Buduj zaawansowane i interaktywne strony WWW!

JavaScript i jQuery

nieoficjalny podręcznik

David Sawyer McFarland

O'REILLY®

Helion

Tytuł oryginału: JavaScript & jQuery: The Missing Manual
Tłumaczenie: Piotr Rajca

ISBN: 978-83-246-4381-3

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/jsjqnp.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/jsjqnp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

10

ROZDZIAŁ

Rozbudowa interfejsu stron WWW

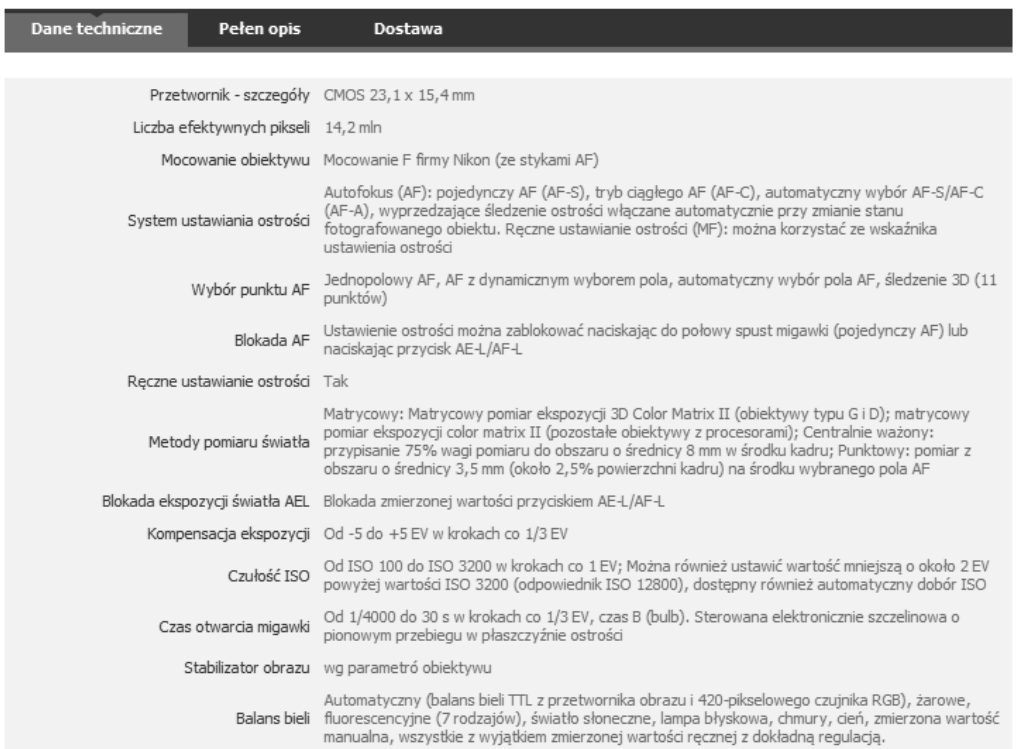
Bywa, że strony WWW przypominają długie, jednostronicowe broszury. Odwiedzające je osoby mogą się czuć przytłoczone przez wiele tekstu i znaczną liczbę obrazków, które muszą długo przewijać, zwłaszcza gdy nie są w stanie szybko znaleźć poszukiwanych informacji. To naszym zadaniem, zadaniem twórców stron jest zapewnienie użytkownikom narzędzi, które ułatwią im znalezienie tego, czego szukają. Przy użyciu JavaScriptu oraz biblioteki jQuery można usprawniać tworzone strony i ułatwiać użytkownikom korzystanie z nich — na przykład poprzez ukrywanie zawartości, aż do momentu gdy będzie potrzebna, oraz zapewnianie łatwiejszego dostępu do informacji.

W tym rozdziale poznasz techniki służące do poprawiania czytelności i łatwości korzystania ze stron WWW. Karty pozwalają na umieszczanie znacznych ilości informacji na stosunkowo niewielkim obszarze i zapewniają możliwość kliknięcia wybranej karty w celu uzyskania dostępu do mniejszej porcji danych. Etykietki ekranowe — niewielkie, wyskakujące okienka pokazywane po wskazaniu jakiegoś elementu strony wskaźnikiem myszy — umożliwiają wyświetlanie dodatkowych informacji. Coraz bardziej popularną formą kontroli zawartości strony są tak zwane slidery (od angielskiego słowa *slide* — przesuwać) — można by je porównać do okna, którego zawartość da się przesuwać, by ukryć jedne, a wyświetlić inne elementy tejże strony. Slidery pozwalają na prezentowanie znacznych ilości informacji i są bardzo często stosowane na stronach głównych witryn.

W tym rozdziale poznasz także kilka przydatnych technik pozwalających na tworzenie własnych komponentów interfejsu użytkownika, takich jak określanie wymiarów okna przeglądarki, konkretnego elementu strony oraz położenia elementu na stronie.

Organizowanie informacji przy użyciu kart

Umieszczenie na stronie zbyt wielu informacji może przytłoczyć użytkownika i sprawić, że strona będzie wyglądała na przepełnioną. Język JavaScript zapewnia wiele możliwości prezentowania znacznych ilości informacji na niewielkim obszarze. Jedną z technik jest stosowanie *kart*. Panel kart składa się z rzędu zakładek wyświetlonych u góry oraz jednej, widocznej karty. Kiedy użytkownik kliknie zakładkę, aktualnie prezentowana karta znika, a na jej miejscu pojawia się inna (patrz rysunek 10.1).



Rysunek 10.1. *Panele kart są często stosowane na witrynach zajmujących się handlem elektronicznym, na których informacje są prezentowane na osobnych kartach. W przedstawionym tu przykładzie (będącym jedynie fragmentem całej strony) przedstawiono karty zawierające opis produktu, jego specyfikację oraz informacje o sposobie dostawy; przy takim rozwiązaniu użytkownik może kliknąć kartę, by wyświetlić te informacje, które go interesują*

Panele kart, tak jak wszystkie komponenty interfejsu użytkownika stron WWW, są tworzone przy użyciu kodów napisanych w językach HTML, JavaScript oraz za pomocą CSS. Każdy z tych elementów paneli kart można pisać na wiele sposobów, poniżej przedstawione zostało bardzo proste rozwiązanie.

Kod HTML

Panel kart składa się z dwóch podstawowych elementów: zakładek (czyli przycisków umieszczonych jeden obok drugiego w wierszu wyświetlonym na górze lub na dole komponentu) oraz kart (będących znacznikami `<div>` zawierającymi informacje, które chcemy prezentować). Dodatkowo komponent może zawierać kilka innych znaczników służących do zapewnienia właściwej jego organizacji oraz mających na celu ułatwienie kodu JavaScript, który go obsługuje. Oto one.

- **Element pojemnika**. Choć właściwie nie jest to niezbędne, jednak zastosowanie dodatkowego znacznika `<div>`, w którym będą umieszczone wszystkie zakładki i karty, może w wyraźny sposób oznaczyć początek i koniec komponentu oraz ułatwić tworzenie kodu JavaScript zwłaszcza wtedy, kiedy na jednej stronie ma znaleźć się więcej takich paneli karty. Oto podstawowy kod HTML takiego elementu:

  ```
  <div class="tabbedPanels">

  </div>
  ```

  Dodanie do tego znacznika atrybutu `class` identyfikuje go i ułatwia tworzenie stylów określających postać elementów wewnątrz panelu oraz tworzenie selektorów jQuery odwołujących się do poszczególnych zakładek i kart. Jeśli na naszej stronie ma się znajdować tylko jeden panel kart, zamiast klasy można by określić identyfikator tego znacznika.

- **Zakładki**. Zazwyczaj tworzy się je w postaci listy wypunktowanej, zawierającej odnośniki:

  ```
  <ul class="tabs">
  <li><a href="#panel1">Informacje ogólne</a></li>
  <li><a href="#panel2">Specyfikacja</a></li>
  <li><a href="#panel3">Dostawa</a></li>
  </ul>
  ```

  Odnośniki umieszczone w poszczególnych punktach listy odwołują się do identyfikatorów przypisanych poszczególnym kartom (opisanym poniżej). Utworzenie odnośnika od karty sprawia, że użytkownicy, którzy wyłączyli w swoich przeglądarkach obsługę języka JavaScript, będą mogli przeskoczyć prosto od wybranego fragmentu treści — kliknięcie takiego odnośnika powoduje przewinięcie strony do określonego miejsca.

Uwaga: Jeśli nie wiesz, jak tworzy się takie odnośniki, krótkie wyjaśnienie możesz znaleźć na stronie *http://www.yourhtmlsource.com/text/internallinks.html*.

- **Pojemnik kart**. Znacznik `<div>` zawierający wszystkie karty może przydać się do określania ich postaci w stylach CSS oraz odwoływania do nich przy użyciu jQuery:

  ```
  <div class="panelContainer">

  </div>
  ```

- **Karty**. To właśnie w nich umieszczane są właściwe informacje. Każda karta jest reprezentowana przez znacznik `<div>` i może zawierać dowolne treści: nagłówki, akapity, obrazki oraz inne znaczniki `<div>`. Każdy z tych znaczników powinien mieć unikalny identyfikator, odpowiadający identyfikatorowi podanemu w atrybucie `HREF` odnośników tworzących zakładki (patrz drugi punkt listy):

```
<div class="panel" id="panel1">
  <!-- tu będzie treść karty -->
</div>
<div class="panel" id="panel2">
  <!-- tu będzie treść karty -->
</div>
<div class="panel" id="panel3">
  <!-- tu będzie treść karty -->
</div>
```

  Dodanie do każdego z tych znaczników jakiejś klasy — na przykład `class=` ↪`"panel"` — także jest dobrym pomysłem, gdyż zapewnia dodatkowy sposób określania ich wyglądu i pobierania przy użyciu jQuery.

Wszystkie znaczniki `<div>` poszczególnych kart są umieszczane wewnątrz nadrzędnego znacznika `<div>` pełniącego rolę pojemnika. Kompletna struktura kodu HTML panelu kart ma zatem następującą postać:

```
<div class="tabbedPanels">
   <ul class="tabs">
      <li><a href="#panel1">Informacje ogólne</a></li>
      <li><a href="#panel2">Specyfikacja</a></li>
      <li><a href="#panel3">Dostawa</a></li>
   </ul>
   <div class="panelContainer">
      <div class="panel" id="panel1">
         <!-- tu będzie treść karty -->
      </div>
      <div class="panel" id="panel2">
         <!-- tu będzie treść karty -->
      </div>
      <div class="panel" id="panel3">
         <!-- tu będzie treść karty -->
      </div>
   </div>
</div>
```

Kod CSS

Arkusz stylów pozwoli nadać nagłówkom kart wygląd zakładek (przycisków umieszczonych tuż obok siebie), a także sprawi, że same karty będą wyglądały jako spójna całość, w której treść będzie się łączyć z zakładką.

- **Pojemnik**. Nie musimy w żaden sposób określać wyglądu znacznika `<div>`, wewnątrz którego są umieszczone wszystkie karty (w rzeczywistości ten znacznik w ogóle nie jest potrzebny). Jednak może się przydać, gdybyśmy chcieli ograniczyć szerokość całego panelu na przykład po to, by umieścić go obok jakiegoś innego elementu strony bądź umiejscowić obok siebie dwa takie panele. W takim przypadku w stylu odnoszącym się do tego znacznika moglibyśmy określić jego szerokość w następujący sposób:

```
.tabbedPanels {
  width: 50%
}
```

- **Lista wypunktowana oraz jej elementy**. Ponieważ listy wypunktowane są zazwyczaj nieco wcięte, zatem musimy z niej usunąć wszelkie wypełnienia zarówno z lewej, jak i z prawej strony. Co więcej, aby zakładki były rozmieszczone bok siebie, a nie jedna nad drugą, w elementach listy musimy zastosować właściwość `float`. I w końcu nie możemy zapomnieć o usunięciu punktorów, które są standardowo wyświetlane z lewej strony każdego punktu listy. Wszystkie te zadania realizują dwa poniższe style:

```
.tabs {
    margin: 0;
    padding: 0;
}
.tabs li {
    float: left;
    list-style: none;
}
```

Uwaga: Przedstawiony tu kod CSS odnosi się do kodu HTML z poprzedniego punktu rozdziału. Innymi słowy, reguła `.tabs` odwołuje się do listy wypunktowanej — `<ul class="tabs">` — natomiast reguła `.tabs li` — do znaczników `<li>` umieszczonych wewnątrz tej listy.

- Same **zakładki** są reprezentowane przez znaczniki `<a>` umieszczone wewnątrz punktów listy, czyli znaczników `<li>`. W stylu określającym ich postać na pewno trzeba będzie odpowiednio ustawić kilka właściwości. Przede wszystkich chcemy usunąć podkreślenie, którym zazwyczaj są oznaczane wszystkie odnośniki, oprócz tego ich właściwości `display` przypiszemy wartość `block`, by można było określać ich marginesy i wypełnienia. Oto styl określający postać zakładek:

```
.tabs a {
    display: block;
    text-decoration: none;
    padding: 3px 5px;
}
```

  Oczywiście, można przypuszczać, że będziesz chciał uzupełnić tę regułę stylu o dodatkowe właściwości, by zakładki wyglądały naprawdę wspaniale. Mógłbyś na przykład ożywić je, określając kolor tła, zmienić czcionkę, jej kolor i wielkość, by tekst zakładek wyróżniał się wśród pozostałej treści kart.

- **Aktywna zakładka**. Bardzo dobrym pomysłem jest wyróżnienie zakładki skojarzonej z aktualnie widoczną kartą. To rodzaj sygnału „jesteś tutaj", który wizualnie identyfikuje informacje prezentowane na karcie. Popularnym rozwiązaniem stosowanym w tym celu jest utworzenie stylu, który przy użyciu jQuery zostanie dodany do zakładki po jej kliknięciu. Nie ma żadnych obowiązkowych właściwości, które musielibyśmy umieszczać w tym stylu, jednak warto nadać zakładce taki sam kolor tła, który ma powiązana z nią karta (a jednocześnie zapewnić, by pozostałe zakładki miały inny kolor tła), gdyż dzięki temu zakładka oraz karta będą tworzyły wizualną całość:

```
.tabs a.active {
    background-color: white;
}
```

Wskazówka: Często stosowanym rozwiązaniem jest dodawanie obramowań wokół zakładek i kart. Po kliknięciu zakładki ukrywamy jej dolne obramowanie, co sprawia wrażenie, jakby zakładka została zespolona z kartą (patrz rysunek 10.1). Aby powstało takie rozwiązanie, na początek w regule `.tabs a` należy dodać właściwość `border` oraz przypisać dolnemu marginesowi (`margin-bottom`) wartość `-1px`. Zastosowanie wartości ujemnej spowoduje przesunięcie zakładki o jeden piksel w dół, co sprawi, że będzie ona zachodzić na kartę. Dodatkowo w regule `.tabs a.active` należy nadać dolnej krawędzi obramowania kolor odpowiadający kolorowi tła kart. W ten sposób, choć krawędź obramowania wciąż będzie wyświetlana, ze względu na to, że będzie ono mieć ten sam kolor, co tło karty, a dodatkowo będzie zachodzić na jej obramowanie, będzie się wydawało, że zakładka i karta stanowią jedność. (Aby takie rozwiązanie działało w przeglądarce Internet Explorer 8 oraz jej wcześniejszych wersjach, konieczne jest także dodanie do reguły stylu właściwości `position:relative`). W końcu, możemy także dodać obramowanie do pojemnika zawierającego karty — powinno ono mieć taki sam styl, grubość i kolor, co obramowanie użyte w stylu *.tabs a*. Ostateczny efekt zastosowania takich stylów można zobaczyć w przykładzie zamieszczonym na stronie 320.

- **Pojemnik kart**. Bardzo ważny jest styl określający postać znacznika `<div>`, wewnątrz którego są umieszczone poszczególne karty i ich zawartość. Ponieważ w stylu dla zakładek użyliśmy właściwości `float:left` (aby przeglądarka wyświetliła je jedną obok drugiej), zatem musimy zadbać, by dalsza zawartość naszego komponentu była prawidłowo wyświetlana poniżej zakładek. W przeciwnym razie przeglądarka spróbuje wyświetlić ją z ich prawej strony.

  ```
  .panelContainer {
      clear: left;
  }
  ```

  Dodatkowo tego stylu można użyć w celu określenia postaci kart. Ponieważ pojemnik ten tworzy prostokąt wokół wszystkich kart, można w nim określić kolor tła, obramowanie, wypełnienie i tak dalej.

- **Karty**. Zgodnie z informacjami podanymi w poprzednim punkcie listy, do określenia podstawowych aspektów wyglądu kart, takich jak obramowanie, kolor tła i podobne, można użyć reguły stylów odnoszącej się do pojemnika, w którym są one umieszczone. Gdy jednak będziemy chcieli, możemy także określić regułę stylów odnoszącą się do poszczególnych znaczników `<div>` tworzących karty. Wystarczy w tym celu zdefiniować regułę stylu z selektorem `.panel`.

