z informacjami można wypozycjonować na stronie, natomiast znacznik `script` może zostać wykorzystany do ukrycia ich tylko wtedy, gdy obsługa JavaScriptu jest włączona.

Inną popularną techniką DHTML powiązaną z ruchem jest śledzenie ruchów użytkownika strony, który układa elementy na stronie. Technika ta nazywana jest **przeciągnij i upuść** (ang. *drag and drop*) i omówiona została w kolejnym podrozdziale.

Przeciągnij i upuść

Jednym z elementów DHTML, który wywołał duże zainteresowanie, kiedy pierwszy raz został przedstawiony, było przeciąganie i upuszczanie. Wszędzie pojawiły się przykłady zastosowania tego rozwiązania w koszykach z zakupami; ja także utworzyłam kilka takich rozwiązań. Udało mi się stworzyć nawet grę opartą na przeciąganiu i upuszczaniu.

Z czasem jednak okazało się, że zainteresowanie tą techniką przestało na dużą skalę objawiać się w aplikacjach. Rzadko można spotkać program wykorzystujący to rozwiązanie, a kiedy tak się zdarza, użytkownik często jest tym zirytowany. Dlaczego? Nie zawsze przeciąganie i upuszczanie jest takie proste, szczególnie kiedy wykorzystuje się trackpad bądź przeglądarkę obsługiwaną głosowo.

Tym, co ponownie obudziło zainteresowanie przeciąganiem i upuszczaniem, jest Google Maps i technika pozwalająca na przesuwanie mapy wewnątrz ograniczonej przestrzeni. To w gruncie rzeczy pierwszy przypadek, kiedy można było zobaczyć naprawdę efektywne użycie przeciągania i upuszczania. Google Maps i powiązane z nim API zostaną omówione w rozdziale 13., jednak póki co można przyjrzeć się implementacji własnej, niewielkiej emulacji technologii przeciągania i upuszczania.

Tym, co czyni podejście Google Maps naprawdę ekscytującym, jest fakt, iż wraz z przewijaniem mapy, aplikacja naprawdę pobiera kolejne elementy z serwera i integruje je ze stroną za pomocą mechanizmu pamięci podręcznej. Dzięki temu wydaje się, że nigdy nie osiągnie się końca mapy. Jest to zrobione naprawdę dobrze.

Na listingu 12.5 tworzony jest element `DIV`, a następnie jeden ze zrzutów ekranu z pierwszego rozdziału książki osadzany jest wewnątrz elementu. Oprócz przeciągania i upuszczania wykorzystano również atrybut `overflow`. Zostanie on użyty także w dalszej części książki, jednak na razie element `DIV` jest ustawiany tak, by chował bądź odcinał przepełnienie z treści elementu. Zapobiega to nakładaniu się obrazka poza wyznaczoną przestrzeń.

Listing 12.5. Efekt Google Maps — przeciąganie i upuszczanie obiektu w pojemniku

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Efekt Google Maps</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <style type="text/css">
         #div1 {
            overflow: hidden;
            position: absolute;
            top: 100px;
            left: 100px;
            border: 5px solid #000;
            width: 400px;
            height: 200px;
         }
         img {
            border: 1px solid #000;
         }
      </style>

      <script type="text/javascript">
         //<![CDATA[

         // zmienne globalne
         var dragObject  = null;
         var mouseOffset = null;

         // przechwycenie zdarzeń myszy
         document.onmousemove = mouseMove;
         document.onmouseup   = mouseUp;

         // utworzenie punktu myszy
         function mousePoint(x,y) {
            this.x = x;
            this.y = y;
         }
```

```
            // odnalezienie pozycji myszy
            function mousePosition(evnt){
               var x = parseInt(evnt.clientX);
               var y = parseInt(evnt.clientY);
               return new mousePoint(x,y);
            }

            // otrzymanie pozycji przesunięcia elementu wewnątrz strony
            function getMouseOffset(target, evnt){
               evnt = evnt || window.event;
               var mousePos  = mousePosition(evnt);

               var x = mousePos.x - target.offsetLeft;
               var y = mousePos.y - target.offsetTop;
               return new mousePoint(x,y);
            }

            // wyłączenie przeciągania
            function mouseUp(evnt){
               dragObject = null;
            }

            // przechwycenie ruchu myszy, tylko przy przeciąganiu
            function mouseMove(evnt){
               if (!dragObject) return;
               evnt = evnt || window.event;
               var mousePos = mousePosition(evnt);

