/Jacek Ross

PHP i HTML

Rozpocznij naukę tworzenia serwisów WWW już dziś
Poznaj najpopularniejsze rozwiązania stosowane w sieci
Dołącz do grupy osób najczęściej poszukiwanych na rynku pracy

Helion

PHP i HTML. Tworzenie dynamicznych stron WWW

Autor: Jacek Ross
ISBN: 978-83-246-2597-0
Format: 158×235, stron: 208

- Rozpocznij naukę tworzenia serwisów WWW już dziś
- Poznaj najpopularniejsze rozwiązania stosowane w sieci
- Dołącz do grupy osób najczęściej poszukiwanych na rynku pracy

W prasie codziennej i internecie pełno jest ogłoszeń z ofertami pracy dla programistów. Duża ich część jest skierowana do osób profesjonalnie zajmujących się tworzeniem i utrzymywaniem serwisów WWW. Nic dziwnego, bowiem technologie internetowe przeżywają obecnie prawdziwy boom, coraz więcej ludzi ma dostęp do sieci i mają oni coraz większe wymagania wobec tego, co w niej znajdują. Najbardziej popularnymi narzędziami używanymi do tworzenia serwisów WWW od dłuższego już czasu są – i pozostaną jeszcze bardzo długo – języki PHP i HTML. Standardem jest też zastosowanie języka JavaScript, kaskadowych arkuszy stylów i technologii AJAX. Od czego jednak należy rozpocząć naukę?

Jeśli chciałbyś zacząć projektować atrakcyjne serwisy WWW i szybko dołączyć do grona najlepszych profesjonalistów w tej dziedzinie, sięgnij po książkę „PHP i HTML. Tworzenie dynamicznych stron WWW". Znajdziesz w niej przegląd najbardziej popularnych technik i nowoczesnych narzędzi, które pozwolą Ci odnaleźć się w skomplikowanym świecie technologii internetowych. Co ważniejsze, uda Ci się to bez konieczności wertowania grubych annałów informatycznych i przekopywania się przez niezrozumiałe dla przeciętnego człowieka specyfikacje techniczne. Książka ta ma szanse zastąpić kilka innych podręczników poświęconych tworzeniu serwisów WWW, a praktyczny sposób prezentacji wiedzy stanowi jeden z jej największych atutów. Lektura nie wymaga ukończenia wyższych studiów informatycznych, ponieważ zawarte w dodatkach podstawy umożliwią rozpoczęcie programowania nawet początkującym twórcom.

- Projektowanie serwisów WWW
- Korzystanie z języków PHP, XML i HTML
- Możliwości języka JavaScript i technologii DHTML
- Podstawy technologii AJAX
- Używanie kaskadowych arkuszy stylów
- Praktyczne zastosowania technologii internetowych

Sięgnij do kompetentnego źródła wiedzy o tworzeniu dynamicznych serwisów WWW!

Spis treści

Rozdział 5.

JavaScript i DHTML

5.1. Skrypty JavaScript, ich ulokowanie w dokumencie

JavaScript to obecnie najpopularniejszy język używany w aplikacjach sieciowych do akcji po stronie klienta. Jest interpretowany przez wszystkie nowoczesne przeglądarki. (Warto przy tym pamiętać, że nie identycznie — niestety pewne techniki dostępne są tylko na niektórych przeglądarkach, a inne interpretowane inaczej; przenośność jest nadal przekleństwem JS). Skrypty JS mogą i w przypadku większych fragmentów powinny być umieszczane w osobnych plikach. W dokumencie HTML zawsze jednak musi się znaleźć odwołanie do skryptów, których chcesz użyć. Podstawowy sposób, w jaki możesz to uczynić, polega na wstawieniu znacznika `SCRIPT` do znacznika `HEAD`.