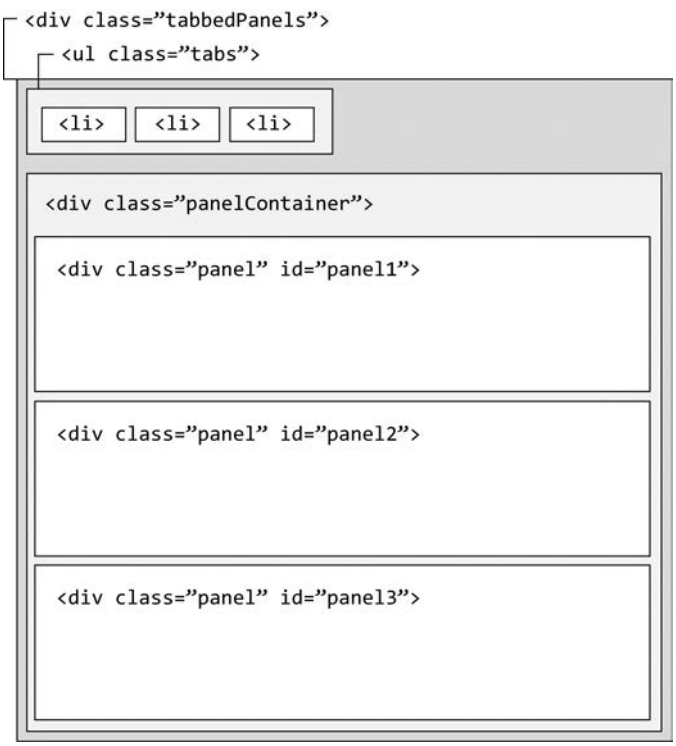
- **Zawartość kart**. Do określenia postaci zawartości umieszczonej na kartach można zastosować selektory elementów potomnych, które pozwolą odwoływać się do znaczników wewnątrz elementów `<div>` tworzących karty. Aby na przykład określić postać znaczników `<h2>` umieszczonych wewnątrz kart i wyświetlić ich zawartość czcionką Arial, w kolorze pomarańczowym, moglibyśmy użyć następującego stylu:

  ```
  .panel h2 {
      color: orange;
      font-family: Arial, Helvetica, sans-serif
  }
  ```

  Podobnie, by określić postać akapitów na kartach, należałoby użyć selektora w postaci `.panel p`.

Kod JavaScript

Przygotowaliśmy już kod HTML oraz arkusz stylów CSS i możemy zobaczyć zakładki wyświetlone w rzędzie u góry komponentu oraz trzy znaczniki `<div>` (karty), umieszczone poniżej, jeden nad drugim (patrz rysunek 10.2). Podstawowy wygląd komponentu jest zatem zgodny z naszymi zamierzeniami. Musimy jeszcze napisać kod JavaScript, który będzie obsługiwał otwieranie i zamykanie kart, zmieniał klasy oraz pozwalał na wyróżnianie aktywnej zakładki i przywracanie pozostałych do standardowej postaci. Oto czynności, jakie trzeba wykonać.



Rysunek 10.2. *Struktura kodu HTML panelu kart jest stosunkowo prosta; składa się ze znacznika , listy wypunktowanej, kilku odnośników oraz kilku kolejnych znaczników *

1. **Dodać obsługę zdarzeń `click` do odnośników w zakładkach**.

   Panele kart są nierozerwalnie związane z iteracją użytkownika z zakładkami — kliknięcie pierwszej zakładki powoduje wyświetlenie pierwszej karty, kliknięcie innej — wyświetlenie odpowiadającej jej karty.

2. **Dodać funkcję anonimową obsługującą zdarzenia `click`, która:**

   (a) Ukryje aktualnie widoczną kartę,

   (b) Usunie klasę `active` z wybranej wcześniej zakładki,

   (c) Doda klasę `active` do klikniętej zakładki,

   (d) Wyświetli kartę skojarzoną z kliknięta zakładką.

3. **Zgłosić zdarzenie `click` dla pierwszej zakładki**.

   Ten krok jest konieczny, gdyż w momencie wyświetlania strony widoczne są wszystkie karty, a żadna zakładka nie jest wyróżniona. Oczywiście, można napisać kod, który wyróżni pierwszą zakładkę oraz ukryje wszystkie karty z wyjątkiem pierwszej, jednak takie rozwiązanie nie jest konieczne — mamy do dyspozycji funkcję anonimową obsługującą zdarzenia `click` (patrz krok 2.), która zrobi to za nas. Możemy się zatem ograniczyć do programowego „kliknięcia" pierwszej zakładki, co spowoduje wykonanie tej funkcji.

Tak ogólnie wygląda kod, który musimy napisać. Napiszesz go krok po kroku w ramach przykładu, przedstawionego w następnym punkcie rozdziału.

Przykład — panel kart

Skoro już rozumiesz podstawowe założenia związane z tworzeniem panelu kart, tu znajdziesz opis czynności, jakie musisz wykonać, by go ostatecznie uruchomić. W tym przykładzie dodasz kody CSS oraz JavaScript, które przekształcą prostą listę odnośników przedstawioną na stronie 316 w interaktywny pasek nawigacyjny.

Uwaga: Informacje dotyczące pobierania przykładów do książki można znaleźć na stronie 43.

---

1. **W edytorze tekstów otwórz plik *tabs.html* umieszczony w katalogu *R10*.**

   Plik *tabs.html* zawiera kod HTML opisany na stronie 315: nadrzędny znacznik `<div>` całego panelu karty, wypunktowaną listę odnośników pełniących role zakładek, kolejny znacznik `<div>` zawierający karty oraz po jednym znaczniku `<div>` dla każdej karty. Znajdziesz w nim także podstawowe style CSS. Jeśli wyświetlisz ten plik w przeglądarce, zobaczysz trzy zakładki i zawartość trzech kart (wszystkie te karty są rozmieszczone w pionie, jedna nad drugą).

Uwaga: Starając się w możliwie jak największym stopniu poprawić przejrzystość przykładu, używany kod CSS umieściliśmy bezpośrednio w kodzie HTML strony, w formie arkusza wpisanego. Jeśli masz zamiar wielokrotnie korzystać z niego podczas tworzenia własnych paneli kart, umieść go w zewnętrznym pliku CSS.

   Plik biblioteki jQuery został już dołączony do strony, a w sekcji nagłówka znajduje się wywołanie funkcji `$(document).ready()`. Kolejnym krokiem, jaki wykonasz, będzie ukrycie kart.

2. **Kliknij puste miejsce wewnątrz funkcji `$(document).ready()` i wewnątrz niej dodaj poniższy kod wyróżniony pogrubieniem**:

```
$(document).ready(function() {
  $('.tabs a').click(function() {

  }); // koniec funkcji click
}); // koniec funkcji ready
```

   Wywołanie `$('.tabs a')` pobiera wszystkie znaczniki `<a>` umieszczone wewnątrz elementu klasy `tabs` (czyli naszej wypunktowanej listy). (Funkcja `click()`

została opisana na stronie 175). Aktualnie dysponujesz pustą funkcją anonimową, musisz zatem uzupełnić jej kod. Zaczniesz od prostej instrukcji, która pozwoli poprawić wydajność działania kodu.

3. **Wewnątrz funkcji anonimowej wpisz poniższy, pogrubiony kod:**

```
$('.tabs a').click(function() {
   $this = $(this);
}); // koniec funkcji click
```

Zgodnie z informacjami podanymi na stronie 162, wyrażenie `$(this)` stosowane wewnątrz funkcji anonimowej obsługującej zdarzenia pozwala odwołać się do elementu, do którego zdarzenie zostało skierowane — w naszym przypadku odwołuje się ono do zakładki kliknięte przez użytkownika. Za każdym razem, gdy używamy wyrażenia `$()` do pobrania elementu, wywołujemy funkcję jQuery, zmuszając tym samym przeglądarkę do wykonania wielu wierszy kodu JavaScript. Jeśli wewnątrz jakiejś funkcji będziemy wielokrotnie używali jakiegoś selektora jQuery, doskonałym pomysłem będzie zapisanie go w zmiennej. W powyższym przykładzie `$this` jest zwyczajną zmienną zdefiniowaną przez programistę (czyli przez Ciebie).

Zapisanie wartości wyrażenia `$(this)` w zmiennej oznacza, że gdy tylko będziesz chciał odwołać się do odnośnika, wystarczy skorzystać ze zmiennej `$this` — nie będziesz musiał ponownie pobierać go przy użyciu selektora jQuery. Innymi słowy, jeśli w kodzie funkcji dwukrotnie pojawi się wywołanie `$(this)`, będzie to oznaczać, że przeglądarka musi dwukrotnie wykonać funkcję jQuery w celu pobrania tego samego elementu. Jeśli za pierwszym razem zapiszesz wartość `$(this)` w zmiennej — `$this` — będziesz mógł z niej wielokrotnie korzystać bez zmuszania przeglądarki do wykonywania jakichkolwiek dodatkowych czynności (bardziej szczegółowe informacje o zaletach, jakie daje zapisywanie selektorów jQuery w zmiennych, można znaleźć na stronie 422).

Teraz zajmiesz się ukryciem kart i aktywacją kliknięte zakładki.

4. **Wpisz kod z wierszy 3. i 4. poniższego fragmentu:**

```
1  $('.tabs a').click(function() {
2      $this = $(this);
3      $('.panel').hide();
4      $('.tabs a.active').removeClass('active');
5  }); // koniec funkcji click
```

Wiersz 3. powoduje ukrycie wszystkich kart. Ponieważ każda z nich jest znacznikiem `<div>` należącym do klasy `panel`, selektor `$('.panel')` pobiera je wszystkie, a wywołanie funkcji `.hide()` (patrz strona 198) powoduje ich ukrycie. Musisz to zrobić, gdyż w przeciwnym razie po otworzeniu jednej karty poprzednia pozostałaby widoczna.

Wiersz 4. usuwa klasę `active` ze wszystkich zakładek — odnośników umieszczonych w elemencie należącym do klasy `tabs`. Na stronie 317 wyjaśniliśmy, że utworzenie klasy `active` pozwoli zmienić wygląd zakładki kliknięte przez użytkownika (by wyglądała jak wizualny sygnał „jesteś tutaj"). Oczywiście, kiedy użytkownik kliknie zakładkę, by ją uaktywnić, trzeba usunąć klasę `active` z zakładki, która do tej pory była aktywna. I właśnie to robi kod z wiersza 4., używając przy tym funkcji `removeClass()` (opisanej na stronie 155). Teraz zajmiesz się wyróżnieniem kliknięte zakładki.

5. **Dodaj kod umieszczony w wierszu 5.:**

```
1  $('.tabs a').click(function() {
2      $this = $(this);
3      $('.panel').hide();
4      $('.tabs a.active').removeClass('active');
5      $this.addClass('active').blur();
6  }); // koniec funkcji click
```

Pamiętasz zapewne, że `$this` jest zmienną, utworzoną w wierszu 2., która zawiera odwołanie do klikniętego odnośnika. A zatem wywołanie `$this.addClass ↪('active')` dodaje do tego odnośnika klasę `active` — przeglądarka użyje jej do określenia postaci klikniętej zakładki. Umieszczone na końcu wiersza wywołanie funkcji `.blur()` korzysta z techniki tworzenia sekwencji wywołań, jaką daje biblioteka jQuery (która została opisana na stronie 149). To zwyczajna funkcja wywoływana po wykonaniu funkcji `addClass()`. Funkcja ta usuwa ognisko wprowadzania z wybranego elementu (odnośnika lub pola formularza). W naszym przypadku jej wywołanie sprawi, że przeglądarka nie będzie wyświetlać wokół tekstu klikniętego odnośnika cienkiej, przerywanej linii. Gdybyśmy jej nie wywołali, zakładka nie wyglądałaby równie dobrze.

I to już prawie wszystko... jeszcze tylko musisz wyświetlić kartę.

6. **Dodaj kod umieszczony w wierszach 6. i 7.:**

```
1  $('.tabs a').click(function() {
2      $this = $(this);
3      $('.panel').hide();
4      $('.tabs a.active').removeClass('active');
5      $this.addClass('active').blur();
6      var panel = $this.attr('href');
7      $(panel).fadeIn(250);
8  }); // koniec funkcji click
```

Każda zakładka jest w rzeczywistości odnośnikiem wskazującym powiązaną z nią kartę. Pamiętasz zapewne, że kod HTML karty wygląda w następujący sposób: `<div id="panel1" class="panel">`; natomiast kod HTML odpowiadającej mu zakładki to: `<a href="#panel1">`. Zwróć uwagę, że zawartość atrybutu `href` odnośnika wygląda dokładnie tak samo jak selektor identyfikatora CSS. Ponieważ jQuery korzysta z selektorów CSS do pobierania elementów stron, zatem wystarczy pobrać wartość atrybutu `href` i użyć jej do pobrania karty, którą chcemy wyświetlić. W wierszu 6. tworzymy nową zmienną — `panel` — w której zapisujemy wartość atrybutu `href` odnośnika (używana przy tym funkcja `attr()` jQuery została opisana na stronie 159).

W wierszu 7. korzystamy z odczytanej wcześniej wartości do pobrania karty i stopniowego jej wyświetlenia (używamy przy tym funkcji `fadeIn()` opisanej na stronie 200). Moglibyśmy ją zastąpić jakąkolwiek inną funkcją jQuery generującą efekty wizualne, taką jak `show()`, `slideDown()` bądź `animate()`.

Teraz, gdy już prawie cały kod jest gotowy, musimy programowo wygenerować zdarzenie `click` podczas wczytywania strony, spowoduje to wywołanie funkcji, ukrycie kart, wyróżnienie pierwszej zakładki i wyświetlenie skojarzonej z nią karty. Na szczęście, jQuery pozwala w bardzo prosty sposób symulować zgłoszenie zdarzenia.

7. **Poniżej funkcji `click()` dodaj jeszcze jeden wiersz, powodujący zgłoszenie zdarzenia `click` dla pierwszej zakładki.**

```
1  $('.tabs a').click(function() {
2      $this = $(this);
3      $('.panel').hide();
4      $('.tabs a.active').removeClass('active');
5      $this.addClass('active').blur();
6      var panel = $this.attr('href');
7      $(panel).fadeIn(250);
8  }); // koniec funkcji click
9  $('.tabs li:first a').click();
```

Jak widać, w wyróżnionym wierszu zastosowaliśmy złożony selektor — `.tabs li:first a` — który służy do pobrania pierwszej zakładki. Selektor ten (podobnie jak wszystkie inne selektory elementów potomnych) należy analizować od strony prawej do lewej. Litera `a` umieszczona z prawej strony selektora oznacza, że chodzi o pobranie znacznika `<a>`. Umieszczony w środkowej części łańcuch `li:first` korzysta z pseudoelementu `first` pobierającego pierwszy element potomny. W naszym przypadku cały ten fragment selektora pozwala pobrać znacznik `<li>` będący pierwszym dzieckiem innego elementu. Ponieważ zakładki zostały utworzone przy użyciu listy, zatem `li:first` odpowiada pierwszemu elementowi tej listy (czyli pierwszej zakładce). I w końcu fragment `.tabs` pozwoli mieć pewność, że pobierzemy odnośnik umieszczony na liście będącej częścią naszego panelu kart. Zabezpiecza on przed przypadkowym pobraniem odnośnika zapisanego na innej liście wypunktowanej (na przykład pełniącej rolę paska nawigacyjnego), umieszczonej w innym miejscu strony.

Po pobraniu interesującego nas elementu wywołujemy funkcję `click()`; jednak w tym przypadku nie służy ona do określenia funkcji anonimowej obsługującej zdarzenia `click`, lecz do jego zgłoszenia. Innymi słowy, wywołanie umieszczone w wierszu 9. oznacza: „hej, przeglądarko — kliknij pierwszą zakładkę". Zgłoszenie tego zdarzenia powoduje całą sekwencję czynności: ukrycie kart, wyróżnienie zakładki i stopniowe wyświetlenie odpowiedniej karty. O rany! To już prawie wszystko. Gdybyś jednak stronę w tej postaci wyświetlił w przeglądarce, zauważyłbyś pewnie jeden, niewielki problem. Ponieważ zakładki są odnośnikami, gdy zatem okno przeglądarki będzie niewielkie, można zaobserwować, że po kliknięciu zakładki przeglądarka nie tylko wyświetli odpowiednią kartę, lecz także do niej przeskoczy. Musisz zatem poinstruować przeglądarkę, by nie przechodziła do miejsca docelowego odnośnika.