               // jeśli da się przeciągnąć, ustawienie nowej pozycji bezwzględnej
               if(dragObject){
                  dragObject.style.position = 'absolute';

                  dragObject.style.top      = mousePos.y - mouseOffset.y + "px";
                  dragObject.style.left     = mousePos.x - mouseOffset.x + "px";
                  return false;
               }
            }

            // uczynienie elementu dającym się przeciągnąć
            function makeDraggable(item){
               if (item) {
                  item = document.getElementById(item);
                  item.onmousedown = function(evnt) {
                     dragObject  = this;
                     mouseOffset = getMouseOffset(this, evnt);
                     return false;
                  };
               }
            }

            //]]>
         </script>
      </head>
      <body onload="makeDraggable('img1');">
         <div id="div1" >
            <img id="img1" src="Rysunek1.1.jpg" />
         </div>
      </body>
   </html>
```

Jest to najbardziej skomplikowany przykład przedstawiony dotychczas w książce, dlatego warto omówić kod w JavaScripcie po kolei, od góry do dołu:

- Na początku tworzone są dwa obiekty globalne — `dragObject` oraz `mouseOffset`. Pierwszy z nich jest obiektem przeciąganym, natomiast drugi — wartością przesunięcia obiektu. Przesunięcie jest pozycją obiektu względem pojemnika — w tym przypadku strony. Przechwytuje się również zdarzenia `mousemove` oraz `mouseup` dla dokumentu i przypisuje je do programów obsługi zdarzeń — `mouseMove` oraz `mouseUp`.
- Później pojawia się obiekt `mousePoint`. Służy on do opakowania dwóch współrzędnych myszy — „x" oraz „y". Utworzenie obiektu pozwala na łatwiejsze przekazywanie obu wartości.
- Kolejną funkcją jest `mousePosition`. Uzyskuje ona dostęp do docelowych wartości `clientX` oraz `clientY` obiektu i zwraca obiekt `mousePoint` reprezentujący lokalizację „x" oraz „y" względem obszaru okna klienta z wyłączeniem ramki. Funkcja `parseInt` zapewnia, że wartości zwracane są w postaci liczbowej.
- Następnie pojawia się funkcja `getMouseOffset`, która przyjmuje jako parametry cel obiektu oraz zdarzenie `event`. Po normalizacji obiektu zdarzenia w stosunku do różnych przeglądarek pozycja zdarzenia myszy ustawiana jest na omówiony właśnie obiekt, `mousePoint`. Konieczne jest wprowadzenie poprawki w oparciu o właściwości `offsetLeft` oraz `offsetTop` obiektu. Gdyby nie wykonano tych obliczeń, obiekt przesuwałby się wraz z myszą, jednak prawdopodobnie ruch byłby dziwny i szarpany, a obiekt zdawałby się unosić ponad, pod bądź po boku kursora myszy. Po normalizacji jest on wykorzystywany do utworzenia znormalizowanego obiektu `mousePoint`, który następnie zwracany jest z obiektu.
- Kolejną funkcją jest `mouseUp`, która wyłącza przeciąganie poprzez ustawienie `dragObject` na `null`. Następnie pojawia się funkcja `mouseMove`, w której odbywa się większość obliczeń związanych z przeciąganiem. Jeśli obiekt przeciągający nie zostanie ustawiony, funkcja ta jest kończona. W przeciwnym wypadku odnajdywana jest znormalizowana pozycja myszy, pozycjonowanie obiektu ustawiane jest na `absolute` i ustawiane są jego właściwości `left` oraz `top` (po dostosowaniu o przesunięcie).
- Ostatnią funkcją jest `makeDraggable`, która czyni obiekt przekazywany do funkcji zdatnym do przeciągania. Oznacza to dodanie do zdarzenia `mousedown` obiektu funkcji, która ustawia obiekt przeciągania na obiekt i otrzymuje jego wartość przesunięcia.

Wydaje się, że kodu jest sporo, jednak w rzeczywistości kod ten jest o wiele mniej skomplikowany niż kiedyś, w starych przeglądarkach, ponieważ większość nowoczesnych przeglądarek wykorzystuje te same właściwości, jeśli chodzi o pozycjonowanie. Chwała za to, bo operacje przeciągania i upuszczania są wystarczająco skomplikowane nawet bez tego dodatkowego problemu. Ponownie w Google Maps dodano dodatkowy element poprzez wykorzystanie Ajaksa do ciągłego odświeżania mapy, by nigdy się ona nie skończyła. Wykracza to jednak poza tematykę niniejszej książki. Można to rozważyć jako dodatkowe wyzwanie na przyszłość.

Rozmiar i przycinanie