Plik *Simple.html*:

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript" language="javascript">
<!--
alert('hello');
 //-->
</SCRIPT>
</HEAD>
</HTML>
```

Skrypt wewnątrz znaczników komentarza (`<!--` i `-->`) można dodać w celu zgodności ze starymi przeglądarkami, które nie obsługiwały JavaScript i dzięki temu zabiegowi po prostu zignorują skrypt. Jeśli zakładasz, że Twoja strona musi być uruchamiana na nowszej przeglądarce, to możesz opuścić ten fragment.

Jeśli chcesz wstawić skrypt z innego dokumentu, powinieneś podać atrybut `src` i wskazać nim lokalizację skryptu, a zawartość znacznika `SCRIPT` pozostawić pustą:

```
<script src="../js/admin.produkty.js" language="javascript"></script>
```

Dawniej starsze wersje przeglądarek tolerowały skrypty wyłącznie wewnątrz znacznika HEAD. Ograniczało to możliwości, nie pozwalając na osiągnięcie wielu interesujących efektów. Obecnie można umieszczać je w zasadzie w dowolnym miejscu dokumentu. Należy przy tym pamiętać, że jest on przetwarzany sekwencyjnie, co oznacza, że kod znajdujący się w górnej części dokumentu zostanie zinterpretowany najpierw, a ten znajdujący się pod koniec dopiero w drugiej kolejności. W szczególnym przypadku, jeśli strona nie załaduje się do końca (z powodu błędu lub anulowania ładowania przez użytkownika), część skryptów może nie zostać wykonana.

Skrypty JS mogą być umieszczone także w atrybutach określających zdarzenia związane ze znacznikami. W dalszej części rozdziału zostanie to opisane dokładniej. Tutaj zaprezentuję, jak wygląda taki skrypt na przykładzie użytecznego zdarzenia onLoad znacznika BODY. Kod JS umieszczony w obsłudze tego zdarzenia zostanie wykonany, gdy cała strona będzie finalnie załadowana.

```
<BODY onLoad="javascript: location.href='inna_strona.php">
```

Powyższy przykład pokazuje, jak za pomocą JavaScript wykonać przekierowanie na inną stronę. Po załadowaniu strony do końca wykonany zostanie kod location.href=
↪'inna_strona.php, który spowoduje, że przeglądarka rozpocznie ładowanie adresu *inna_strona.php*. (Wadą takiego przekierowania jest to, że użytkownik przez chwilę będzie widział zawartość strony, więc jeśli chce się tego uniknąć, to przekierowanie trzeba zrobić inaczej).

Jak już napisano — różne przeglądarki różnie interpretują kod JavaScript, mają także inny zestaw klas opisujący dokument. Pisząc zaawansowane skrypty w tym języku, prędzej czy później na pewno zostaniesz zmuszony do wykrywania rodzaju przeglądarki i pisania dwóch wersji niektórych fragmentów kodu. Główna linia podziału biegnie między Internet Explorerem a Firefoksem. Część rzadszych przeglądarek może czasem wymagać własnych dostosowań, najczęściej jednak albo ich udział w rynku jest tak znikomy, że można je pominąć, albo zachowują się identycznie jak jedna z dwóch wymienionych. (Przeglądarki oparte na jądrze Gecko będą zachowywać się jak Firefox, z kolei wiele programów ma wbudowane własne wewnętrzne przeglądarki, które poprzez technologię COM oparte są na zainstalowanym w systemie IE. Istnieją także modyfikacje tej ostatniej, które pod względem JS również zachowują z nią zgodność). Firefox odziedziczył wiele cech po przeglądarce Netscape Navigator, może się jednak zdarzyć, że klient korzysta jeszcze z jakiejś wyjątkowo starej wersji tej przeglądarki (lub bardzo starej wersji IE), powinieneś zatem co najmniej sprawić, żeby Twoja strona wyświetliła jakiś komunikat błędu lub chociaż nie zrobiła nic. Jak więc wykryć, z którą przeglądarką ma się do czynienia? Nie ma sposobów idealnych — można po prostu zapytać przeglądarkę, jak się nazywa, ale ze względu na mnogość wersji, odmian i modyfikacji trzeba by użyć dość skomplikowanego zestawu warunków i nie będzie to ani skuteczne (a jeśli nie przewidzimy jakiejś przeglądarki lub powstaną nowe?), ani eleganckie. Lepiej jest sprawdzić, jakie technologie udostępnia bieżąca przeglądarka. W końcu jeśli potrafi zwrócić takie obiekty, jakie zwraca IE6, nie musi Cię obchodzić, co to za program. Ważne, że działa jak IE6 itd. Zwykle podczas takiej weryfikacji sprawdza się, jak w danej przeglądarce można odwołać się do obiektu reprezentującego dokument HTML albo zestaw jego znaczników. Oto przykładowy prosty kod.

Plik *browser1.html*:

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript" language="javascript">
var ns6 = document.getElementById && !document.all;
var ie4 = document.all;
if(ie4)
alert ('Internet Explorer 4 lub nowszy');
else if(ns6)
   alert('NN6 lub FireFox');
else
   alert('Nieznana przeglądarka lub na tyle stara, że nie obsłuży poprawnie naszego
   ↪skryptu');
</SCRIPT>
</HEAD>
</HTML>
```