8. **Na końcu anonimowej funkcji obsługującej zdarzenia `click` (patrz wiersz 9.) dodaj instrukcję `return false;`. A tak powinna wyglądać ostateczna wersja kodu:**

```
1   $(document.ready(function() {
2       $($('.tabs a').click(function() {
3           $this = $(this);
4           $('.panel').hide();
5           $('.tabs a.active').removeClass('active');
6           $this.addClass('active').blur();
7           var panel = $this.attr('href');
8           $(panel).fadeIn(250);
9           return false;
10      }); // koniec funkcji click
11      $('.tabs li:first a').click();
12  }); // koniec funkcji ready
```

9. Zapisz plik i wyświetl stronę w przeglądarce.

   Dokończona przykładowa strona powinna wyglądać tak, jak przedstawiona na rysunku 10.3. Można ją poszerzyć o więcej zakładek i kart, dodając do listy kolejne punkty z odnośnikami wskazującymi kolejne znaczniki `<div>` reprezentujące nowe karty.

Rysunek 10.3. *Panele zakładek stanowią elegancki sposób zapewnienia użytkownikom łatwego dostępu do wielu informacji przy jednoczesnym oszczędzaniu miejsca na stronie*

Uwaga: Pełną wersję tego przykładu — *complete_tabs.html* — można znaleźć w katalogu *R10*. Dodatkowo umieściliśmy tam także bardziej złożoną wersję tego samego przykładu, pozwalającą na umieszczanie na jednej stronie kilku paneli kart. Znajdziesz ją w pliku *complete_complex_tabs.html*. Zostały w niej zastosowane zaawansowane funkcje jQuery służące do poruszania się po drzewie DOM strony, opisane w dalszej części książki, na stronie 432.

WIEDZA W PIGUŁCE

Projekt jQuery UI

Bardziej zaawansowaną wersję panelu kart można znaleźć w projekcie jQuery UI. Jest to oficjalny projekt zespołu jQuery, którego celem jest pisanie wtyczek rozwiązujących popularne zadania związane z tworzeniem interfejsu użytkownika; są to „akordeony" (ang. *accordion*), karty, okna dialogowe, kalendarze oraz elementy stron, które można przeciągać. Uczestnicy projektu dążą do opracowania jednej wtyczki, która zapewniłaby możliwość rozwiązania większości problemów napotykanych podczas tworzenia interfejsu użytkownika aplikacji internetowych. Projekt ten ma swoją własną witrynę WWW (*http://jqueryui.com/*), na której można znaleźć najnowszą wersję kodu, przykłady oraz odnośnik do dokumentacji umieszczonej na głównej witrynie jQuery.

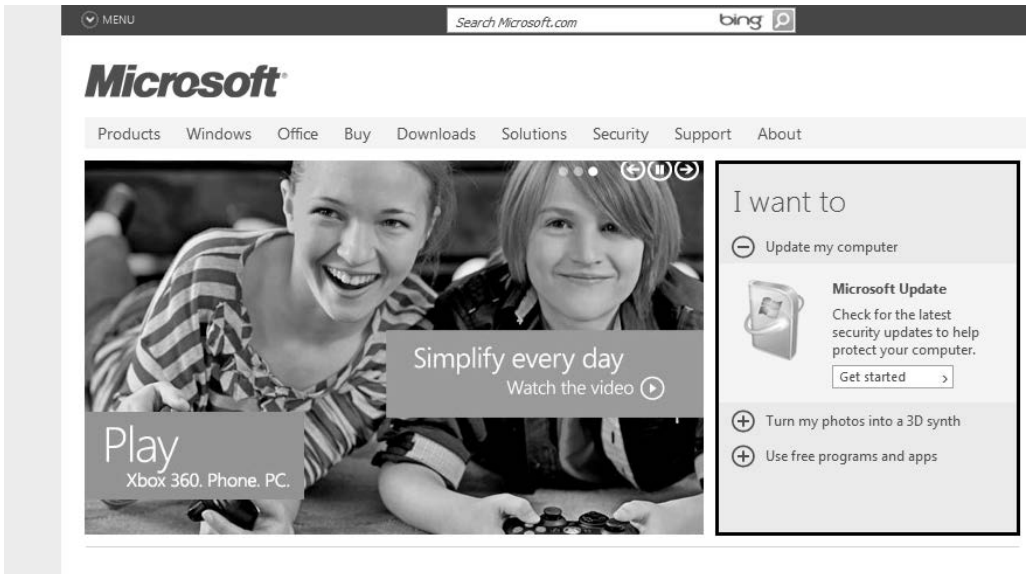
Projekt jQuery UI udostępnia wiele narzędzi dla projektantów stron, a nawet obsługuje tematy CSS — pozwalające na zapewnienie wspólnego, spójnego wyglądu wszystkich elementów jQuery UI. Projekt ten jest stosunkowo złożony i w jego skład wchodzi wiele różnych elementów. Pliki projektu można także modyfikować i dostosowywać do własnych potrzeb — usuwać z nich komponenty, których nie będziemy potrzebować, ograniczając tym samym wielkość pliku i skracając czas jego pobierania. Można nawet tworzyć własne tematy CSS dopasowujące wygląd komponentów jQuery UI do wyglądu naszej witryny. Cały ten proces ułatwia specjalne narzędzie służące do przygotowywania pliku jQuery UI, dostępne na stronie *http://jqueryui.com/donwload*.

W poprzednim wydaniu tej książki wtyczka jQuery UI była używana w kilku rozdziałach, jednak w międzyczasie zespół jej twórców zdecydował się na całkowite przepisanie jej kodu i dodanie wielu nowych, fascynujących komponentów oraz możliwości. Niestety, w czasie pisania tej książki najnowsza wersja wtyczki jQuery UI (numer 1.9) nie była jeszcze dostępna, dlatego też nie warto tracić czasu na poznawanie grupy komponentów, które i tak niebawem zostaną zastąpione. Dlatego w tym wydaniu książki jQuery UI i jej komponenty nie zostały opisane.

Jednak jQuery UI zapowiada się doskonale i zdecydowanie warto się nią interesować. Zajrzyj zatem na witrynę jQuery UI, by przekonać się, czy wersja 1.9 została już opublikowana; jeśli tak, sprawdź ją koniecznie.

Dodawanie sliderów

Kolejnym narzędziem używanym przez projektantów stron do walki ze zbyt wielką liczbą prezentowanych informacji są slidery (ang. *content slider*) — proste komponenty interfejsu użytkownika, prezentujące jedno wybrane zdjęcie lub fragment treści, wybrane z większej grupy. Wiele witryn zawierających bardzo duże ilości informacji, takich jak witryna firmy Microsoft, korzysta ze sliderów w celu prezentowania zdjęć, tekstów oraz odnośników w niewielkich fragmentach, które są przesuwane po ekranie i zastępowane innymi (patrz rysunek 10.4). Slider przypomina nieco panel kart, ale w jego przypadku poszczególne karty mają zazwyczaj tę samą wielkość, są wyświetlane i chowane z wykorzystaniem animacji, które je przesuwają na ekranie, a ich pojawianie się i znikanie jest zazwyczaj obsługiwane przy użyciu liczników czasu. Są one powszechnie stosowane na stronach głównych, gdyż mają bardzo atrakcyjną postać, a jednocześnie pozwalają na zachowanie prostoty strony. Często używa się ich także jako zwiastunów reklamujących treści lub produkty opisane na innych stronach witryny. Kliknięcie karty takiego slidera zazwyczaj powoduje przejście na inną stronę.



Rysunek 10.4. *Aby zminimalizować natłok informacji na ekranie, witryny, takie jak Microsoft.com, korzystają ze sliderów (na rysunku jeden został zaznaczony czarną ramką), wewnątrz których prezentowane są obrazki lub fragmenty treści, po jednym w danej chwili. W przedstawionym przykładzie zaznaczony obrazek może zostać wysunięty na bok, odsłoni się wtedy inny obrazek z kolejnymi informacjami*

Utworzenie slidera wymaga opanowania kilku możliwości języka JavaScript i biblioteki jQuery, a konkretnie tworzenia animacji, korzystania z liczników czasu oraz manipulacji kodem HTML i CSS. Choć — oczywiście — można utworzyć swój własny slider, jednak istnieje sporo wtyczek jQuery udostępniających wiele przydatnych możliwości. Jedną z najbardziej wszechstronnych wtyczek tego typu jest AnythingSlider (kod można pobrać ze strony *https://github.com/ProLoser/AnythingSlider/*).

Stosowanie slidera AnythingSlider

Do działania wtyczki AnythingSlider potrzebujemy kilku plików; są to jQuery (co chyba oczywiste), zewnętrzny plik JavaScript z kodem obsługującym slider, plik CSS ze stylami określającymi podstawowy wygląd slidera i stosowanych w nim efektów oraz obrazek z kontrolkami slidera (przyciskami następny i poprzedni). Pliki te można pobrać ze strony *https://github.com/ProLoser/AnythingSlider/*. (Dodaliśmy je także do przykładów dołączonych do tej książki i umieściliśmy w katalogu z przykładami do tego rozdziału). Aby skorzystać z tej wtyczki, należy wykonać następujące, bardzo proste czynności.

1. **Dołączyć do strony plik CSS *anythingslider.css*.**

   Ten zewnętrzny arkusz stylów CSS określa sposób formatowania przycisków nawigacyjnych slidera, jak również ukryte style służące do właściwego rozmieszczania poszczególnych kart. Istnieje możliwość określania postaci podstawowych elementów slidera poprzez modyfikowanie reguł CSS umieszczonych w tym pliku.

2. **Dołączyć do strony plik biblioteki jQuery.**

   Biblioteka jQuery udostępnia wszystkie podstawowe narzędzia niezbędne do utworzenia i obsługi slidera AnythingSlider. Podobnie jak podczas korzystania ze wszystkich innych wtyczek jQuery, także i teraz na początek należy wczytać sam plik biblioteki. Jeśli wtyczka zostanie wczytana przed biblioteką jQuery, slider nie będzie działał.

3. **Dołączyć plik JavaScript z kodem wtyczki AnythingSlider.**

   Ten plik zawiera cały kod odpowiadający za przekształcenie kodu HTML w interaktywny slider.

4. **Dodać kod HTML.**

   AnythingSlider nie wymaga żadnego skomplikowanego kodu HTML. Należy tylko określić pojemnik — znacznik `<div>` z identyfikatorem slidera: `<div id="slider">` — a wewnątrz niego umieścić po jednym znaczniku `<div>` dla każdej z kart. Jak widać, rozwiązanie to w znacznym stopniu przypomina panel kart (opisany na stronie 315).

5. **Dodać znacznik `<script>`, a wewnątrz niego umieścić wywołanie funkcji `$(document).ready()`, w której z kolei wywołana zostanie funkcja slidera**.

   Jedną z ogromnych zalet korzystania z wtyczek jQuery jest to, że zazwyczaj kod, jaki w tym celu musimy napisać, jest bardzo krótki i prosty. W naszym przypadku wszystkim, czego potrzebujemy do utworzenia slidera, jest dopisanie poniżej odwołania dodanego w kroku 3. poniższego fragmentu kodu:

```
<script>
$(document).ready(function) {
    $('#slider").anythingSlider();
}); // koniec funkcji ready
</script>
```

Istnieje wiele różnych sposobów określania postaci slidera AnythingSlider, o czym przekonasz się dalej w tym rozdziale. Jednak najpierw sprawdź, jak taki slider działa.

Przykład — AnythingSlider

Utworzenie prostego slidera jest bardzo proste. Przekonasz się o tym, wykonując przedstawiony tu przykład. Możesz w tym celu wykorzystać dowolny edytor HTML.

Uwaga: Informacje dotyczące pobierania przykładów dołączonych do tej książki można znaleźć na stronie 43.

1. **W edytorze tekstów otwórz plik *slider.html* umieszczony w katalogu *R10*.**

   Pierwszym krokiem będzie dodanie do strony pliku CSS wtyczki.

2. **Kliknij pusty wiersz umieszczony poniżej wiersza `<link href="../_css/site.css" rel="stylesheet">` i wpisz w nim:**

```
<link rel="stylesheet" href="anythingSlider/anythingslider.css">
```

   Ten wiersz kodu wczytuje arkusz stylów *anythingslider.css*, zawierający style określające postać slidera. Zawartości tego pliku przyjrzymy się dokładniej nieco później, kiedy będziemy zajmowali się aktualizacją wyglądu naszego komponentu. Kolejnym krokiem będzie dodanie do strony niezbędnych plików JavaScript.

3. **W kolejnych dwóch wierszach wpisz poniższy kod:**

```
<script src="../_js/jquery-1.6.3.min.js"></script>
<script src="anythingSlider/jquery.anythingslider.min.js"></script>
```

Pierwszy z powyższych znaczników `<script>` powoduje wczytanie pliku biblioteki jQuery, natomiast drugi wczytuje plik wtyczki AnythingSlider. Kolejnym krokiem będzie dodanie do strony kodu HTML.

Wskazówka: W tym przykładzie zakładamy, że pliki JavaScript oraz CSS wtyczki są umieszczone w odrębnym katalogu *anythingSlider*, umieszczonym wewnątrz katalogu z przykładami do rozdziału — *R10*. Umieszczenie wszystkich plików wymaganych do prawidłowego działania wtyczki w osobnym katalogu jest doskonałym sposobem zagwarantowania, że wszystkie będą się znajdowały tam, gdzie powinny; takie rozwiązanie ułatwia także wykorzystanie wtyczki na innych witrynach. Jeśli podobają Ci się efekty działania wtyczki AnythingSlider, wystarczy, że skopiujesz katalog *anythingSlider* na swoją witrynę (umieść go w katalogu głównym bądź wewnątrz katalogu, gdzie przechowywane są pliki JavaScript).

---

4. **W kodzie strony odszukaj znacznik nagłówka poziomu pierwszego — `<h1>Anything Slider</h1>` — i w wierszu poniżej niego wpisz:**

```
<div id="slider">

</div>
```

Jak widać, użyliśmy tu znacznika `<div>` (czyli znacznika HTML służącego do definiowania regionów stron WWW). Ten konkretny znacznik będzie reprezentować sam komponent slidera. Wewnątrz niego umieścisz kolejne znaczniki `<div>` — po jednym dla każdej z kart slidera.

5. **Wewnątrz elementu `<div>` slidera (czyli pomiędzy otwierającym znacznikiem `<div>` i zamykającym znacznikiem `</div>`) wpisz:**

```
<div>
<a href="page1.html"><img src="images/pumpkin.jpg" width="700"
height="390"
alt="Dynia"></a>
</div>
```

Ten drugi znacznik `<div>` zawiera odnośnik i obrazek. Kliknięcie obrazka spowoduje przejście na inną stronę: takie rozwiązanie jest powszechnie stosowane w sliderach, które bardzo często spełniają rolę animowanych reklam na stronie. Każda karta slidera jest swoistym zwiastunem reklamującym inne treści, a zatem użytkownicy mogą ją kliknąć, by przejść do artykułu lub innego fragmentu witryny.

Gdy wykorzystamy slider AnythingSlider, w kartach będziemy mogli umieszczać dowolny kod HTML — nasze możliwości nie są ograniczone do jednego dużego zdjęcia. Można w nich umieszczać tekst, obrazki oraz inne znaczniki `<div>` — niemal wszystko, co tylko zechcemy.

6. **Wewnątrz elementu `<div>` slidera dodaj kolejne dwa znaczniki `<div>`:**

```
<div>
<a href="page2.html"><img src="images/grapes.jpg" width="700"
height="390"
alt="Winogrona"></a>
</div>
<div>
```

```
<a href="page3.html"><img src="images/various.jpg" width="700"
➥height="390"
alt="Rozmaitości"></a>
</div>
```

Te dwa zagnieżdżone znaczniki `<div>` reprezentują kolejne karty. Możesz ich dodać tyle, ile zechcesz. Teraz nadszedł czas, by zabrać się za pisanie kodu JavaScript.

7. **W górnej części pliku, poniżej drugiego znacznika `<script>`, lecz przed zamykającym znacznikiem `</head>`, dodaj pusty wiersz i wpisz w nim:**

```
<script>
$(document).ready(function() {
    $('#slider').anythingSlider();
});
</script>
```

Możesz w to uwierzyć lub nie, jednak wszystko, co musisz zrobić, to pobranie znacznika `<div>` slidera — `$('#slider')` — i wywołanie funkcji `anything➥Slider()`. Całą resztą zajmie się już sama wtyczka.

8. **Zapisz plik i wyświetl stronę w przeglądarce.**

Strona powinna wyglądać tak, jak przedstawiona na rysunku 10.5. (Jeśli jednak tak nie wygląda, powinieneś ponownie sprawdzić jej kod. Możesz go porównać z plikiem *complete_slider.html*, zawierającym końcową, pełną wersję kodu przykładu). Wypróbuj działanie elementów sterujących: strzałka w prawo powoduje wyświetlenie następnej karty, a strzałka w lewo — karty poprzedniej, przyciski z cyframi pozwalają przejść do konkretniej karty, a przycisk *Start* uruchamia automatyczny pokaz slajdów.