Rozmiar elementu kontrolowany jest przez zbiór sześciu atrybutów CSS. Najczęściej wykorzystywane są pierwsze dwa — `width` oraz `height` — które służą do ustawiania bezwzględnej szerokości oraz wysokości elementu. Pozostałe cztery — `min-height`, `min-width`, `max-height` oraz `max-width` — są przydatnymi atrybutami CSS (w szczególności w przypadku pracy z obrazkami), jednak nie są zbyt często używane w pracy z efektami dynamicznymi.

Tak naprawdę na szerokość i wysokość elementu składa się kilka czynników związanych z różnymi atrybutami, w tym `border`, `margin`, `padding` oraz `content`. Wszystkie one połączone są w tak zwany *box model* CSS, powiązany z elementami bloku — elementami, które wymuszają złamanie wiersza przed i po elemencie. Więcej informacji na ten temat znajduje się na stronie W3C *Box Model* pod adresem: *http://www.w3.org/TR/REC-CSS2/box.html*.

Jeśli treść elementu jest zbyt długa dla ustawień elementu, przepełnieniem zarządza atrybut CSS `overflow`, który może zostać ustawiony na `visible` (renderuje całą treść i przepełnia granice elementu), `hidden` (przycina treść), `scroll` (przycina treść i udostępnia paski przewijania) oraz `auto` (przycinanie i paski przewijania występują tylko wtedy, gdy treść jest ukryta).

Po co w ogóle ustawiać szerokość elementu? W końcu jeśli wysokość nie zostanie zdefiniowana, a przepełnienie nie będzie ustawione na `clip`, rozmiar elementu zostanie automatycznie zmieniony, by pasował do treści.

Jeśli treść mieści się w dwóch kolumnach położonych obok siebie, można chcieć ustawić ich wysokość po to, by jedna nie była znacznie dłuższa od drugiej.

Przepełnienie i treść dynamiczna

Kiedy treść elementu jest zastępowana dynamicznie — albo poprzez wywołanie Ajax, albo jakieś inne zdarzenie — dopasowanie treści wewnątrz elementu może się dramatycznie zmienić. Jednym z rozwiązań zapewniających, że treść zawsze będzie dostępna, jest ustawienie wartości `overflow` na `auto`. Jeśli treść będzie zbyt duża, udostępnione zostaną paski przewijania. Na listingu 12.6 znajdują się dwa bloki — jeden z nich zawiera dużą ilość tekstu, a drugi niewielką. Wymiary obu elementów ustawiane są tak, by bezpiecznie przechowywać swoją zawartość, kiedy strona jest ładowana. Pojawia się jednak odnośnik, który służy do przełączania treści: dłuższej z pierwszego bloku na krótszą z drugiego i odwrotnie. W CSS wartość `overflow` dla drugiego elementu ustawiana jest na `auto`.

Listing 12.6. Zmiana treści i wpływ ustawienia overflow

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Przepełnienie</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <style type="text/css">
         #div1 {
            width: 700px;
            height: 150px;
         }
         #div2 {
            width: 600px;
            height: 100px;
            overflow: auto;
         }
      </style>

      <script type="text/javascript">
         //<![CDATA[
```

```
            function switchContent() {
               var div1 = document.getElementById("div1").innerHTML;
               var div2 = document.getElementById("div2").innerHTML;
               document.getElementById("div1").innerHTML = div2;
               document.getElementById("div2").innerHTML = div1;
            }

            //]]>
         </script>

      </head>
      <body>
         <p>
            <a href="javascript:switchContent();">Switch</a>
         </p>
         <div id="div1">
            <p>Jednym z pierwszych elementów HTML charakterystycznych dla prezentacji był
            element font. Jest on również jednym ze starszych elementów HTML, które nadal
            zbyt często można odnaleźć na stronach internetowych. Nie powinno być
            zaskoczeniem, że właśnie właściwości font oraz text były tak interesujące
            dla budowania stron internetowych. Niewiele zmian, które można wprowadzić
            w atrybutach style elementu, może przynieść taki efekt jak zmiany text bądź
            font.</p>
            <p>Warto zauważyć, że mowa jest o właściwościach font oraz text. Właściwość
            font wiąże się z samymi znakami: ich rodziną, rozmiarem, typem i innymi
            cechami ich wyglądu. Atrybuty text mają więcej wspólnego z dekoracją
            dołączoną do tekstu, jego położeniem i tak dalej.</p>
         </div>
         <div id="div2">
            <p>Mniejszy element.</p>
         </div>
      </body>
   </html>
```