Powyższy kod opiera się na tym, że IE4 i nowsze udostępniają zawartość dokumentu poprzez *document.all*, a Firefox i pochodne nie udostępniają takiej właściwości i należy w nich skorzystać z innych metod obiektu `dokument`. Toteż wszędzie, gdzie wystąpi potrzeba odwołania się do jednej z tych konstrukcji, trzeba będzie niestety wstawić instrukcję warunkową.

Jako ciekawostkę podaję jeszcze inny, bardziej rozbudowany przykład rozpoznawania przeglądarek, który dokonuje dodatkowo rozróżnienia między NN4 a NN6 i nowszymi:

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript" language="javascript">
var nav4 = document.layers ? true : false;
var iex = document.all ? true : false;
var nav6 = document.getElementById && !document.all ? true : false;
var old = ( !( nav4 || iex || nav6 ));
</SCRIPT>
</HEAD>
</HTML>
```

5.2. Modyfikowanie dokumentu HTML przez JavaScript

JavaScript umożliwia bardzo daleko idącą kontrolę nad dokumentem HTML. W zasadzie można dokonać w nim niemal nieograniczonych zmian, całkowicie modyfikując jego zawartość. Można podmieniać atrybuty, znaczniki czy fragmenty HTML-a. Oczywiście fakt, że techniki takie są dozwolone, nie oznacza, że można z tym przesadzać. Nadmiar ingerencji w treść dokumentu może spowodować, że całość stanie się kompletnie nieczytelna, a znalezienie w niej błędu niemal niemożliwe. Każdej techniki należy więc używać z umiarem i wtedy, kiedy naprawdę jest potrzebna. Przyjrzyjmy im się.

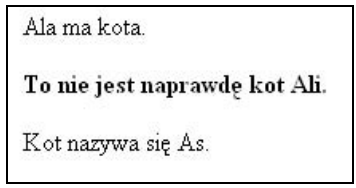
Najprostszą metodą na zmodyfikowanie dokumentu HTML jest wygenerowanie jego fragmentu w miejscu działania skryptu. Oto przykład.

Plik *ala1.html*:

```
<HTML><BODY>
<P>Ala ma kota.</P>
<SCRIPT type="text/javascript">
   document.writeln('<B>To nie jest naprawdę kot Ali.</B>');
</SCRIPT>
<P>Kot nazywa się As.</P>
</BODY></HTML>
```

Powyższy kod spowoduje wyświetlenie się w przeglądarce przykładowego tekstu (rysunek 5.1).

Rysunek 5.1. *Zrzut ekranu: wykonanie przykładu ala1.html*



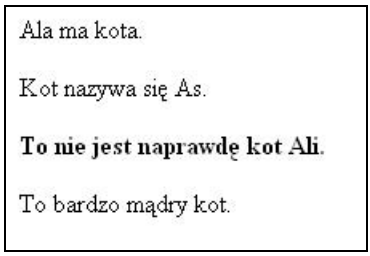
Jak widać, kod JavaScript wstawia fragment HTML-a. Działanie przeglądarki polega na tym, że interpretuje on dokument normalnie do napotkania kodu JS, następnie wykonuje go, po czym interpretuje dalej, ale dokument zawiera już w tym miejscu dodatkowe znaczniki, które przeglądarka traktuje tak, jakby znajdowały się w dokumencie od samego początku.

Aby lepiej zrozumieć, jak przeglądarka przetwarza kod skryptów, przeanalizuj poniższy, nieco bardziej złożony przykład.

Plik *ala2.html*:

```
<HTML><BODY>
<P>Ala ma kota.</P>
<SCRIPT type="text/javascript">
function NapiszPrawdę()
{
   document.writeln('<B>To nie jest naprawdę kot Ali.</B>');
}
</SCRIPT>
<P>Kot nazywa się As.</P>
<SCRIPT>
   NapiszPrawdę();
</SCRIPT>
<P>To bardzo mądry kot.</P>
</BODY></HTML>
```

W przeglądarce zostanie wyświetlony następujący ciąg tekstów:



Rysunek 5.2. *Zrzut ekranu: wykonanie przykładu ala2.html*

Nie jest bowiem ważne miejsce ulokowania kodu, ale miejsce jego faktycznego wykonania. Kod funkcji zostaje wykonany w momencie, gdy przeglądarka interpretuje fragment dokumentu pomiędzy tekstem „Kot nazywa się As" a tekstem „To bardzo mądry kot", i właśnie w to miejsce wstawiany jest nowy fragment tekstu.

Korzystając z obiektu `document`, masz dostęp do całego dokumentu HTML, dzięki czemu możesz dowolnie manipulować znacznikami. Należy pamiętać, że obiekt ten ma nieco inną postać i trochę inne metody oraz właściwości w różnych przeglądarkach, dlatego często trzeba będzie pisać kod warunkowy, co zostało omówione w podrozdziale 5.1. Do poszczególnych znaczników najłatwiej będzie dostać się poprzez identyfikator. Aby nadać go znacznikowi, należy dodać mu atrybut `id` i przypisać do niego unikatową wartość. Wówczas bardzo łatwo można podmienić wnętrze takiego znacznika.

Plik *ala3.html*:

```
<HTML><BODY>
<P>Ala ma kota.</P>
<P id="imię_kota">Kot nazywa się As.</P>
<SCRIPT type="text/javascript">
   var ns6=document.getElementById&&!document.all;
   var ie4=document.all;
   var imię = '<B>Kot nazywa się Mruczek.</B>';
   if(ie4)   document.all('imię_kota').innerHTML = imię;
   else      document.getElementById('imię_kota').innerHTML = imię;
</SCRIPT>
</BODY></HTML>
```

Tutaj istotne jest, że znacznik z `id='imię_kota'` znajduje się wcześniej niż skrypt. Obiekt `document` jest bowiem budowany w trakcie interpretacji dokumentu HTML i jeśli zamieni się te dwa fragmenty, to nie zostanie podmieniony. Przy tym fragmenty wstawione już przez sam kod JS również są dostępne, można to zaobserwować, modyfikując powyższy przykład.

Plik *ala4.html*:

```
<HTML><BODY>
<P>Ala ma kota.</P>
<P id="imię_kota">Kot nazywa się As.</P>
<SCRIPT type="text/javascript">
   document.write('<p id="po_raz_drugi">ddd</p>');
   var ns6=document.getElementById&&!document.all;
   var ie4=document.all;
   var imię = '<B>Kot nazywa się Mruczek.</B>';
   if(ie4)   document.all('imię_kota').innerHTML = imię;
```

```
   else      document.getElementById('imię_kota').innerHTML = imię;
   if(ie4)   document.all('po_raz_drugi').innerHTML = imię;
   else      document.getElementById('po_raz_drugi').innerHTML = imię;
</SCRIPT>
</BODY></HTML>
```

5.3. Obsługa zdarzeń w JavaScript

Podczas przeglądania dokumentu HTML użytkownik może zainicjować wiele czynności, takich jak kliknięcie we fragment dokumentu, przesunięcie wskaźnika myszy itp. Istnieje możliwość implementacji funkcji w języku JavaScript, która zostanie wykonana, gdy takie zdarzenie wystąpi. Poza inicjowanymi przez użytkownika istnieją zdarzenia, które pojawiają się samoczynnie, są związane z wewnętrzną pracą przeglądarki lub zostały zaprogramowane.

Przykładem zdarzenia jest przedstawiony poniżej kod.

Plik *events1.html*:

```
<HTML><HEAD>
<SCRIPT type="text/javascript">
function Funkcja1()
{
   alert('witam');
}
</SCRIPT>
</HEAD>
<BODY onLoad="javascript:Funkcja1()">
</BODY></HTML>
```

Zdarzenie `onLoad` znacznika `BODY` jest wywoływane, kiedy cały dokument się załaduje (włącznie z ewentualnym ładowaniem plików multimedialnych). Oznacza to, że w powyższym przypadku funkcja `Funkcja1` zostanie wywołana po tym, jak załaduje się cały dokument. Zamiast wywoływać funkcje, można wprowadzić bezpośrednio poniższy kod.

Plik *events2.html*:

```
<HTML><HEAD>
</HEAD>
<BODY onLoad="javascript:alert('witam')">
</BODY></HTML>
```

Oto lista interesujących zdarzeń, które można obsłużyć:

- `onMouseDown`, `onMouseUp`, `onMouseOver`, `onMouseOut` — zdarzenia związane z myszką, odpowiednio: wciśnięcie klawisza myszy, puszczenie go, przesunięcie wskaźnika ponad elementem, przesunięcie wskaźnika poza obszar elementu.
- `onKeyDown`, `onKeyPress`, `onKeyUp` — zdarzenia związane z klawiaturą, odpowiednio: naciśnięcie i przytrzymanie klawisza, naciśnięcie krótkotrwałe i puszczenie klawisza po wcześniejszym naciśnięciu.

- `onResize` — zmiana rozmiaru okna.
- `onMove` — poruszenie oknem.
- `onSelect` — wybranie elementu (np. przycisku na formularzu).
- `onSubmit` — zdarzenie wywoływane po zatwierdzeniu (wysłaniu formularza).
- `onReset` — zdarzenie wywoływane po wyczyszczeniu formularza (przyciskiem reset).
- `onClick` — kliknięcie w element (dowolny znacznik).
- `onDblClick` — podwójne kliknięcie w element (dowolny znacznik).
- `onChange` — reakcja na zmianę wartości elementu formularza takiego jak pole tekstowe, pole wyboru itp.
- `onFocus` — reakcja na uaktywnienie elementu, np. przycisku.
- `onError` — zdarzenie wywoływane, gdy ładowanie kończy się błędem.
- `onUnload` — zdarzenie wywoływane przy opuszczaniu strony, np. po zamknięciu okna przeglądarki.

Funkcje obsługujące zdarzenia można również podpinać dynamicznie za pomocą kodu JS. Oto przykład.

Plik *events3.html*:

```
<HTML><HEAD>
<SCRIPT type="text/javascript" language="javascript">
var ns6 = document.getElementById && !document.all;
var ie4 = document.all;
function PowiedzWitam()
{
   alert('witam');
}
</SCRIPT>
</HEAD><BODY>

<A href="http://helion.pl" id="linkHelion">Helion</A>

<SCRIPT type="text/javascript">
   var link;
   if(ie4)
      link = document.all['linkHelion'];
   else
      link = document.getElementById('linkHelion');
   if(ns6)
      link.addEventListener('mouseover', function() {PowiedzWitam()}, false);
   else
      link.attachEvent('onmouseout',  function() {PowiedzWitam()});
</SCRIPT>
</BODY></HTML>
```

Program przykładowy podpina zdarzenie `onmouseout` do odnośnika — po najechaniu na niego wskaźnikiem myszki powinien pojawić się komunikat. Funkcje podpina się

inaczej, w zależności od tego, jaką przeglądarką dysponuje użytkownik. W obydwu przypadkach najpierw szuka się elementu w drzewie obiektów JavaScript (`dokument.all` lub `dokument.getElementById`), a następnie podpina się do niego anonimową funkcję, która wywołuje właściwą funkcję obsługującą zdarzenie (`addEventListener` lub `attach` ➥`Event`). Anonimowa funkcja to po prostu fragment kodu, którego zadaniem jest wywołanie funkcji — `PowiedzWitam`.

5.4. Podmiana fragmentów dokumentu

JavaScript umożliwia szeroką manipulację zawartością dokumentu HTML. Można nieomal dowolnie zmieniać jego treść. Aby tego dokonać, należy uzyskać dostęp do odpowiedniego obiektu w drzewie obiektów dokumentu. Istnieje taki dla każdego znacznika HTML w dokumencie. Jednym ze sposobów dotarcia do niego jest użycie funkcji `docu` ➥`ment.getElementById`, co zostało pokazane w przykładach zamieszczonych w poprzednich podrozdziałach. Jako parametru wejściowego używa się atrybutu `id` znacznika. Gdy już ma się odpowiedni obiekt, można wykorzystać jedną z właściwości, która jest dostępna dla każdego obiektu będącego odpowiednikiem znacznika HTML:

- `innerHTML` — umożliwia odczyt lub zmianę wnętrza znacznika w postaci HTML.
- Tylko dla Internet Explorer: `outerHTML` — umożliwia odczyt lub zmianę fragmentu HTML stanowiącego zarówno wnętrze znacznika, jak i jego samego.
- Tylko dla Internet Explorer: `innerText` — umożliwia odczyt lub zmianę fragmentu dokumentu HTML stanowiącego wnętrze znacznika, ale w postaci prostego tekstu, tj. niezawierającego znaczników. Jeśli przypisać do tego pola wartość np. `'<DIV>tekst</DIV>'`, to znaki `'<'` oraz `'>'` zostaną zastąpione znakami specjalnymi i przeglądarka nie wyświetli znacznika `<DIV>`, a pokaże po prostu taki tekst.
- Tylko dla Internet Explorer: `outerText` — podobnie jak w przypadku `innerText`, lecz obejmuje także tekst stanowiący sam znacznik.

Oto przykład. Należy pamiętać, że dla przeglądarki Firefox zadziała wyłącznie innerHTML, pozostałe trzy funkcje powinny być zastąpione przez manipulacje na obiektach (usunięcie jednego obiektu i wstawienie drugiego, podobnie jak w programie przykładowym pokazanym w rozdziale 2.).

Plik *test.html*:

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function Test()
{
   var element = document.getElementById("jakis_div");
   alert('innerHTML: ' + element.innerHTML);
   alert('innerText: ' + element.innerText);
   alert('outerText: ' + element.outerText);
   alert('outerHTML: ' + element.outerHTML);
```

```
}
</SCRIPT>
</HEAD>
<BODY>
<DIV id="jakis_div">
   To jest DIV
   <HR>
   <P>Ala ma kota</P>
</DIV>
<INPUT type="BUTTON" onClick="Test()" value="Test" />
</BODY>
</HTML>
```