Modyfikowanie wyglądu slidera

Jak widać, stosowanie wtyczki AnythingSlider jest bardzo proste. Oczywiście, domyślny wygląd slidera nie musi wcale pasować do projektu naszej witryny i może się zdarzyć, że nie będziemy chcieli bądź potrzebowali wszystkich jego możliwości (takich jak automatyczny pokaz slajdów lub przyciski pozwalające przechodzić do następnej lub poprzedniej karty). Wygląd slidera AnythingSlider można zmieniać na kilka sposobów: modyfikując pliki graficzne, wprowadzając zmiany w arkuszach stylów oraz ustalając opcje wtyczki (ten sposób został opisany w następnym punkcie rozdziału).

Dzięki zastosowaniu techniki „sprajtów CSS" jeden plik graficzny spełnia wiele zadań — określa normalny i „wyróżniony" stan strzałek do przodu i do tyłu, zawiera tło przycisków z numerami kart oraz ustala postać przycisku *Start* (więcej informacji na temat tej techniki można znaleźć na stronie *http://css-tricks.com/css-sprites/*). Możesz otworzyć ten plik w dowolnym programie graficznym i zmodyfikować wygląd przycisków strzałek, każdy z nich ma wymiary 45×140 pikseli.

Można także modyfikować arkusz stylów i w ten sposób wypływać na postać pokazu slajdów. Poniżej zamieszczono listę najczęściej stosowanych zmian, z których, być może, będziesz chciał skorzystać.

Rysunek 10.5. *Przy użyciu wtyczki AnythingSlider można szybko utworzyć interaktywny pokaz slajdów, by zwrócić uwagę użytkownika na wybrane strony witryny lub prezentowane na niej produkty*

- **Wysokość oraz szerokość slidera**. Pierwsza reguła stylu zapisana w pliku *anythingslider.css* — `#slider` — określa ogólną szerokość i wysokość komponentu. Szerokość (`width`) można zmieniać, by tworzyć szersze bądź węższe prezentacje; a wysokość (`height`) regulować, jeśli prezentowana zawartość ma inną wysokość niż domyślne 390 pikseli.
- **Kolor przycisków nawigacyjnych**. Przyciski z cyframi wyświetlane u dołu slidera są zazwyczaj zielone. Jeśli jednak nie podoba Ci się ten kolor, możesz zmodyfikować regułę stylu z bardzo złożonym selektorem `div .anythingSlider. ➥activeSlider .anythingControls ul a.cur, div.anythingSlider.active ➥Slider .anythingControls ul a`. Zmień podany w niej kolor o wartości `#7C9127` na dowolny inny, lepiej odpowiadający projektowi Twojej witryny. Jeśli chciałbyś także określić kolor czcionki, dodaj do reguły właściwość `color`, na przykład:

```
color: #F44439;
```

- **Kolor przycisków wskazanych myszą**. Możesz także zmienić kolor tła przycisków nawigacyjnych, używaną w nich czcionkę oraz dowolne inne aspekty ich wyglądu; wystarczy w tym celu zmodyfikować regułę stylu z selektorem `div .anythingSlider .anythingControls ul a:hover`. W domyślnej postaci usuwa ona z przycisku obraz tła (cień).

- **Aktualnie wybrany przycisk nawigacyjny**. Aby określić styl wyróżniający przycisk skojarzony z aktualnie wybraną kartą, należy dodać do arkusza regułę z selektorem `div.anythingSlider .activeSlider .anythingControls ul a.cur` i określić w nim kolor tła, czcionkę i tak dalej. Ważne jest, by reguła ta była umieszczona w arkuszu stylów poniżej reguły opisanej w punkcie „Kolor przycisków nawigacyjnych"; alternatywnie możesz także zmodyfikować opisaną tam regułę stylu, usuwając z niej selektor `div.anythingSlider .activeSlider .anythingControls ul a.cur`. Ponieważ w stylu podanym wcześniej jest już określony kolor tła, zatem przesłoni on kolor podany w tej regule, chyba że zostanie ona umieszczona bliżej końca arkusza stylów.
- **Kolory przycisków rozpoczynających i zatrzymujących prezentację**. Postać przycisków służących do rozpoczynania i zatrzymywania automatycznej prezentacji kart jest kontrolowana przy użyciu dwóch stylów. Aby zmienić zielone tło oraz czcionkę przycisku rozpoczynającego prezentację, należy zmodyfikować regułę stylu z selektorem `div.anythingSlider .start-stop`. Z kolei modyfikując regułę z selektorem `div .anythingSlider .start-stop.playing`, można zmienić czerwony kolor przycisku przerywającego automatyczną prezentację.
- **Usunięcie cieni oraz inne zmiany wyglądu przycisków nawigacyjnych**. Jeśli nie podobają Ci się cienie widoczne przy przyciskach nawigacyjnych oraz obsługujących automatyczną prezentację, powinieneś zmodyfikować reguły stylów z selektorami `div.anythingSlider .anythingControls ul a` oraz `div .anythingSlider .start-stop`. Konkretnie rzecz biorąc, musisz usunąć z nich właściwość `border-image`. Możesz także zmodyfikować właściwości `border-radius`, `-moz-border-radius` oraz `-webkit-border-radius`, by całkowicie usunąć lub zmienić promień okrągłych wierzchołków tych przycisków. Ogólnie mówiąc, style te określają podstawowe aspekty wyglądu przycisków slidera, zatem warto z nimi poeksperymentować, by przekonać się, jakie efekty można za ich pomocą uzyskać.
- **Zielone obramowania powyżej i poniżej prezentacji**. Powyżej oraz poniżej slidera wyświetlane jest zielone obramowanie o szerokości trzech pikseli. Aby je zmienić, należy zmodyfikować styl z selektorem `div.anythingSlider .anythingWindow`. Jeśli chcesz całkowicie usunąć obramowanie, usuń właściwości `border-top` oraz `border-bottom`; ewentualnie zmodyfikuj podane w nich wartości, by zmienić kolor bądź szerokość obramowania.
- **Położenie przycisków strzałek**. Położenie przycisków pozwalających na przejście do poprzedniej i następnej karty można kontrolować, modyfikując odpowiednio regułę stylu z selektorem `div.anythingSlider .back` (strzałka w lewo) oraz `div.anythingSlider .right` (strzałka w prawo). Oprócz tego, reguła z selektorem `div.anythingSlider .arrow` określa pewne wspólne aspekty wyglądu obu tych przycisków, w tym ich położenie pośrodku obszaru slidera. Gdybyś na przykład chciał wyświetlić te przyciski bliżej górnej krawędzi slidera, wystarczy, że zmodyfikujesz styl `div.anythingSlider .arrow`, zmieniając właściwość `top: 50%` na `top: 20%` bądź nawet na wartość bezwzględną wyrażoną w pikselach — `top: 45px`.

Modyfikacja działania slidera

Już sama modyfikacja arkusza stylów CSS pozwala na wprowadzenie wielu zmian w wyglądzie slidera. By jednak wprowadzić fundamentalne zmiany w sposobie działania tej wtyczki, konieczne jest określenie wartości kilku jej właściwości. W tym celu trzeba przekazać do niej odpowiedni literał obiektowy (patrz strona 158):

```
{
   buildArrows : false,
   startText : "Uruchom prezentację",
   stopText : "Zatrzymaj prezentację"
}
```

W tym przykładzie `buildArrows` jest właściwością wtyczki, natomiast `false` — przypisywaną jej wartością. Zastosowanie tej konkretnej wartości sprawi, że wtyczka nie wyświetli w sliderze strzałek do przechodzenia pomiędzy kolejnymi kartami. Za każdą parą nazwa-wartość, z wyjątkiem ostatniej, należy umieścić przecinek (zwróć uwagę, że nie ma go za parą `stopText : "Zatrzymaj prezentację"`).

Ten literał obiektowy należy następnie przekazać w wywołaniu funkcji `anything ↪Slider()`. Oto przykład:

```
$('#slider').anythingSl[PR12]ider({
   buildArrows : false,
   startText : "Uruchom prezentację",
   stopText : "Zatrzymaj prezentację"
});
```

Poniżej przedstawiono kilka najbardziej przydatnych opcji.

- **Ukrycie przycisków nawigacyjnych**. Aby ukryć przyciski ze strzałkami, należy przypisać wartość `false` właściwości `buildArrows`:

  ```
  buildArrows : false
  ```

- **Zmiana etykiet przycisków**. Aby zmienić teksty wyświetlane po wskazaniu przycisków rozpoczynającego i przerywającego automatyczną prezentację, należy je podać we właściwościach `startText` oraz `stopText`:

  ```
  startText : "Uruchom prezentację",
  stopText : "Zatrzymaj prezentację"
  ```

- **Wyłączenie automatycznego odtwarzania**. Być może nie będziesz chciał wyświetlać przycisków do rozpoczynania i przerywania automatycznej prezentacji, bo preferujesz zapewnienie użytkownikowi możliwości samodzielnego wyboru karty, która ma być widoczna. W takim przypadku powinieneś przypisać wartość `false` właściwości `buildStartStop`:

  ```
  buildStartStop : false
  ```

- **Animacja w pionie**. Aby karty w sliderze były przesuwane w kierunku pionowym, a nie w poziomie, przypisz wartość `true` właściwości `vertical`:

  ```
  vertical : true
  ```

- **Automatyczne odtwarzanie**. Jeśli chcesz, by w momencie wyświetlenia strony rozpoczynała się automatyczna prezentacja kart slidera, przypisz wartość `true` właściwości `autoPlay`:

  ```
  autoPlay : true
  ```

Automatyczne rozpoczynanie prezentacji jest bardzo popularnym rozwiązaniem, często wykorzystywanym na witrynach ze sliderami — pozwala wyświetlić więcej treści bez konieczności zmuszania użytkownika do klikania przycisku „rozpocznij pokaz".

- **Karty o różnej wielkości**. Jeśli zawartości poszczególnych kart są różnej wielkości, możesz zażądać, by okno slidera zmieniało wielkość i dostosowywało się do wymiarów zawartości aktualnie prezentowanej karty. Załóżmy na przykład, że pierwsza karta zawiera znacznik `<div>`, wewnątrz którego został umieszczony pojedynczy akapit tekstu; natomiast na drugiej karcie mamy znacznik `<div>` zawierający nagłówek, dwa duże zdjęcia oraz trzy akapity tekstu. Jeśli w takim przypadku właściwości `resizeContents` przypiszesz wartość `true`, slider będzie automatycznie zmieniał swoją wielkość, dostosowując się do wielkości zawartości prezentowanej karty. W naszym przypadku w sliderze prezentowana jest początkowo karta z jednym akapitem, a zatem jego wysokość będzie niewielka. Jednak po kliknięciu przycisku i przejściu do następnej karty — o znacznie większej zawartości — wysokość slidera zostanie powiększona. Aby zapewnić takie działanie komponentu, umieść w literale obiektowym następującą właściwość:

```
resizeContents : true
```

Aby przekonać się, jakie efekty daje zastosowanie niektórych spośród tych właściwości, wyświetl w przeglądarce przykładową stronę *complete_slider2.html*.

Jak mogłeś się przekonać, zarówno umieszczanie komponentu AnythingSlider na stronie, jak i dostosowywanie jego wyglądu i działania do własnych potrzeb jest całkiem proste. Przedstawiliśmy tu jednie drobny ułamek wszystkich możliwości tej wtyczki. Pozwala ona także na prezentowanie klipów wideo, dodawanie efektów specjalnych oraz stosowanie własnego kodu JavaScript, który zapewni, że prezentowany slider będzie działał dokładnie tak, jak chcemy. Więcej informacji na ten temat można znaleźć w Wiki wtyczki AnythingSlider, dostępnej na stronie *https://github.com/ProLoser/AnythingSlider/wiki*.

Określanie wielkości i położenia elementów strony

Podczas dynamicznego modyfikowania zawartości stron lub dodawania do nich nowych treści z wykorzystaniem języka JavaScript i biblioteki jQuery często bardzo przydatna może się okazać znajomość rozmiaru oraz położenia elementów strony. Przykładowo może się zdarzyć, że będziesz chciał wyświetlić nad zawartością strony dodatkową warstwę, tak zwaną *nakładkę* (ang. *overlay*; tworzy ona efekt, w którym zawartość strony zostaje „wyszarzona", podobny do tego, jaki daje wtyczka FancyBox opisana na stronie 244). W tym celu konieczne jest dodanie do strony bezwzględnie umieszczonego znacznika `<div>`, który przykryje całą zawartość okna przeglądarki. Aby to zrobić, trzeba mieć pewność, że znacznik ten będzie miał dokładnie takie same wymiary jak okno programu, a to oznacza, że trzeba będzie w jakiś sposób je określić.

Gdybyśmy chcieli utworzyć etykietki ekranowe — niewielkie okienka wyświetlane, gdy użytkownik umieści wskaźnik myszy w obszarze jakiegoś elementu — konieczne będzie określenie współrzędnych wskaźnika myszy, co pozwoli umieścić etykietkę w odpowiednim miejscu.

Określanie wysokości i szerokości elementów

Biblioteka jQuery udostępnia funkcje `.height()` oraz `.width()`, które zwracają odpowiednio wysokość i szerokość wybranego elementu strony. Podając odpowiedni selektor, można określić wymiary dowolnego znacznika wchodzącego w skład strony, a nawet całego okna przeglądarki lub całej zawartości dokumentu.

- **Wysokość i szerokość okna przeglądarki**. Jeśli chcesz poznać wysokość i szerokość okna przeglądarki, musisz użyć selektora `$(window)`, a następnie wywołać funkcję `height()` lub `width()`:

  ```
  var winH = $(window).height();
  var winW = $(window).width();
  ```

  Powyższy fragment kodu pobiera wysokość oraz szerokość okna przeglądarki i zapisuje je w dwóch zmiennych. Pobieranie wymiarów okna przeglądarki jest przydatne, kiedy chcemy mieć pewność, że jakiś element nie zostanie umieszczony poza widocznym obszarem strony.

- **Wysokość i szerokość dokumentu**. Dokument to nie to samo, co okno przeglądarki, i w większości przypadków ma zupełnie inne wymiary. Dokument reprezentuje naszą stronę WWW; jeśli umieścimy na niej tylko niewielki fragment tekstu — na przykład jeden akapit — dokument będzie miał wysokość tego akapitu (powiększoną dodatkowo o marginesy górny i dolny). Na dużym monitorze wysokość takiego dokumentu będzie mniejsza od wysokości okna przeglądarki.

  W odwrotnej sytuacji — gdy zawartość strony jest bardzo obszerna i użytkownik musi ją przewijać, by dotrzeć do jej końca — wysokość dokumentu będzie większa od wysokości okna. Podobnie, gdybyś zmodyfikował styl określający postać znacznika `<div>`, wewnątrz którego umieszczona jest cała zawartość strony, i nadał mu szerokość 1500 pikseli, okazałoby się, że szerokość dokumentu jest większa od szerokości okna przeglądarki. Aby określić wysokość i szerokość dokumentu, wywołania metod `height()` i `width()` należy poprzedzić selektorem `$(document)`:

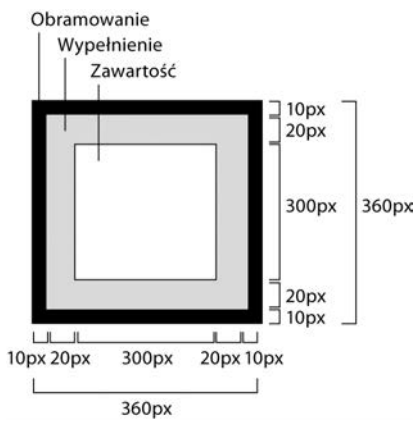
  ```
  var docH = $(document).height();
  var docW = $(document).width();
  ```

Funkcji `heigth()` oraz `width()` można także używać do określania wymiarów zwyczajnych elementów HTML, takich jak akapity, sekcje (znaczniki `<div>`) czy też obrazki, jednak w takich przypadkach nie zawsze będą one zwracać te informacje, których poszukujemy. Funkcje te zwracają wartości właściwości `height` oraz `width` CSS, a te nie zawsze są takie same jak faktyczne wymiary elementów strony. Właściwości te określają wymiary przydzielane zawartości znacznika, czyli na przykład tekstowi umieszczonemu wewnątrz akapitu. Kiedy jednak dodamy do elementu marginesy, wypełnienie oraz obramowanie, całkowity obszar, jaki zajmuje na stronie, będzie większy od tego, co wskazują właściwości `height` i `width`.

Aby zrozumieć, jak to działa, przeanalizujmy poniższy, przykładowy kod CSS określający postać znacznika `<div>`:

```
div {
   width : 300px;
   height : 300px;
   padding : 20px;
   border : 10px solid black;
}
```

Schemat tego elementu został przedstawiony na rysunku 10.6. Rzeczywista wysokość i szerokość tego elementu na stronie będzie wynosić 360 pikseli, gdyż stanowi sumę wysokości (lub szerokości) wypełnienia oraz obramowania. A zatem, faktyczna szerokość elementu jest sumą grubości jego lewego obramowania, szerokości lewego wypełnienia, szerokości określonej w stylu CSS, szerokości prawego wypełnienia oraz grubości prawego obramowania; analogicznie, faktyczna wysokość jest sumą grubości górnego obramowania, wysokości górnego wypełnienia, wysokości określonej w stylu CSS, wysokości dolnego wypełnienia oraz grubości dolnego obramowania.