Kiedy treść jest przełączana, pierwszy blok zawiera niewiele tekstu i dużą liczbę białych znaków. Jedynym sposobem na zmianę tej sytuacji jest korekta wymiarów tego elementu `DIV`. Niestety, w prawdziwym świecie ustalenie odpowiedniego rozmiaru dla nowej treści nie jest takie proste.

Drugi element `DIV` niespodziewanie zawiera po prawej stronie pasek przewijania, który pozwala na przewijanie treści. Zamiast próbować zmieniać rozmiar poprzez zgadywanie, lepszym rozwiązaniem wydaje się być ustawienie wartości `overflow` na `auto` i tym samym wywołanie paska przewijania. W ten sposób blok ten jest na stronie relatywnie stabilny, nie spycha ciągle innych elementów, nie pojawiają się wielkie bloki wypełnione białymi znakami, a treść jest nadal dostępna.

Innym podejściem do radzenia sobie ze zmieniającą się treścią jest zmiana rozmiaru bloku za pomocą właściwości tylko do odczytu `offsetWidth` oraz `offsetHeight` w celu ustalenia prawdziwej wielkości treści. Istnieją jednak różnice pomiędzy przeglądarkami w implementacji tych właściwości. Internet Explorer uwzględnia w rozmiarze bloku również ramkę oraz dopełnienie, natomiast Mozilla/Firefox bierze pod uwagę jedynie rozmiar niezbędny dla samej treści.

Dostęp do obliczonej wysokości oraz szerokości elementu można uzyskać za pomocą zdefiniowanej wcześniej metody `getStyle`, wykorzystując `width` oraz `height` w miejsce `backgroundColor`.

Choć `width` oraz `height` kontrolują rozmiar elementu, nie zawsze mają wpływ na to, co w elemencie jest widoczne. Tę kwestię można kontrolować za pomocą prostokąta odcinającego powiązanego z elementem.

Prostokąt odcinający

Według konsorcjum W3C obszar odcinający (ang. *clipping region*):

> […] definiuje, jaka część treści wyświetlanego elementu jest widoczna. Domyślnie obszar odcinający ma ten sam rozmiar i kształt co ramki elementów. Obszar ten można jednak zmodyfikować za pomocą właściwości `clip` (z dokumentu *Visual Effects* konsorcjum W3C dostępnego pod adresem: *http://www.w3.org/TR/REC-CSS2/visufx.html*).

Właściwość CSS `clip` określa kształt oraz wymiary tego kształtu. Obecnie jedynym obsługiwanym kształtem jest prostokąt, oznaczony `rect`, definiowany przez cztery wymiary: górny, prawy, dolny oraz lewy.

```
clip: rect(topval, rightval, bottomval, leftval);
```

Obszar odcinający określa, ile treści elementu będzie wyświetlone. Wymaga również, by atrybut `position` ustawiony został na `absolute`.

Jeśli element ma 200 pikseli szerokości i 300 pikseli wysokości, obszar odcinający `rect(0px, 200px, 300px, 0px)` nie odcina żadnego fragmentu bloku — oczywiście jest to uzależnione od tego, czy element posiada ramkę bądź inne ustawienie, które może zmienić jego efektywną wysokość czy szerokość. Obszar odcinający `rect(10px, 190px, 290px, 10px)` odcina 10 pikseli z każdej strony. Warto zauważyć, że inkrementowanie wartości dla góry oraz lewej strony, a dekrementowanie jej dla dołu i prawej strony powoduje odcinanie.

Z perspektywy DHTML odcinanie można wykorzystać do utworzenia jakiegoś efektu związanego z przewijaniem, obojętnie czy jest on związany z ruchem elementu czy też nie. Może ono również zostać użyte do utworzenia nowego „efektu akordeonu”, który stał się tak popularny (zademonstrowanego w rozdziale 14.).