Rysunek 10.6. *Podczas stosowania stylów CSS wysokość i szerokość są prostymi zagadnieniami. Właściwości width i height CSS określają wtedy wyłącznie wymiary zawartości elementu. Jednak faktyczna szerokość i wysokość obszaru, jaki element zajmuje na stronie, wyliczana jest poprzez dodanie do wymiarów zawartości elementu także wielkości jego wypełnienia i obramowania*

Ze względu na te wszystkie wymiary jQuery udostępnia trzy zestawy funkcji służących do określania różnych szerokości i wysokości elementów strony. Oto one.

- `width()` i `height()` — funkcje te zwracają szerokość i wysokość podane w stylu CSS. Załóżmy na przykład, że strona zawiera znacznik `<div>`, którego postać określa przestawiona powyżej reguła CSS.

  ```
  var divW = $('div').width(); //300
  var divH = $('div').height(); //300
  ```

  Po wykonaniu powyższego fragmentu kodu zmiennym `divW` oraz `divH` zostanie przypisana wartość `300` — czyli szerokość i wysokość określone w stylu CSS.

- Funkcja `innerWidth()` zwraca szerokość elementu podaną w stylu CSS powiększoną o szerokość prawego i lewego wypełnienia, a funkcja `innerHeight()` — wysokość elementu podaną w stylu CSS powiększoną o wysokość górnego i dolnego wypełnienia:

  ```
  var divW = $('div').innerWidth(); //340
  var divH = $('div').innerHeight(); //340
  ```

  W tym przypadku zmienne `divW` oraz `divH` przyjmą wartość `340`, odpowiadającą szerokości (i wysokości) podanej w regule stylu, powiększonej o wymiary wypełniania z obu stron elementu.

- Funkcja `outerWidht()` zwraca szerokość podaną w stylu CSS, powiększoną o szerokość prawego i lewego wypełnienia oraz grubość prawego i lewego obramowania; analogicznie, funkcja `outerHeight()` zwraca wysokość elementu podaną w stylu CSS, powiększoną o wysokość górnego i dolnego wypełnienia oraz grubość górnego i dolnego obramowania.

```
var divW = $('div').outerWidth(); //360
var divH = $('div').outerHeight(); //360
```

Po wykonaniu powyższego fragmentu kodu w zmiennych `divW` oraz `divH` zostanie zapisana wartość `360`, czyli szerokość (i wysokość) podana w regule CSS, powiększona o wielkość wypełnienia i grubość obramowania z obu stron elementu.

Funkcje `outerWidht()` oraz `outerHeight()` pobierają także jeden dodatkowy argument — wartość `true`, której przekazanie sprawi, że funkcje te będą uwzględniały w obliczeniach także wielkość marginesów elementu. Przykładowo załóżmy, że postać znacznika `<div>` jest określana przy użyciu następującej reguły CSS:

```
div {
   width : 300px;
   height : 300px;
   padding : 20px;
   border : 10px solid black;
   margin: 20px;
}
```

Warto zwrócić uwagę na właściwość `margin: 20px`. Jeśli chcesz, by w wyliczanej szerokości i wysokości elementu zostały uwzględnione te marginesy, musisz wywołać funkcje `outerWidth()` oraz `outerHeight()` w następujący sposób:

```
var divW = $('div').outerWidth(true); //400
var divH = $('div').outerHeight(true); //400
```

To, których funkcji należy użyć, zależy od tego, co chcemy osiągnąć. Załóżmy, że chcemy zasłonić czarnym prostokątem pewien tekst wyświetlony na stronie — na przykład odpowiedź na pytanie quizowe — a później go wyświetlić. Jednym z potencjalnych rozwiązań będzie zasłonięcie odpowiedzi prostokątem z czarnym tłem. W takim przypadku możemy użyć funkcji `width()` oraz `height()`, by określić wymiary samego tekstu (z pominięciem wypełnień i obramowań), i na podstawie pobranych wartości określić wymiary prostokąta, który następnie wyświetlimy nad tekstem.

A teraz dla odmiany załóżmy, że tworzymy własną wersję znanej gry Pong, w której niewielka piłeczka odbija się od krawędzi pola gry. Oczywiście, będziemy chcieli, by piłeczka cały czas pozostawała wewnątrz wyznaczonego obszaru (znacznika `<div>`, który najprawdopodobniej będzie miał wyświetlone obramowanie). W tym przypadku potrzebna jest znajomość wymiarów całego obszaru wewnątrz obramowań, co pozwoli upewnić się, że animowana piłeczka nie „wyleci" poza element oraz jego krawędzie. W takim przypadku powinniśmy skorzystać z funkcji `innerHeight()` oraz `innerWidth()`, gdyż piłeczka może się znaleźć w dowolnym miejscu wewnątrz pudełka elementu, nawet jeśli będzie on miał wypełnienie.

Uwaga: Funkcji `innerHeight()`, `innerWidth()`, `outerHeight()` oraz `outerWidth()` nie należy stosować podczas określania wymiarów okna przeglądarki (z selektorem `$(window)`) czy dokumentu (z selektorem `$(document)`). W tych dwóch przypadkach można korzystać wyłącznie z funkcji `height()` i `width()`.

Określanie położenia elementu na stronie

Znajomość położenia elementu na stronie przydaje się bardzo często, na przykład chcemy wyświetlić nad obrazkiem etykietę ekranową, kiedy użytkownik wskaże go myszą. Położenie tej etykiety powinno być zależne od położenia obrazka na stronie — oznacza to, że musimy najpierw określić położenie obrazka, a dopiero potem na jego podstawie wyliczyć współrzędne miejsca, gdzie ma się pojawić etykieta. Biblioteka jQuery udostępnia kilka funkcji ułatwiających określanie położenia elementów na stronie. Oto one.

- Funkcja `offset()`. Jej wywołanie zwraca obiekt zawierający właściwości `top` i `left`, określające odpowiednio położenie lewego, górnego wierzchołka elementu od górnej oraz lewej krawędzi dokumentu. Przykładowo załóżmy, że kiedy użytkownik wskaże myszą obrazek, chcemy wzdłuż jego górnej krawędzi wyświetlić opis. W takim przypadku musimy znać położenie obrazka. Załóżmy dodatkowo, że obrazek ten ma identyfikator `captionImage`. Współrzędne określające jego położenie można pobrać przy użyciu następującego wywołania:

  ```
  var imagePosition = $('#captionImage').offset();
  ```

  W efekcie, w zmiennej `imagePosition` zostaną zapisane współrzędne obrazka. Są one zapisane w obiekcie JavaScript, z którego można je odczytać, stosując zapis z kropką, opisany na stronie 82. Współrzędna pozioma jest zapisana we właściwości `left`, natomiast współrzędna pionowa — we właściwości `top`:

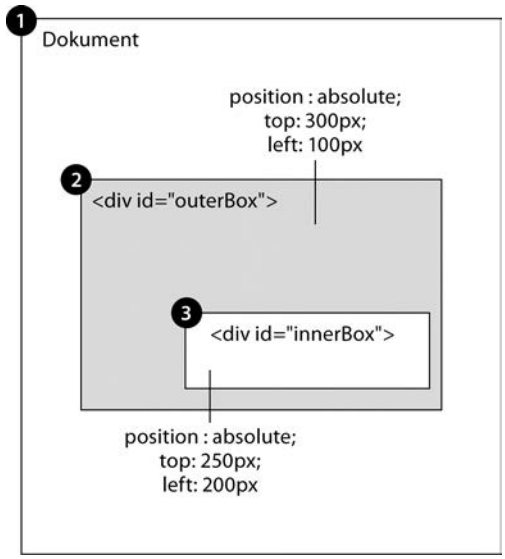
  ```
  imagePosition.top  // liczba pikseli od górnej krawędzi dokumentu
  imagePosition.left  // liczba pikseli od lewej krawędzi dokumentu
  ```

  Załóżmy teraz, że chcemy użyć tych informacji, by wyświetlić na stronie znacznik `<div>` o identyfikatorze `caption`. Możemy użyć funkcji `.css()` jQuery (patrz strona 155), by określić jego właściwości CSS `top`, `left` oraz `position`, i tym samym wyświetlić go w odpowiednim miejscu strony:

  ```
  $('#caption').css({
      'position' : 'absolute',
      'left' : imagePosition.left,
      'top' : imagePosition.top
  });
  ```

Uwaga: Funkcje `offset()` oraz `position()` zwracają współrzędne wyrażone w pikselach nawet wtedy, kiedy położenie elementu na stronie zostanie określone przy użyciu takich jednostek jak em lub wartości procentowe.

- Funkcja `position()`. Wywołanie tej funkcji zwraca obiekt zawierający współrzędne elementu liczone względem jego pierwszego przodka, w którego stylach CSS została określona wartość właściwości `display`. Zrozumienie tego wcale nie jest łatwe, posłużymy się zatem przykładem dwóch elementów `div` przedstawionych na rysunku 10.7. Oba te elementy zostały umiejscowione w sposób bezwzględny; położenie elementu `outerBox` jest określone względem dokumentu, natomiast położenie elementu `innerBox`, którego kod HTML jest umieszczony wewnątrz elementu `outerBox`, względem zewnętrznego znacznika `<div>`. Położenie elementu zewnętrznego jest określane względem dokumentu, gdyż nie jest on umieszczony wewnątrz żadnego innego znacznika HTML, w którym



Rysunek 10.7. *Biblioteka jQuery udostępnia dwie funkcje pozwalające na określanie współrzędnych elementu na stronie. Gdy na stronie jeden element umiejscowiony bezwzględnie (#innerBox) zostanie umieszczony wewnątrz innego elementu umiejscowionego bezwzględnie (#outerBox), funkcje te zwrócą różne wyniki*

właściwości CSS `display` została przypisana wartość `absolute`, `relative` bądź `fixed`. W przypadku tego elementu funkcja `position()` zwróci dokładnie takie same wyniki, co funkcja `offset()`, a zatem wywołanie:

```
$('#outerBox').position()  // { left : 100, top : 300 }
```

zwróci obiekt, którego właściwość `left` będzie miała wartość `100`, a właściwość `top` wartość `300`. Takie wartości zostały podane w regule stylów dla tego elementu.

Jednak w przypadku wewnętrznego znacznika `<div>` — o identyfikatorze `inner ↪Box` — którego położenie jest określane względem elementu zewnętrznego, wywołania funkcji `offset()` i `position()` zwrócą inne wyniki:

```
$('#innerBox').offset() // { left : 300, top : 550 }
$('#innerBox').position() // { left : 200, top : 250 }
```

Teraz funkcja `offset()` zwróci współrzędne liczone względem całego dokumentu, czyli 300 pikseli na prawo od lewej krawędzi dokumentu i 550 pikseli poniżej jego górnej krawędzi. Natomiast funkcja `position()` zwróci wartości podane w regule stylów CSS dla tego elementu.

Zazwyczaj bardziej przydatna z tych dwóch funkcji jest `offset()`, gdyż pozwala określić położenie elementu w odniesieniu do całej strony i dostarcza informacje potrzebne do wyznaczenia współrzędnych elementu względem innego elementu strony.

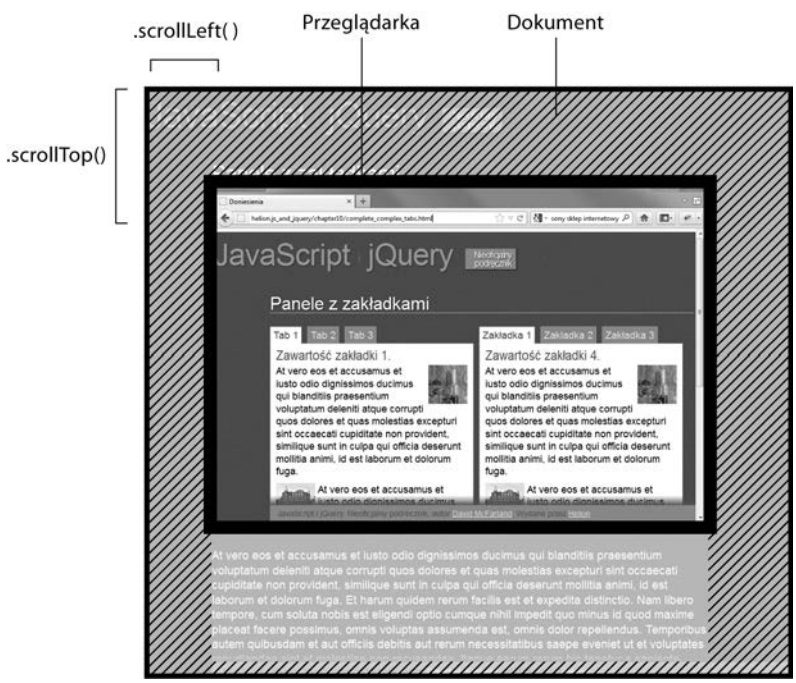
Wskazówka: Funkcji `offset()` można także używać do ustawiania położenia elementu na stronie. Wystarczy w tym celu przekazać w jej wywołaniu obiekt, co pokazano na poniższym przykładzie:

```
$('#element').offset({
    left : 100,
    top : 200
});
```

W tym przypadku współrzędne muszą być określane w pikselach — nie można stosować innych jednostek, takich jak `em` (na przykład `20em`) lub wartości procentowe (na przykład 20%).

Uwzględnianie przewinięcia strony

Strony WWW bardzo często są większe od okna przeglądarki, w którym są prezentowane: dokumenty HTML zawierające bardzo obszerną zawartość niejednokrotnie są wyższe, a zdarza się także, że i szersze od okna przeglądarki. Aby wtedy przejrzeć całą zawartość strony, użytkownik musi ją przewijać (patrz rysunek 10.8). Kiedy użytkownik przewija stronę, część jej treści przestaje być widoczna. Przykładowo strona przedstawiona na rysunku 10.8 została przewinięta nieco w dół i w prawo, co sprawia, że jej fragmenty przy górnej i lewej krawędzi nie są widoczne. Oznacza to, że górny, lewy wierzchołek okna przeglądarki nie będzie się pokrywał z górnym, lewym wierzchołkiem dokumentu. Jeśli w takim przypadku spróbujemy wyświetlić jakiś element, na przykład animowany pasek reklamowy, u góry ekranu, po przypisaniu właściwościom `top` i `left` wartości `0` pojawią się problemy. Będą one spowodowane tym, że element został umieszczony w lewym, górnym wierzchołku dokumentu, lecz poza fragmentem strony, który jest aktualnie widoczny w oknie przeglądarki.



Rysunek 10.8. *Często zdarza się, że wielkość strony wykracza poza wymiary okna przeglądarki. W takich przypadkach użytkownik musi ją przewijać, by wyświetlić początkowo niewidoczne fragmenty. Niemniej jednak, choć w danej chwili widać tylko część dokumentu, przeglądarka zawsze dysponuje całą jego zawartością*

Na szczęście, biblioteka jQuery udostępnia dwie funkcje pozwalające na określenie, o jaki dystans strona została przewinięta w pionie oraz w poziomie (inaczej mówiąc, zwracają one liczby określające w pikselach, jaki fragment dokumentu znajduje się powyżej górnej krawędzi okna przeglądarki oraz poza jego lewą krawędzią). Poniższy fragment kodu pozwala określić wysokość fragmentu dokumentu umieszczonego nad górną krawędzią okna przeglądarki:

```
$(document).scrollTop()
```

By natomiast określić szerokość obszaru dokumentu umieszczonego poza lewą krawędzią okna, można użyć następującego wywołania:

```
$(document).scrollLeft()
```

Obie te funkcje zwracają wielkości liczbowe wyrażone w pikselach, z których można skorzystać podczas wyliczania współrzędnych elementów na stronie. Jeśli na przykład chcemy wyświetlić okienko na środku strony, nawet jeśli została ona nieco przewinięta w pionie, trzeba będzie określić, o jaki dystans została przewinięta, i przesunąć wyświetlane okienko o odpowiedni odcinek w dół strony. Dużą ostrożność należy także zachować w przypadku wyświetlania etykietek ekranowych na przewiniętych stronach — bardzo łatwo doprowadzić do sytuacji, w której etykietka będzie wyświetlana w obszarze strony, lecz poza jej fragmentem, który w danej chwili jest widoczny w oknie przeglądarki. Dlatego też w 12. kroku opisu kolejnego przykładu, zamieszczonym na stronie 349, zobaczysz, jak skorzystać z funkcji `scrollTop()`, by chronić się przed wyświetlaniem etykietek nad górną krawędzią prezentowanego w przeglądarce obszaru strony.