Na listingu 12.7 zaprezentowano proste zastosowanie odcinania do utworzenia animowanego elementu zjeżdżającego na dół. Kliknięcie nagłówka elementu albo go rozszerza, albo zwija, w zależności od jego aktualnego stanu. Do animacji wykorzystano licznik czasu. Możliwe jest również ustawienie pełnego wyświetlenia bądź ukrycia z każdym kliknięciem i pominięcie licznika.

Listing 12.7. Zjeżdżająca na dół animacja utworzona przy użyciu licznika czasu oraz odcinania

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN""http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Proste rozwijanie z odcinaniem</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <style type="text/css">

         #data1 {
            position: absolute;
            top: 100px; left: 100px;
            padding: 0;
            width: 200px;
            height: 200px;
            background-color: #ff0;
```

```
            clip: rect(0px,200px,200px,0px);
         }

         #data1 h3 {
            margin: 0;
            padding: 5px;
            font-size: smaller;
            background-color: #006;
            color: #fff;
         }

         #contained {
            margin: 10px
         }

      </style>

      <script type="text/javascript">
         //<![CDATA[

         var bottom = 200;
         var hidden = false;
         var obj = null;
         function clipItem() {
            obj = document.getElementById("data1");
            if (hidden) {
               showItem();
            } else {
               hideItem();
            }
         }

         function hideItem() {
            bottom-=20;
            var clip = "rect(0px,200px," + bottom + "px,0px)";
            obj.style.clip = clip;
            if (bottom == 20) {
               hidden=true;
            } else {
               setTimeout("hideItem()",100);
            }
         }

         function showItem() {
            bottom+=20;
            var clip = "rect(0px,200px," + bottom + "px,0px)";
            obj.style.clip=clip;
            if (bottom == 200) {
               hidden=false;
            } else {
               setTimeout("showItem()",100);
            }
         }