Dodawanie etykietek ekranowych

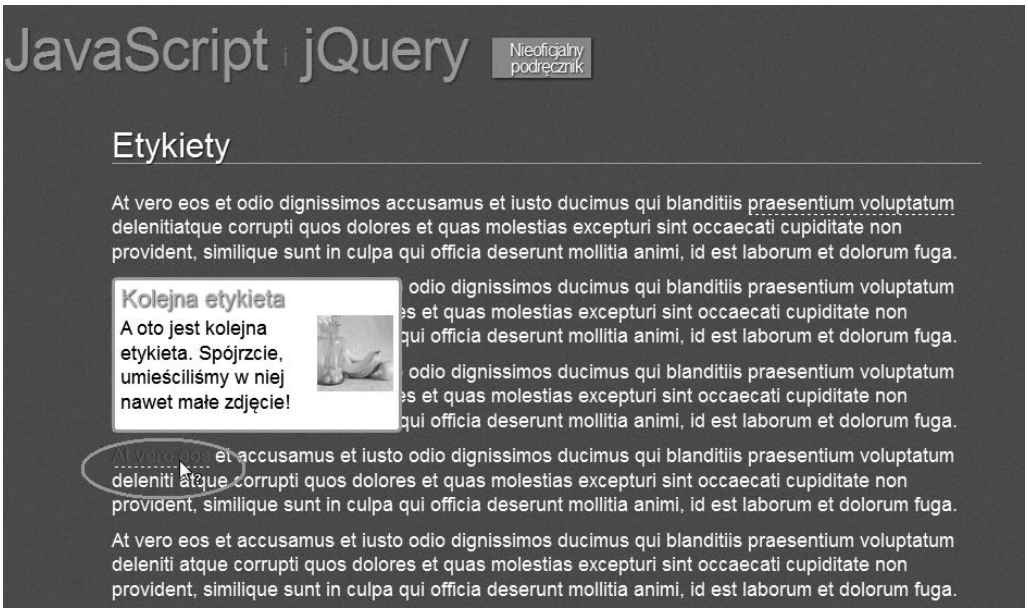
Etykietki są często stosowanym sposobem prezentowania informacji uzupełniających. Są to niewielkie, wyskakujące okienka, wyświetlane po wskazaniu myszą wybranego elementu strony — odnośnika, słowa, obrazka i tym podobnych. Często są one stosowane do wyświetlania definicji słowa, podpisu pod zdjęciem, a nawet bardziej szczegółowych informacji, takich jak czas, koszt oraz lokalizacja jakiegoś zdarzenia.

Podstawowa zasada działania etykietek ekranowych jest bardzo prosta: wskazujemy wybrany element myszą i wyświetlamy inny element (zazwyczaj będzie nim znacznik `<div>`) w pobliżu wskazanego; po usunięciu wskaźnika myszy z obszaru elementu etykietka znika. Poznałeś już sposoby tworzenia kodu JavaScript niezbędnego do zaimplementowania takiego rozwiązania, zatem bez przeszkód możemy opisać cały proces tworzenia etykietek krok po kroku.

Wykorzystamy w nim kody JavaScript, CSS oraz HTML w celu uzyskania ostatecznego efektu przedstawionego na rysunku 10.9. Kod HTML posłuży do zdefiniowania zarówno elementu wyzwalającego wyświetlenie etykiety (czyli tego, który należy wskazać myszą), jak i samej etykiety. Podstawowe aspekty wyglądu etykiety ustalimy przy użyciu arkusza stylów CSS, natomiast kod JavaScript pozwoli ukryć etykietę w momencie wczytywania strony. Dodatkowo określimy także procedurę obsługi zdarzeń `hover` i dodamy ją do wszystkich elementów strony, dla których chcemy wyświetlać etykiety.

Kod HTML

Etykieta ekranowa składa się z dwóch elementów: samej etykiety, czyli elementu wyświetlanego na stronie, gdy użytkownik wskaże myszą element wyzwalający, oraz tegoż elementu wyzwalającego, którym może być dowolny inny element strony, taki jak obrazek, odnośnik, nagłówek bądź znacznik `<span>`.



Rysunek 10.9. *Etykiety ekranowe są niewielkimi okienkami zawierającymi dodatkowe informacje. Są one wyświetlane, kiedy użytkownik wskaże myszą określony element strony, tak zwany element wyzwalający (został zakreślony na rysunku), oraz ukrywane, gdy wskaźnik myszy zostanie usunięty z obszaru tego elementu*

Etykieta jest znacznikiem `<div>` należącym do klasy `tooltip` i posiadającym unikalny identyfikator. Wewnątrz tego znacznika można umieścić dodatkowy kod HTML, chociażby nagłówki, akapity tekstu i obrazki. Nie należy jednak umieszczać w nich odnośników, gdyż nie będą działały prawidłowo — przesunięcie wskaźnika myszy do obszaru etykiety w celu kliknięcia odnośnika spowoduje usunięcie go z obszaru elementu wyzwalającego, co sprawi, że etykieta zniknie.

Oto bardzo prosty kod etykiety ekranowej:

```
<div class="tooltip" id="aardvarkTooltip">
    <h2>Mrówkojad</h2>
    <p>Średniej wielkości ssak ryjący, prowadzący nocny tryb życia;
występuje
        głównie w Afryce.</p>
</div>
```

Choć znaczniki `<div>` etykiet można umieścić w dowolnym miejscu kodu HTML strony (w końcu przez znaczną większość czasu i tak są one niewidoczne), jednak najlepszym rozwiązaniem jest umieszczenie ich tuż przed zamykającym znacznikiem `</body>`. To optymalne miejsce, gdyż pozwala uniknąć wszelkich dziwnych problemów związanych ze sposobem prezentacji treści, które mogłyby powstać po umieszczeniu etykiet wewnątrz innego elementu, pozycjonowanego względnie lub bezwzględnie.

Elementem wyzwalającym może być dowolny element strony — znacznik `<h1>`, `<div>` lub `<img>`. Jeśli chcemy, by elementem wyzwalającym było słowo lub grupa słów, trzeba je umieścić wewnątrz znacznika `<span>`. Do prawidłowego działania elementu wyzwalającego niezbędne są dwie informacje.

- **Nazwa klasy**. Wszystkie elementy wyzwalające muszą należeć do tej samej klasy, na przykład `trigger`. Nazwa klasy jest niezbędna, by kod JavaScript mógł odszukać wszystkie te elementy i dodać do nich procedury obsługi zdarzeń odpowiadające za wyświetlanie i ukrywanie etykiet.
- **Dane identyfikujące etykietę**. Każdy element wyzwalający jest skojarzony z jedną etykietą. Etykieta ta jest znacznikiem `<div>`, który zazwyczaj jest niewidoczny, lecz można go wyświetlić, gdy użytkownik umieści wskaźnik myszy na odpowiednim elemencie wyzwalającym. Każda etykieta musi posiadać unikalny identyfikator, a my musimy dysponować jakimś sposobem skojarzenia elementu wyzwalającego z odpowiednim znacznikiem `<div>` etykiety, dzięki któremu będziemy wiedzieli, jaki znacznik `<div>` wyświetlić, kiedy użytkownik wskaże myszą element wyzwalający. Prostym rozwiązaniem jest umieszczenie identyfikatora etykiety w jakimś atrybucie znacznika elementu wyzwalającego (warto dodać przed nim znak #, dzięki czemu pobranie etykiety przy użyciu jQuery będzie naprawdę bardzo proste). Język HTML5 pozwala na dodawanie danych do znaczników HTML, jeśli nazwa atrybutu rozpoczyna się od ciągu znaków `data-`.

  Przykładowo załóżmy, że elementem wyzwalającym dla naszej etykiety jest słowo *mrówkojad*. Wskazanie go myszą powinno spowodować wyświetlenie etykiety (czyli, w rzeczywistości, znacznika `<div>` o identyfikatorze `aardvarkTooltip`). Element wyzwalający możemy utworzyć, umieszczając wybrane słowo wewnątrz znacznika `<span>`, co pokazano na poniższym przykładzie:

  ```
  <span class="trigger" data-tooltip="#aardvarkTooltip">mrówkojad</span>
  ```

  Niestandardowe atrybuty danych standardu HTML5 są naprawdę rewelacyjne. Pozwalają projektantom na umieszczanie w znacznikach przeróżnych informacji, które później można pobierać przy użyciu kodu JavaScript. Szczegółowy opis tych atrybutów można znaleźć na stronie *http://html5doctor.com/html5-custom-data-attributes/*.

  Jeśli korzystasz z języków XHTML 1 bądź HTML 4.01 i obawiasz się problemów zgodności ze standardami, nie będziesz mógł używać takich atrybutów danych. Zamiast tego możesz wykorzystać jeden z prawidłowych atrybutów języka HTML 4, na przykład `title`:

  ```
  <span class="trigger" title="#aardvarkTooltip">mrówkojad</span>
  ```

  Takie zastosowanie atrybutu `title` nie jest — co prawda — zgodne z jego przeznaczeniem i niektórzy projektanci stron mogą nie pochwalać takiego rozwiązania. Najprościej będzie skorzystać z języka HTML5 i użyć atrybutów danych.

Na jednej stronie można umieścić dowolną liczbę elementów wyzwalających oraz skojarzonych z nimi etykiet.

Kod CSS

Każdy znacznik `<div>` etykiety należy do klasy `tooltip`, a zatem dodanie do używanego na stronie arkusza stylów reguły z selektorem `.tooltip` pozwoli określić ich ogólny wygląd (na przykład kolor tła, obramowanie, szerokość i tak dalej). Oto wersja tej reguły umieszczona w przykładowym pliku dołączonym do książki:

```
.tooltip {
    width: 25%;
    padding: 5px;
    background-color: white;
    border: 3px solid rgb(195,151,51);
    border-radius : 5px;
}
```

Bez zastosowania dodatkowych stylów użytkownicy nie będą w stanie określić, że elementy wyzwalające pełnią szczególną rolę. Ma to szczególne znaczenie w przypadkach, gdy będziemy dodawać etykietę ekranową do wybranego słowa należącego do większego akapitu tekstu. Można utworzyć specjalny styl CSS, który wyróżni elementy wyzwalające — doda do nich obramowanie, kolor tła i tym podobne. Poniższa, prosta reguła dodaje dolną krawędź do wszystkich elementów należących do klasy `trigger`:

```
.trigger {
    border-bottom: 1px dashed white;
    cursor : help;
}
```

Szczególnie użyteczna jest właściwość CSS `cursor` — kontroluje ona postać wskaźnika myszy, w czasie gdy będzie się znajdował w obszarze elementu. Kiedy wskaźnik myszy zostanie umieszczony w obszarze tekstu, wygląda jak kursor do zaznaczania, jednak tę postać wskaźnika można zmienić — użycie wartości `help` sprawi, że będzie wyglądał jak znak zapytania (co jest dobrym rozwiązaniem, gdy etykieta zawiera definicję jakiegoś terminu), a wartości `pointer` — że będzie wyglądał jak dłoń z wyprostowanym palcem wskazującym, czyli w sposób standardowy dla wskaźnika umieszczonego na odnośniku. Informacje o pozostałych dostępnych kształtach wskaźnika myszy można znaleźć na stronie *www.w3schools.com/cssref/pr_class_cursor.asp*.

Oprócz tego, do arkusza stylów można dodać regułę z pseudoelementem `:hover`, określającą postać elementu wyzwalającego w przypadku umieszczenia w jego obszarze wskaźnika myszy; oto przykład takiej reguły:

```
.trigger:hover {
    color: rgb(255,0,0);
}
```

Kod JavaScript

Najprościej rzecz ujmując, etykieta ekranowa powinna zostać wyświetlona, gdy użytkownik umieści wskaźnik myszy w obszarze elementu wyzwalającego, i ma zniknąć, kiedy wskaźnik zostanie z tego elementu usunięty. Już wcześniej, w rozdziale 6., dowiedziałeś się, jak można wyświetlać i ukrywać elementy. Jednak w tym przykładzie zwyczajne wyświetlenie i ukrycie elementu to za mało. Kluczową czynnością związaną z wyświetleniem każdej etykiety jest umieszczenie jej w pobliżu elementu wyzwalającego. To z kolei wiąże się z koniecznością użycia funkcji jQuery w celu określenia szerokości, wysokości oraz wymiarów tego elementu. I to jest najtrudniejsze zadanie. Aby nieco lepiej wyjaśnić wykonywane czynności, w tym przykładzie zdecydowaliśmy się podzielić opis tworzonego kodu na trzy części.

1. **Ukrycie etykiet**.

   W momencie wczytywania strony wszystkie znajdujące się na niej etykiety (czyli znaczniki `<div>` umieszczone na samym końcu jej treści) powinny być ukryte. Oczywiście, można by to zrobić przy użyciu odpowiedniego stylu CSS jeszcze przed wczytaniem strony, jednak w takim przypadku żaden użytkownik, przeglądający stronę za pomocą przeglądarki, w której została wyłączona obsługa języka JavaScript, nie byłby w stanie uzyskać dostępu do treści etykiet. Jeśli informacje zamieszczone w etykietach nie są bardzo ważne i można zaakceptować fakt, że niektórzy użytkownicy (w tym także mechanizmy wyszukiwawcze) ich nie zobaczą, to proszę bardzo — możesz ukryć etykiety poprzez zastosowanie odpowiedniego stylu CSS:

   ```
   .tooltip {
       display: none;
   }
   ```

2. **Dodanie do elementów wyzwalających procedury obsługi zdarzeń `hover`**.

   Czynność ta ma kluczowe znaczenie dla działania etykiet. Kiedy użytkownik wskaże myszą element wyzwalający, muszą zostać wykonane dwie operacje.

   - Musi zostać wyświetlony znacznik `<div>` odpowiedniej etykiety.
   - Znacznik ten należy umieścić w pobliżu elementu wyzwalającego. W tym celu trzeba określić bieżące położenie tego elementu. Dodatkowo trzeba się upewnić, że etykieta nie przesłoni tego elementu oraz że nie będzie wystawać poza widoczny obszar okna przeglądarki.

3. **Dodanie do elementów wyzwalających procedury obsługi zdarzeń `mouseover`.**

   To bardzo proste zadanie — wystarczy ukryć znacznik `<div>`, kiedy użytkownik usunie wskaźnik myszy z jego obszaru.

Aby przekonać się, jak ten program działa, czas przejść do przykładu, w którym utworzysz swoje własne etykiety ekranowe.

Przykład — etykiety ekranowe

Utworzenie prostych etykiet ekranowych naprawdę nie jest trudnym zadaniem. W tym przykładzie szczegółowo opiszemy cały ten proces. Do pracy nad tym przykładem możesz wykorzystać dowolny edytor HTML.

Uwaga: Informacje dotyczące pobierania przykładów do książki można znaleźć na stronie 43.

---

1. **W edytorze HTML otwórz plik *tooltip.html* umieszczony w katalogu *R10*.**

   W tym pliku został już umieszczony wewnętrzny arkusz stylów CSS, zawierający kilka reguł określających wygląd elementów wyzwalających oraz etykiet. Są to dokładnie te same style, które zostały przedstawione we wcześniejszej części rozdziału, na stronie 342. Jednak w kodzie strony nie ma jeszcze żadnych etykiet — ich kod będziesz musiał dodać.

2. **Odszukaj zamykający znacznik `</body>` umieszczony na samym końcu pliku i powyżej niego dodaj poniższy kod HTML tworzonej etykiety:**

```
<div class="tooltip" id="tip1">
    <h2>Etykieta</h2>
    <p>To jest tekst etykiety. Został on umieszczony wewnątrz
    ↪znacznika div,
    dzięki temu można tu umieścić prawie wszystko.</p>
</div>
```

   Najważniejszym elementem tego kodu jest zewnętrzny znacznik `<div>`. Użyliśmy w nim klasy `tooltip`, co jest niezbędne zarówno po to, by określić postać etykiety, jak i ze względu na kod programu, który utworzysz już niebawem. Dodatkowo w znaczniku umieściliśmy unikalny identyfikator, który pozwoli zidentyfikować daną etykietę i skojarzyć ją z elementem wyzwalającym, jaki utworzysz w następnym kroku. Wewnątrz etykiety możesz umieścić dowolny kod HTML — w tym przypadku jest to nagłówek oraz jeden akapit tekstu.

3. **Odszukaj znacznik `<p>` umieszczony w kodzie bezpośrednio poniżej znacznika `<h1>Etykiety ekranowe</h1>`, mniej więcej w połowie wielkości pliku. Wybierz kilka słów i zapisz je pomiędzy znacznikami `<span>`, co pokazano na poniższym przykładzie:**

```
<span class="trigger" data-tooltip="#tip1">accusamus et iusto</span>
```

   Zastosowanie klasy `trigger` identyfikuje ten znacznik `<span>` jako element wyzwalający etykiety. Jedna z reguł arkusza stylów umieszczonego w sekcji nagłówka strony formatuje dowolny znacznik należący do tej klasy w szczególny sposób. Dodatkowo umieszczony w kodzie znacznika atrybut `data-tooltip` identyfikuje etykietę, z którą dany element wyzwalający jest skojarzony.