         //]]>
      </script>
   </head>
   <body>
      <div id="data1">
         <h3 onclick="clipItem();">Kliknij, by rozszerzyć bądź zwinąć</h3>
         <div id="contained">
            Ten tekst jest zawarty wewnątrz bloku div.
         </div>
      </div>
   </body>
</html>
```

Warto zauważyć, że zamiast otrzymywać wartość odcinania bezpośrednio z właściwości `style` w celu sprawdzania stanu, skorzystano ze zmiennej globalnej. Takie rozwiązanie będzie się w DHTML wykorzystywać tak często, jak to możliwe — zmienna `get` jest tańsza od obiektu `get`, w szczególności takiego, który musi działać w różnych przeglądarkach.

Wyświetlanie, widoczność oraz przezroczystość

Ciekawą kwestią związaną z elementami strony internetowej jest to, że mogą one być całkowicie przezroczyste i niewidoczne, jednak nadal będą wpływać na układ strony. Przyczyną tego jest fakt, iż niewidoczność i przezroczystość oraz wyświetlanie i jego brak nie są w CSS tym samym.

Element może zostać ukryty poprzez ustawienie `visibility` na `hidden`, a pokazany poprzez ustawienie `visibility` na `visible`. Właściwość ta może również zostać ustawiona na `inherit`, a element odziedziczy ustawienie właściwości `visibility` po elemencie go zawierającym.

Jak zademonstrowano to rozdziale 11., możliwa jest również zmiana przezroczystości elementu, dopóki nie będzie on całkowicie przezroczysty i niewidoczny. Jednak tak jak w przypadku właściwości `visibility` element nadal zachowuje swoją pozycję na stronie.

Jeśli właściwość `display` elementu ustawiona jest na `none`, element pozostanie również ukryty. Usuwany jest również wpływ, jaki ten element ma na układ strony. By element uczynić widocznym, ma się kilka możliwości. Może on stać się `visible` i może działać jak element na poziomie bloku (złamanie wiersza przed i po tym elemencie) poprzez ustawienie jego właściwości display na `block`. Jeśli takie zachowanie nie jest pożądane, można ustawić `display` na `inline`, dzięki czemu element będzie wyświetlany na miejscu, a nie jako blok.

Dodatkowo można wyświetlić element za pomocą domyślnych ustawień wyświetlania niektórych elementów HTML, takich jak `inline-block`, `table`, `table-cell`, `list-item`, `compact`, `run-in` i tak dalej. Jest to atrybut o potężnych możliwościach, którym warto się nieco pobawić, aż nie osiągnie się zadowalających rezultatów.

Właściwe narzędzie dla właściwego efektu

Gdy ma się do dyspozycji te wszystkie sposoby ukrywania i wyświetlania elementów, jaka metoda powinna zostać wykorzystana dla danego efektu?

Jeśli element jest pozycjonowany bezwzględnie, a następnie ukrywany i pokazywany w oparciu o jakieś zdarzenie, takie jak kliknięcie myszą bądź wysłanie formularza, należy skorzystać z właściwości `visibility`. Jest ona prosta i łatwa w użyciu, a element pozycjonowany bezwzględnie i tak jest usuwany z układu strony. W przypadku pomocy podręcznej warto zatem używać `visibility`.

Jeśli ukryta treść powinna spychać na dół elementy strony znajdujące się po niej, kiedy jest ona wyświetlana, na przykład w sytuacji kliknięcia zwiniętej listy opcji przy wypełnianiu formularza, należy skorzystać z `display` i przełączać pomiędzy wartościami `none` oraz `block`. Należy użyć `display` do ukrywania i pokazywania pól formularza w celu otrzymania informacji zwrotnych od użytkownika.

Jeśli tworzy się efekt blaknięcia bądź chce się zmniejszyć widoczność jakiegoś elementu, należy skorzystać z właściwości `opacity`. Można ją ustawić tak, by element stawał się całkowicie przezroczysty, zazwyczaj po animowanym blaknięciu trwającym jakiś czas. Należy wykorzystać `opacity` do podkreślania czegoś i dostarczania informacji wizualnych. Właściwość ta może również służyć do sygnalizowania przejścia, jak zademonstrowano w pokazie slajdów zaprezentowanym w rozdziale 11.

Ważna kwestia związana z efektami wizualnymi, które służą celom informacyjnym: efekty te powinny zawsze zawierać jakiś element tekstowy, by osoby używające przeglądarek niewizualnych (bądź takich z ograniczonymi możliwościami wizualnymi) otrzymywały tyle samo informacji. Nigdy nie należy polegać w całości na efekcie wizualnym dla celów otrzymania informacji zwrotnych od użytkownika.

Czas na mały pokaz działania.

Informacje w locie

Jedna z najlepszych stron, jakie znam, udostępnia jakąś formę pomocy za każdym razem, kiedy użytkownik strony pytany jest o jakieś informacje. Nawet jeśli pyta się o jego imię, można przedstawić wyjaśnienie polityki prywatności oraz sposobu wykorzystania tych danych.

Możliwe jest udostępnienie pomocy w formie wskazówek typu *tooltip* poprzez ustawienie atrybutu `title` łącza otaczającego podpis pola, jednak zazwyczaj ogranicza to ilość informacji, jakie można przekazać. Można również wyświetlić wyskakujące okno z informacją, co jest szczególnie przydatne, gdy informacja ta jest długa oraz szczegółowa i zawiera opis opcji. Z kolei w przypadkach znajdujących się pomiędzy tymi dwoma, kiedy ma się więcej informacji niż tylko trochę, a mniej niż bardzo dużo, miło byłoby móc je zamieścić bezpośrednio na stronie.

Zazwyczaj jednak formularze zajmują większość przestrzeni, a duża ilość tekstu może sprawić, że strona będzie wyglądała na zaśmieconą. Jednym z możliwych rozwiązań jest umieszczenie informacji na stronie, jednak pokazanie ich w oparciu o jakieś zdarzenie.

Jest to jeden z bardziej użytecznych efektów, jakie można uzyskać za pomocą DHTML a przy okazji również jeden z najłatwiejszych. Na listingu 12.8 zaprezentowano stronę zawierającą dwa elementy formularza, a każdy z nich posiada ukryty blok pomocy. W skrypcie, kiedy kliknięty zostanie podpis elementu, jeśli jakaś pomoc dla tego elementu jest już pokazywana, zostanie ona schowana, a pokazany będzie nowy blok z pomocą.

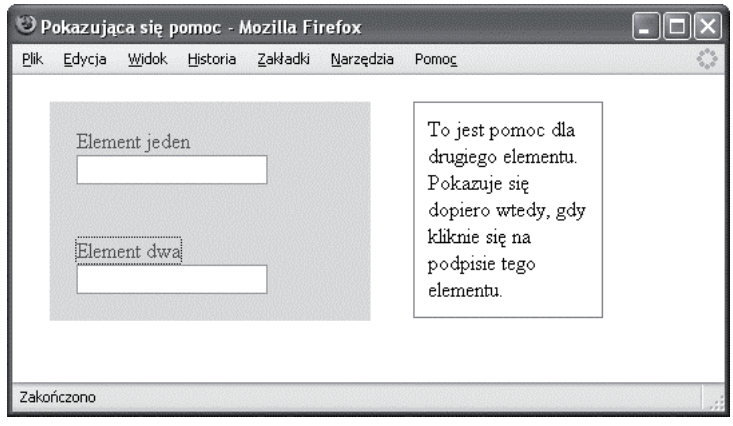
Listing 12.8. Wykorzystywanie ukrytych pól z pomocą

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Pokazująca się pomoc</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <style type="text/css">

         .help {
            position: absolute;
            left: 300px;
            top: 20px;
```

```
            visibility: hidden;
            width: 120px;
            padding: 10px;
            border: 1px solid #f00;
         }

         form {
            margin: 20px;
            background-color: #DFE1CB;
            padding: 20px;
            width: 200px;
         }

         form a {
            color: #060;
            text-decoration: none;
         }

         form a:hover {
            cursor: help;
         }
      </style>

      <script type="text/javascript">
         //<![CDATA[

         var item = null;

         function showHelp(newItem) {
            if (item) {
               item.style.visibility='hidden';
            }
            item = document.getElementById(newItem);
            item.style.visibility='visible';
         }

         //]]>
      </script>
   </head>
   <body>
      <form>
         <label><a href="javascript:showHelp('item1')" alt="pomoc">Element jeden</a>
         </label>
         <input type="text"><br /><br /><br />
         <label><a href="javascript:showHelp('item2')" alt="pomoc">Element dwa</a><
         /label>
         <input type="text">
      </form>
      <div id="item1" class="help">
         To jest pomoc dla pierwszego elementu. Pokazuje się dopiero wtedy, gdy
         kliknie się podpis tego elementu.
      </div>
      <div id="item2" class="help">
         To jest pomoc dla drugiego elementu. Pokazuje się dopiero wtedy, gdy kliknie
         się podpis tego elementu.
      </div>
   </body>
</html>
```

Do strony dodano także nieco CSS, żeby lepiej wyglądała. Formularz posiada kolorowe tło, blok pomocy jest otoczony czerwoną ramką, a kiedy kursor myszy znajdzie się nad podpisem każdego z pól formularza, ikona kursora zostaje zamieniona na ikonę `help`. Zazwyczaj wygląda ona jak strzałka z niewielkim znakiem zapytania bądź też jak sam znak zapytania.

Jest to prosty, ale efektowny sposób dostarczenia wskazówki użytkownikowi strony — podobnie jak znacznik `alt` z napisem `pomoc`. Ten ukryty system pomocy przedstawiono na rysunku 12.3.



Rysunek 12.3. System pomocy wykorzystujący właściwość visibility

Formularze zwijane

Konieczność rozbijania funkcjonalności formularzy na wiele stron jest bolesna, jednak strona zawierająca zbyt dużo elementów formularzy wyświetlonych jednocześnie może być nieczytelna.

Dodatkowo na popularności zyskuje bezpośrednia edycja danych. Działa to tak, że tytuły dla części z danymi aktywowane są dla właściciela danych, a ich kliknięcie otwiera formularz lub pola do wpisywania, w których dane te mogą być zmienione.

Obie sytuacje z dużym prawdopodobieństwem mogą korzystać z **formularzy zwijanych** (ang. *collapsible forms*). Są to formularze lub ich części, które są ukryte na stronie. Wyświetlane są dopiero wtedy, gdy coś zostanie aktywowane. I nie tylko są wyświetlane — spychają również inne dane z drogi, zajmując to samo miejsce, które formularz normalnie by zajmował, gdyby został wyświetlony w całości.

Google, Flickr oraz wiele różnych firm wykorzystuje ten typ zwijanych treści. Biorąc pod uwagę fakt, iż jest to również jedno z najprostszych rozwiązań, nie powinno to stanowić zaskoczenia. Gdyby obsługa JavaScriptu była wyłączona, obsługa zdarzeń powiązanych z tytułami, które normalnie wyświetlają treść, nie byłaby aktywna, a formularze nie otworzyłyby się. Możliwe jest dodanie elementów menu, które będą otwierać osobne strony formularza lub też będą wyświetlać znacznik `noscript`.