   W następnym kroku dodasz do strony kolejną etykietę.

4. **Tuż poniżej znacznika `<div>` dodanego w kroku 2. (lecz wciąż przed zamykającym znacznikiem `</body>`) dodaj kolejny `<div>`:**

```
<div class="tooltip" id="tip2">
    <h2>Kolejna etykieta</h2>
    <p><img src="../_images/small/yellow_h.jpg"
    alt="yellow" width="70" height="70" class="imgRight">
    A oto jest kolejna etykieta. Spójrzcie, umieściliśmy w niej nawet
    ↪małe zdjęcie!</p>
</div>
```

   Dodałeś właśnie drugą etykietę. Zwróć uwagę, że użyliśmy w niej tej samej nazwy klasy, co w poprzedniej, czyli `tooltip`. Natomiast identyfikator tego znacznika jest unikalny — `tip2`. Dodatkowo wewnątrz etykiety umieściliśmy zdjęcie. Teraz musisz utworzyć element wyzwalający dla tej etykiety.

5. **Wybierz kolejnych kilka słów z jakiegoś akapitu tekstu i ponownie umieść je wewnątrz znacznika `<span>`:**

```
<span class="trigger" data-tooltip="#tip2">At vero eos</span>
```

   Zwróć uwagę, by podać identyfikator drugiej etykiety — `#tip2`. Nic nie stoi na przeszkodzie, abyś dodał kolejne etykiety i elementy wyzwalające; pamiętaj jedynie, żeby każda z etykiet miała unikalny identyfikator i podaj ten identyfikator w atrybucie `data-tooltip` elementu wyzwalającego skojarzonego z daną etykietą.

   Teraz nadszedł czas, by zająć się pisaniem kodu JavaScript. Do strony został już dołączony plik biblioteki jQuery oraz wywołanie funkcji `$(document).ready()`.

Kolejnym zadaniem będzie zatem ukrycie wszystkich etykiet w momencie wczytywania strony.

6. **Kliknij pusty wiersz wewnątrz funkcji `$(document).ready()` i wpisz w nim:**

```
$('.tooltip').hide();
```

Ten wiersz kodu jest bardzo prosty. Wywołanie funkcji `hide()` (opisanej na stronie 198) powoduje ukrycie wszystkich etykiet, dzięki czemu użytkownik nie zobaczy ich zaraz po wyświetleniu strony. Oczywiście, chcemy, by konkretne etykiety pojawiały się, kiedy użytkownik wskaże myszą odpowiednie elementy wyzwalające, a zatem kolejnym krokiem będzie pobranie wszystkich elementów wyzwalających i dodanie do nich procedury obsługi zdarzeń `mouseover`.

7. **Poniżej kodu dodanego w poprzednim kroku dodaj kolejny fragment:**

```
$('.trigger').mouseover(function() {

}); // koniec funkcji mouseover
```

To szkielet kodu procedury obsługi zdarzeń, podobny do tego, który został opisany na stronie 174. W tym przypadku pobieramy wszystkie elementy należące do klasy `trigger` i dodajemy do nich procedurę obsługi zdarzeń `mouseover`. Funkcja ta stanowi kluczowy element naszego kodu obsługującego etykiety ekranowe, gdyż to właśnie ona będzie kontrolować wyświetlanie oraz odpowiednie rozmieszczanie etykiet na ekranie. Precyzyjne określenie miejsca, w którym ma zostać wyświetlona etykieta, jest dosyć złożone i będzie wymagało pobrania wielu różnych informacji. Dlatego też na samym początku tej funkcji zdefiniujesz kilka zmiennych.

8. **Wewnątrz funkcji anonimowej dodanej w kroku 7. wpisz poniższy, wyróżniony pogrubieniem fragment kodu:**

```
1   $('.trigger').mouseover(function() {
2       var ttLeft,
3           ttTop,
4   }); // koniec funkcji mouseover
```

Zaczynamy od utworzenia dwóch zmiennych — `ttLeft` zawiera poziomą współrzędną etykiety, natomiast `ttTop` — współrzędną pionową. Początkowo obie te zmienne są puste, ponieważ jeszcze nie wiemy, jakie mają być ich wartości.

Ten sposób tworzenia zmiennych może Ci się wydawać nieco dziwny; gdyż zapewne jesteś przyzwyczajony do tworzenia dwóch zmiennych przy wykorzystaniu dwóch słów kluczowych `var`, w sposób pokazany poniżej:

```
var ttLeft;
var ttTop;
```

Takie rozwiązanie jest całkowicie prawidłowe, jednak podczas tworzenia większej liczby zmiennych często stosuje się technikę wykorzystującą tylko jedno słowo kluczowe `var`, za którym są podawane nazwy wszystkich zmiennych oddzielone przecinkami. Dzięki temu możemy uniknąć konieczności wielokrotnego wpisywania słowa `var`. Przecinek umieszczony na końcu wiersza 3. nie jest żadnym błędem — już zaraz dodasz do kodu kolejne zmienne.

9. **Do kodu programu dodaj kolejną zmianą (umieszczoną w wierszu 4.):**

```
1   $('.trigger').mouseover(function() {
2       var ttLeft,
```

```
3           ttTop,
4           $this=$(this),
5   }); // koniec funkcji mouseover
```

W tym przypadku wyrażenie `$(this)` odwołuje się do elementu wyzwalającego, natomiast cała instrukcja przypisania `$this = $(this)` pozwala zapisać odwołanie do tego elementu w zmiennej. Dokładnie to samo zrobiliśmy w przykładzie pokazującym sposób tworzenia panelu kart, w jego 3. kroku (opisanym na stronie 321). Dalej w kodzie tej funkcji będziemy wielokrotnie odwoływali się do elementu wyzwalającego i gdybyśmy za każdym razem używali wywołania `$(this)`, zmuszalibyśmy interpreter JavaScriptu przeglądarki do wielokrotnego wykonywania kodu funkcji jQuery, co stanowiłoby duże marnotrawstwo czasu i mocy procesora. Gdy zamiast tego zapiszemy wynik wywołania `$(this)` w zmiennej, funkcja jQuery konieczna do pobrania elementu wyzwalającego zostanie wykonana tylko raz, przez co nasz program stanie się bardziej efektywny (więcej informacji dotyczących zalet zapisywania elementów pobieranych przy użyciu jQuery w zmiennych można znaleźć na stronie 422).

Kolejnym krokiem będzie pobranie etykiety skojarzonej z danym elementem wyzwalającym.

Wskazówka: W przypadku zapisywania elementów pobieranych przy użyciu jQuery w zmiennych często stosowaną praktyką jest umieszczenie na początku nazwy zmiennej znaku `$`:

```
var $banner = $('#banner');
```

Oczywiście, nie jest to konieczne; zmienna `var banner = $('#banner')` będzie działać równie dobrze. Jednak znak `$` przypomina o tym, że zmienna zawiera selekcję jQuery, a nie jakiekolwiek inne, zwyczajne wartości, takie jak łańcuchy znaków lub liczby.

---

10. **Dodaj kolejną zmienną (umieszczoną w wierszu 5.):**

```
1   $('.trigger').mouseover(function() {
2       var ttLeft,
3           ttTop,
4           $this=$(this),
5           $tip = $($this.attr('data-tooltip')),
6   }); // koniec funkcji mouseover
```

Zmienna `$tip` zawiera pobrany przy użyciu jQuery znacznik etykiety. Wywołanie `$($this.attr('data-tooltip'))` spełnia kilka zadań, więc rozbijemy je na elementy i dokładnej przeanalizujemy. Fragment umieszczony wewnątrz wywołania jQuery `$()` — `$this.attr('data-tooltip')` — korzysta z funkcji `attr()`, by pobrać wartość atrybutu `data-tooltip` elementu wyzwalającego (pamiętaj, że to właśnie do niego odwołuje się zmienna `$this`). Innymi słowy, całe to wywołanie odwołuje się do aktualnego elementu wyzwalającego, odnajduje jego atrybut `data-tooltip` i pobiera jego wartość. Dla elementu wyzwalającego dodanego w kroku 3. wywołanie to zwróci łańcuch znaków `'#tip1'`; natomiast dla elementu dodanego w kroku 5. byłaby to wartość `'#tip2'`.

Po pobraniu wartości atrybutu `data-tooltip` jest ona używana w wywołaniu funkcji jQuery `$()` (zewnętrzna funkcja wywołania zapisanego w wierszu 5. powyższego kodu). Innymi słowy, w rzeczywistości kod ten sprowadza się do wywołania o postaci `$('#tip1')` lub `$('#tip2')`. Hej, ale przecież Ty to znasz! To jest zwyczajny sposób pobierania elementów przy użyciu biblioteki jQuery!

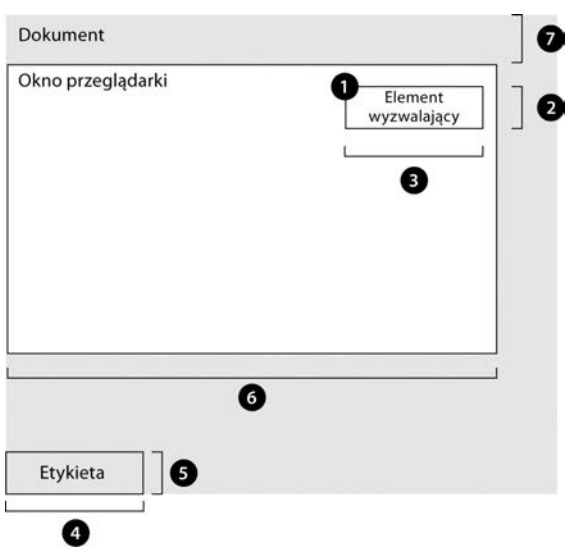
Po wykonaniu 5. wiersza powyższego fragmentu kodu w zmiennej o nazwie `$tip` będzie zapisany obiekt jQuery z pobraną odpowiednią etykietą. Możesz go użyć, by wyświetlić, animować lub określić położenie etykiety na stronie.

Kolejnym zadaniem jest zgromadzenie wszystkich informacji niezbędnych do określenia prawidłowego położenia etykiety na stronie.

11. **Do funkcji mouseover dodaj kod z wierszy od 6. do 12. poniższego fragmentu kodu:**

```
1   $('.trigger').mouseover(function() {
2       var ttLeft,
3           ttTop,
4           $this=$(this),
5           $tip = $($this.attr('data-tooltip')),
6           triggerPos = $this.offset(),
7           triggerH = $this.outerHeight(),
8           triggerW = $this.outerWidth(),
9           tipW = $tip.outerWidth(),
10          tipH = $tip.outerHeight(),
11          screenW = $(window).width(),
12          scrollTop = $(document).scrollTop();
13  }); // koniec funkcji mouseover
```

Dodane wiersze kodu pobierają i zapisują w zmiennych informacje o położeniu oraz wymiarach kilku elementów. Schemat przedstawiony na rysunku 10.10 pomoże Ci wyobrazić sobie i zrozumieć znaczenie każdej z tych wartości. Widać na nim całą stronę WWW (przedstawioną jako szary prostokąt), która jest większa od okna przeglądarki (zaznaczonego jako czarna ramka). Strona została nieco przewinięta w dół, zatem pewien jej fragment znalazł się powyżej górnej krawędzi okna przeglądarki. Co więcej, ponieważ strona jest zarówno dłuższa, jak i szersza od okna przeglądarki, zatem pewne jej fragmenty są także ukryte poza prawą oraz dolną krawędzią okna przeglądarki.



Rysunek 10.10. *Ten schemat prezentuje stronę WWW (szary obszar), która jest zarówno wyższa, jak i szersza od okna przeglądarki (oznaczonego jako czarny prostokąt). Użytkownik przeglądający tę stronę przewinął ją nieco ku dołowi, a zatem jej górna część jest niewidoczna, podobnie zresztą jak fragmenty położone przy jej prawej oraz dolnej krawędzi. Etykieta została przedstawiona jako niewielki prostokąt położony na samym dole strony, gdyż właśnie tam znajduje się bezpośrednio po wczytaniu strony. Dopiero w kroku 15. dokończysz pisanie kodu odpowiedzialnego za określenie właściwego położenia etykiety oraz umieszczenie jej w tym miejscu*

Wiersz 6. powyższego fragmentu kodu pobiera współrzędne elementu wyzwalającego (oznaczonego na rysunku 10.10 cyfrą 1). Ich znajomość jest niezbędna, gdyż to właśnie względem tego elementu musimy określić położenie etykiety.

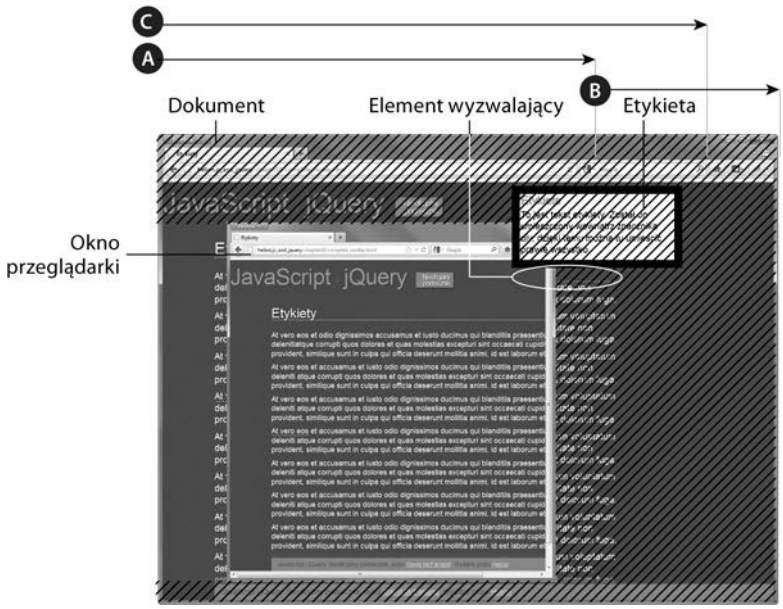
W wierszach 7. i 8. wywoływane są funkcje `outerHeight()` (patrz strona 335) oraz `outerWidth()` (patrz strona 335), które pozwalają pobrać odpowiednio wysokość (cyfra 2 na rys. 10.10) oraz szerokość (cyfra 3) elementu wyzwalacza (uwzględniając przy tym wielkości wypełnienia i obramowania). Kolejne wiersze — 9. i 10. — pobierają odpowiednio szerokość (cyfra 4) oraz wysokość (cyfra 5) etykiety. Ponieważ nie chcemy, by etykieta była wyświetlana poza oknem przeglądarki, zatem musimy także znać jego szerokość (pobieramy ją w wierszu 11. powyższego fragmentu kodu, a na rysunku 10.10 została ona oznaczona cyfrą 6) i wiedzieć, czy użytkownik nie przewinął strony w dół (a jeśli przewinął, to o ile; wiersz 12., cyfra 7). Nie możesz także zapomnieć o dodaniu średnika na końcu wiersza 12., gdyż właśnie w nim kończy się instrukcja `var` rozpoczęta w wierszu 2.

Być może zastanawiasz się, do czego są Ci potrzebne te wszystkie informacje. Czy nie byłoby łatwiej określić położenie elementu wyzwalającego, a następnie wyświetlić etykietę bezpośrednio nad nim? W większości przypadków takie rozwiązanie zdałoby egzamin, jednak istnieje kilka przypadków, w których nie działałoby prawidłowo. I tak w przypadku zilustrowanym na rysunku 10.11 element wyzwalający znajduje się w prawym, górnym wierzchołku okna przeglądarki, a fragment strony jest przewinięty i ukryty poza górną krawędzią okna. Gdybyśmy teraz umieścili etykietę bezpośrednio nad elementem wyzwalającym, jej znaczna część byłaby niewidoczna. Innymi słowy, nasz kod musi działać inteligentnie — powinien określić, czy umieszczenie etykiety nad elementem wyzwalającym nie sprawi, że jej część znajdzie się poza oknem przeglądarki. Gdyby faktycznie tak miało się stać, nasz kod musi wyznaczyć nowe położenie etykiety.

Trzeba zacząć od sprawdzenia, czy etykieta umieszczona bezpośrednio nad elementem wyzwalającym zmieści się w górnej części okna przeglądarki.

12. **Do tej samej funkcji dodaj wiersze od 13. do 17. z poniższego fragmentu kodu:**

```
1   $('.trigger').mouseover(function() {
2       var ttLeft,
3           ttTop,
4           $this=$(this),
5           $tip = $($this.attr('data-tooltip')),
6           triggerPos = $this.offset(),
7           triggerH = $this.outerHeight(),
8           triggerW = $this.outerWidth(),
9           tipW = $tip.outerWidth(),
10          tipH = $tip.outerHeight(),
11          screenW = $(window).width(),
12          scrollTop = $(document).scrollTop();
13          if (triggerPos.top - tipH - scrollTop > 0 ) {
14              ttTop = triggerPos.top - tipH - 10;
15          } else {
16              ttTop = triggerPos.top + triggerH +10 ;
17          }
18  }); // koniec funkcji mouseover
```



Rysunek 10.11. *Podczas dynamicznego rozmieszczania elementów na stronie należy zachować dużą ostrożność. Nigdy nie będziemy wiedzieć, jakie będą wymiary okna przeglądarki, ani czy strona nie została w nim przewinięta. Jeśli czynniki te nie zostaną wzięte pod uwagę, bardzo łatwo można umieścić element w taki sposób, że częściowo lub nawet w całości znajdzie się poza widocznym obszarem strony (czyli w zakreskowanym obszarze rysunku)*

W tym krótkim fragmencie kodu dzieje się całkiem sporo, jednak jego analizę warto zacząć od przyjrzenia się, gdzie dokładnie chcemy umieścić etykietę w stosunku do elementu wyzwalającego. Normalnie wyświetlilibyśmy etykietę 10 pikseli ponad elementem wyzwalającym, aby go nie przesłaniała. Aby określić współrzędną pionową etykiety, należy zacząć od pobrania pionowej współrzędnej elementu wyzwalającego, następnie odjąć od niej wysokość etykiety, a wynik pomniejszyć o dodatkowe 10 pikseli. W ramach przykładu załóżmy, że element wyzwalający jest umieszczony 150 pikseli pod górną krawędzią strony, a etykieta ma wysokość 100 pikseli. Chcemy umieścić etykietę tak, aby nie przesłaniała elementu wyzwalającego, należy zatem wziąć jego współrzędną pionową — 150 — odjąć 100, a od uzyskanego wyniku (50) odjąć jeszcze 10 (zostawiając w ten sposób niewielki odstęp pomiędzy etykietą i elementem wyzwalającym). W rezultacie okazuje się, że etykieta powinna być umieszczona 40 pikseli poniżej górnej krawędzi dokumentu.

A co by się stało, gdyby element wyzwalający był umieszczony 10 pikseli poniżej górnej krawędzi dokumentu, a etykieta miała wysokość 100 pikseli? Gdybyśmy bezmyślnie skopiowali powyższe równanie, okazałoby się, że współrzędna pionowa etykiety wynosi –90 pikseli (10 – 100 = –90); innymi słowy, znalazłaby się ona ponad górną krawędzią dokumentu, czyli byłaby niewidoczna!

I właśnie w tym miejscu do akcji wkracza warunek umieszczony w wierszu 13. Od wartości pionowej współrzędnej elementu wyzwalającego odejmujemy wysokość etykiety oraz wielkość przewinięcia strony. Następnie sprawdzamy, czy uzyskany wynik jest większy od zera (gdyby był mniejszy, etykieta zostałaby umieszczona poza górną krawędzią okna przeglądarki). W tych obliczeniach musimy uwzględnić także przewinięcie strony, gdyż może się zdarzyć, że etykieta zmieści się na stronie powyżej elementu wyzwalającego, jeśli jednak strona zo-

stanie przewinięta, może się okazać, że tak umieszczona etykieta znalazłaby się poza obszarem strony widocznym w przeglądarce (właśnie taka sytuacja została przedstawiona na rysunku 10.11).

Jeśli ten warunek będzie spełniony, etykieta zostanie umieszczona nad elementem wyzwalającym (a wartość współrzędnej pionowej — `ttTop` — zostanie wyliczona w wierszu 14.). Jeśli jednak warunek nie zostanie spełniony, wykonany będzie wiersz 16., a etykieta pojawi się 10 pikseli poniżej dolnej krawędzi elementu wyzwalającego (jej współrzędną pionową wyliczamy poprzez pobranie współrzędnej pionowej lewego, górnego wierzchołka elementu wyzwalającego — `triggerPos.top` — i dodanie do niej jego wysokości — `triggerH`).

Kolejną czynnością będzie obliczenie poziomej współrzędnej etykiety.

Uwaga: W opisywanym tu przykładzie etykieta jest umieszczana ponad elementem wyzwalającym, jednak nie znaczy to wcale, że tak ma być. Nic nie stoi na przeszkodzie, byś zmienił kod skryptu i sprawdził, czy potrafisz wyświetlić etykietę poniżej elementu wyzwalającego bądź z jego prawej lub lewej strony.

13. **Do tej samej funkcji dodaj wiersz 18. poniższego fragmentu:**

```
1   $('.trigger').mouseover(function() {
2       var ttLeft,
3           ttTop,
4           $this=$(this),
5           $tip = $($this.attr('data-tooltip')),
6           triggerPos = $this.offset(),
7           triggerH = $this.outerHeight(),
8           triggerW = $this.outerWidth(),
9           tipW = $tip.outerWidth(),
10          tipH = $tip.outerHeight(),
11          screenW = $(window).width(),
12          scrollTop = $(document).scrollTop();
13          if (triggerPos.top - tipH - scrollTop > 0 ) {
14              ttTop = triggerPos.top - tipH - 10;
15          } else {
16              ttTop = triggerPos.top + triggerH +10 ;
17          }
18          var overFlowRight = (triggerPos.left + tipW) - screenW;
19  }); // koniec funkcji mouseover
```

Wyliczenie współrzędnej poziomej etykiety jest nieco bardziej złożone niż współrzędnej pionowej. W tym przypadku nie tylko musimy wiedzieć, czy fragment etykiety jest umieszczony poza prawą krawędzią okna przeglądarki, lecz także o ile poza nią wystaje. Przykładowo załóżmy, że współrzędna pozioma elementu wyzwalającego wynosi 850 pikseli (na rysunku 10.11 została oznaczona literą A), etykieta ma 250 pikseli szerokości (została oznaczona literą B), a okno przeglądarki ma szerokość 1000 pikseli (oznaczono ją literą C). Jeśli w takim przypadku etykieta zostanie wyświetlona w punkcie o współrzędnej poziomej 850 pikseli, jej prawa krawędź znajdzie się w miejscu o współrzędnej poziomej 1100 pikseli (A + B). A to oznacza, że prawy fragment etykiety o szerokości 100 pikseli nie będzie widoczny! By wyeliminować ten problem, musimy wiedzieć, jaki fragment etykiety wystaje poza prawą krawędź okna przeglądarki, i odpowiednio skorygować jej współrzędną poziomą.

Kod zapisany w wierszu 18. oblicza całkowitą szerokość fragmentu etykiety wystającego poza prawą krawędź okna przeglądarki (oczywiście, o ile w ogóle jest taki fragment). W tym celu wyliczamy współrzędną poziomą prawej krawędzi

etykiety, zakładając, że byłaby ona umieszczona w tym samym miejscu (w poziomie), co element wyzwalający — `triggerPos.left + tipA` (A + B, na rysunku 10.11). Od uzyskanego wyniku odejmujemy następnie szerokość okna przeglądarki (C). Jeśli ostateczny wynik jest wartością dodatnią, jakiś fragment etykiety znajdzie się poza oknem przeglądarki. Jeśli jednak wynik będzie ujemny, będzie to znaczyć, że w oknie przeglądarki jest na tyle dużo miejsca, by etykieta się w nim zmieściła w całości.

14. **Poniżej wiersza dodanego w poprzednim kroku (czyli 18. wiersza) dodaj następujący fragment kodu**:

```
if (overFlowRight > 0) {
    ttLeft = triggerPos.left - overFlowRight - 10;
} else {
    ttLeft = triggerPos.left;
}
```

Najprościej rzecz ujmując, ten fragment kodu sprawia, że jeśli wartość zmiennej `overFlowRight` jest większa od zera (czyli etykieta nie zmieści się w całości w oknie przeglądarki), współrzędna pozioma etykiety zostanie wyliczona jako współrzędna pozioma elementu wyzwalającego pomniejszona o wielkość, o jaką etykieta wystaje poza prawą krawędź okna przeglądarki. Pomniejszenie wyniku o dodatkowe 10 pikseli sprawia, że etykieta nawet nie będzie dotykać krawędzi okna przeglądarki. Jeśli jednak wartość zmiennej `overFlowRight` jest mniejsza od zera, współrzędna pozioma etykiety może być taka sama jak współrzędna pozioma elementu wyzwalającego — `ttLeft = triggerPos.left;`.

O rany — cała masa arytmetyki! Na szczęście, to już koniec. Teraz, kiedy już wyliczyliśmy współrzędne etykiety, możemy ją wyświetlić. W końcu!

15. **Do funkcji mouseover dodaj wiersze od 24. do 28. poniższego fragmentu kodu:**

```
1   $('.trigger').mouseover(function() {
2       var ttLeft,
3           ttTop,
4           $this=$(this),
5           $tip = $($this.attr('data-tooltip')),
6           triggerPos = $this.offset(),
7           triggerH = $this.outerHeight(),
8           triggerW = $this.outerWidth(),
9           tipW = $tip.outerWidth(),
10          tipH = $tip.outerHeight(),
11          screenW = $(window).width(),
12          scrollTop = $(document).scrollTop();
13          if (triggerPos.top - tipH - scrollTop > 0 ) {
14              ttTop = triggerPos.top - tipH - 10;
15          } else {
16              ttTop = triggerPos.top + triggerH +10 ;
17          }
18          var overFlowRight = (triggerPos.left + tipW) - screenW;
19          if (overFlowRight > 0) {
20              ttLeft = triggerPos.left - overFlowRight - 10;
21          } else {
22              ttLeft = triggerPos.left;
23          }
24          $tip.css({
25              left : ttLeft ,
26              top : ttTop,
27              position: 'absolute'
28              }).fadeIn(200);
29  }); // koniec funkcji mouseover
```

W końcu nadszedł moment prawdy. Korzystając z techniki tworzenia sekwencji wywołań funkcji jQuery (patrz strona 149), najpierw wywołujemy funkcję `.css()` (patrz strona 155) i określamy w ten sposób współrzędne (właściwości `left` oraz `top`) oraz sposób rozmieszczenia znacznika etykiety (właściwość `position`, której przypisujemy wartość `absolute`, gdyż chcemy go umiejscowić w sposób bezwzględny), a następnie funkcję `fadeIn()` (patrz strona 200), która sprawi, że etykieta stopniowo pojawi się na ekranie. Na szczęście, ukrycie etykiety, kiedy wskaźnik myszy zostanie usunięty z obszary elementu wyzwalającego, jest znacznie łatwiejsze.

16. **Dokończ tworzenie kodu, dodając wiersze od 30. do 32.; poniżej przedstawiona została pełna, końcowa wersja kodu.**

```
1   $('.trigger').mouseover(function() {
2       var ttLeft,
3           ttTop,
4           $this=$(this),
5           $tip = $($this.attr('data-tooltip')),
6           triggerPos = $this.offset(),
7           triggerH = $this.outerHeight(),
8           triggerW = $this.outerWidth(),
9           tipW = $tip.outerWidth(),
10          tipH = $tip.outerHeight(),
11          screenW = $(window).width(),
12          scrollTop = $(document).scrollTop();
13          if (triggerPos.top - tipH - scrollTop > 0 ) {
14              ttTop = triggerPos.top - tipH - 10;
15          } else {
16              ttTop = triggerPos.top + triggerH +10 ;
17          }
18          var overFlowRight = (triggerPos.left + tipW) - screenW;
19          if (overFlowRight > 0) {
20              ttLeft = triggerPos.left - overFlowRight - 10;
21          } else {
22              ttLeft = triggerPos.left;
23          }
24          $tip.css({
25              left : ttLeft ,
26              top : ttTop,
27              position: 'absolute'
28              }).fadeIn(200);
29  }); // koniec funkcji mouseover
30  $('.trigger').mouseout(function () {
31      $('.tooltip').fadeOut(200);
32  }); // koniec funkcji mouseout
```

Procedura obsługi zdarzeń `mouseover` jest bardzo prosta: w odpowiedzi na usunięcie wskaźnika myszy z obszaru elementu wyzwalającego wystarczy zaciemnić wszystkie etykiety. I to wszystko. Teraz zapisz plik i wyświetl go w przeglądarce. Pełną wersję kodu przykładu można znaleźć w pliku *complete_tooltip.html* umieszczonym w katalogu *R10*.

ALARM! WTYCZKA!

Etykiety ekranowe w nieco łatwiejszy sposób

Próba utworzenia własnego narzędzia do obsługi etykiet jest doskonałym sposobem opanowania funkcji jQuery służących do określania wymiarów i położenia elementów. Jeśli jednak poszukujesz dodatkowych możliwości, takich jak pięknie wyglądające etykiety, komiksowe dymki, pobieranie zawartości dymków przy użyciu AJAX-a bądź precyzyjne umieszczanie etykiet w wybranych miejscach strony WWW, musisz wiedzieć, że istnieje wiele wtyczek jQuery udostępniających znacznie więcej możliwości niż prosty skrypt utworzony w tym rozdziale.

- **qTip2** (*http://craigsworks.com/projects/qtip2/*) jest bardzo rozbudowaną wtyczką obsługującą etykiety. Pozwala nie tylko na tworzenie prostych etykiet, takich jak przedstawione w naszym przykładzie, lecz także etykiet przypominających dymki z komiksowymi rozmowami; pozwala także na śledzenie ruchu wskaźnika myszy przesuwanego po ekranie, na pobieranie zawartości etykiet z serwera oraz udostępnia wiele innych możliwości, między innymi tworzenie okien dialogowych oraz rozwijalnych menu. Jak widać, wtyczka ta to prawdziwy, wielofunkcyjny zestaw narzędziowy.
- **jQuery Tools Tooltip** (*http://jquerytools.org/demos/tooltip/index.html*) to kolejna doskonała wtyczka do tworzenia etykiet. Generowane przez nią etykiety są bardzo atrakcyjne i mają ogromne możliwości dostosowywania. Skoro już mowa o tej wtyczce, warto zajrzeć także na stronę kolekcji narzędzi jQuery Tools (*http://jquerytools.org/tools/*). Jest ona reklamowana jako „zaginiona biblioteka interfejsu użytkownika dla stron WWW" i mimo że slogan ten jest nieco napuszony, to jednak skrypt ten zaspokaja wiele potrzeb twórców stron WWW. Udostępnia między innymi narzędzia do tworzenia kart, nakładek, formularzy oraz suwaków (podobnie jak wtyczka AnythingSlider opisana na stronie 325).
- **Wtyczka jQuery UI Tooltip** (*http://wiki.jqueryui.com/w/page/12138112/Tooltip*). Dowiedziałeś się już o bibliotece jQuery UI w ramce zamieszczonej na stronie 325. Zawiera ona sporo komponentów interfejsu użytkownika oraz innych narzędzi przeznaczonych dla projektantów i twórców stron WWW. Choć w czasie pisania tej książki twórcy tego projektu jeszcze nie udostępnili oficjalnie wtyczki do tworzenia etykiet, jednak prace nad nią są już całkiem zaawansowane, a według planu ma ona zostać udostępniona w wersji 1.9 biblioteki jQuery UI. Wszystkie wtyczki wchodzące w skład biblioteki jQuery UI są doskonałe.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

ż

JavaScript i jQuery

nieoficjalny podręcznik

JavaScript ma za sobą długą historię, w której bywały okresy lepsze i gorsze. Czasem język ten był wręcz masowo blokowany w przeglądarkach. Jednak te czasy minęły. W tej chwili nie obejdzie się bez niego żadna poważna aplikacja internetowa lub choć trochę bardziej zaawansowana strona WWW. Użytkownicy serwisów internetowych wymuszają na projektantach coraz nowsze i lepsze rozwiązania. Dlatego na rynku wciąż pojawiają się dodatkowe narzędzia dla języka JavaScript, które ułatwiają wykorzystanie jego potencjału.

Najpopularniejszym dodatkiem tego typu jest biblioteka jQuery. Genialna w swojej prostocie, z ogromnymi możliwościami, zdobyła uznanie wszystkich programistów JavaScriptu. Nie potrafią sobie oni wyobrazić programowania bez jej wykorzystania. W tej książce znajdziesz najlepsze techniki, jakie oferuje JavaScript. Nauczysz się nawigować po drzewie DOM, modyfikować zachowanie elementów oraz obsługiwać zdarzenia. Poznasz również narzędzia, które ułatwią Ci pracę oraz debugowanie kodu. Jest to obowiązkowa pozycja dla każdego projektanta stron internetowych. Musisz ją mieć!

Dzięki tej książce:

- **poznasz podstawy języka JavaScript**
- **zobaczysz, jak jQuery potrafi ułatwić pracę z JavaScriptem**
- **opanujesz mechanizm zdarzeń w jQuery**
- **zbudujesz lepszą i ciekawszą witrynę!**

Twórz atrakcyjne strony WWW.
Wzbogać kod HTML o animacje, interaktywność i dynamiczne efekty wizualne!

helion.pl
księgarnia internetowa

Nr katalogowy: 8754

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

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

ISBN 978-83-246-4381-3
9 788324 643813
Cena 79,00 zł

Informatyka w najlepszym wydaniu