Ostatni przykład niniejszego rozdziału, zamieszczony na listingu 12.9, prezentuje formularz zwijany. W tym przypadku jest to zbiór ułożonych na stosie bloków elementów formularza. Kliknięcie podpisu każdego z nich chowa blok, jeśli jest on aktualnie wyświetlany, bądź pokazuje go, jeśli nie jest. W przeglądarkach bez obsługi JavaScriptu tytuły obu bloków otoczone są łączami hipertekstowymi. Kliknięcie łącza pozornie przenosi użytkownika do osobnego formularza statycznego. Dla stron z JavaScriptem zwrócenie wartości `false` dla zdarzenia

`onclick` z łącza anuluje jego domyślne zachowanie. Można to zaobserwować, kiedy wyłączy się obsługę JavaScriptu: kliknięcie łącza zmienia adres URL strony tak, by odzwierciedlał fragment URI (*#name* bądź *#address*). Jednak kiedy wykorzystywanie skryptów jest dozwolone, nie da się tego zobaczyć, a zamiast tego wyświetlony zostanie formularz.

Listing 12.9. Implementacja zwijanego formularza

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Zwijany formularz</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <style type="text/css">

         .label {
            background-color: #003;
            width: 400px;
            border-right: 1px solid #fff;
            padding: 10px;
            margin: 0 20px;
            color: #fff;
            text-align: center;
            border-bottom: 1px solid #fff;
         }

         .label a {
            color: #fff;
         }

         .elements {
            background-color: #CCD9FF;
            margin: 0 20px;
            padding: 10px;
            width: 400px;
            display: none;
         }
      </style>

      <script type="text/javascript">
         //<![CDATA[

         window.onload=setup;

         function setup() {
            document.getElementById('one').style.display='none';
            document.getElementById('two').style.display='none';
         }

         function show(newItem) {
            var item = document.getElementById(newItem);
            if (item.style.display=='none') {
               item.style.display='block';
            } else {
               item.style.display='none';
            }
         }

         //]]>
      </script>
   </head>
   <body>
```

```
        <form action="GET">
           <div class="label" onclick="show('one')">
              <a href="#name" onclick="return false">Dane osobowe</a>
           </div>
           <div class="elements" id="one">
              <label>Imię:</label><br /><input type="text" name="firstname" /><br /><br />
              <label>Nazwisko:</label><br /><input type="text" name="lastname" /><br /><br />
           </div>
           <div class="label" onclick="show('two')">
              <a href="#address" onclick="return false">Dane adresowe</a>
           </div>
           <div class="elements" id="two">
              <label>Ulica:</label><br /><input type="text" name="street" /><br /><br />
              <label>Miasto:</label><br /><input type="text" name="city" /><br /><br />
              <label>Państwo:</label><br /><input type="text" name="state" /><br /><br />
           </div>
        </form>

        <p>Inne dane bądź informacje.</p>
     </body>
  </html>
```

I znów jest to właśnie ten typ funkcjonalności, jaki chce się dodawać do własnych stron. Jest prosty, robi wrażenie i jest relatywnie łatwy w konwersji na wersję bez JavaScriptu, kiedy wyłączona jest obsługa skryptów.

Niniejszy rozdział przedstawił ledwie czubek góry lodowej, jeśli chodzi o to, co można osiągnąć za pomocą JavaScriptu i CSS. Powinien jednak stanowić dobry punkt wyjścia dla własnych pomysłów. W rozdziale 13. wprowadzono podstawy Ajaksa, a następnie omówiono łączenie Ajaksa i efektów DHTML w potężne aplikacje.

Pytania

1. Próbuje się uzyskać dostęp do koloru tekstu elementu w JavaScripcie za pomocą `obj.style.color`, jednak żadna wartość nie jest zwracana. Wiadomo jednak, że została ona ustawiona w arkuszu stylów. Dlaczego wartość ta nie jest zwracana i w jaki sposób można zmienić aplikację tak, by ją zwracała?
2. Mając tekst znajdujący się w bloku `DIV`, w jaki sposób można zmienić jego wyświetlanie na czcionkę `14pt` z czerwonym kolorem i odstępem pomiędzy wierszami równym `16pt`?
3. Jeśli powyższe nie działa, co mogło spowodować niepowodzenie?
4. Jakie są dwa sposoby sprawienia, by blok zniknął?
5. Jeśli przeciąganie i upuszczanie nie jest efektywną techniką dla koszyka z zakupami, jaki efekt DHTML byłby użyteczny dla tego typu usługi?

Odpowiedzi znajdują się w dodatku.