Jak już wspomniałem, podmianę fragmentów dokumentu HTML można również wykonać poprzez bezpośrednie manipulacje na drzewie obiektów, tzn. poprzez usuwanie pewnych elementów i dodawanie innych. Zostało to intensywnie wykorzystane w programie przykładowym do rozdziału 2.

5.5. Podmiana stylów CSS

Za pomocą kodu JavaScript można również manipulować stylami CSS. Dzięki temu uzyskuje się w łatwy sposób wiele ciekawych efektów, w tym zarówno dynamiczną szatę graficzną serwisu (zmieniającą się pod wpływem interakcji użytkownika), jak i możliwość manipulacji np. pojawianiem się pewnych elementów czy ich lokalizacją.

Jednym z prostszych sposobów manipulowania stylami jest przygotowanie w arkuszu stylów kilku zestawów wyglądu pewnego elementu i następnie dynamiczne przełączanie między nimi poprzez zmianę atrybutu `class` elementu. Aby tego dokonać, musisz znaleźć obiekt odpowiadający znacznikowi i zmienić wartość pola `className`. Oto przykład.

Plik *KlasyCSS.html*:

```
<HTML>
<STYLE>
.bialy { color: white;}
.czerwony { color: red;}
.zielony { color: green;}
</STYLE>
<BODY>
<P id="zmiana">Zmień mój kolor</P>

<INPUT TYPE="BUTTON" value="Biały" onClick="javascript:
➥document.getElementById('zmiana').className='bialy'" />
<INPUT TYPE="BUTTON" value="Czerwony" onClick="javascript:
➥document.getElementById('zmiana').className='czerwony'" />
<INPUT TYPE="BUTTON" value="Zielony" onClick="javascript:
➥document.getElementById('zmiana').className='zielony'" />
</BODY>
</HTML>
```

Zastosowanie powyższej techniki pozwala uzyskać wiele interesujących efektów. Nie trzeba jednak koniecznie uciekać się do przygotowywania gotowych zestawów klas i podmieniania ich. Można manipulować bezpośrednio zawartością stylu. Każdy obiekt będący odpowiednikiem znacznika HTML posiada pole *style*, które jest obiektem odpowiadającym stylowi tego elementu zdefiniowanemu w arkuszu CSS lub bezpośrednio w dokumencie. Element stylu posiada z kolei pola odpowiadające różnym właściwościom CSS, które można modyfikować. W efekcie modyfikacji powyższego przykładu, tak aby wykorzystywał opisaną technikę, uzyskasz:

```
<HTML>
<BODY>
<P id="zmiana" style="color: yellow">Zmień mój kolor</P>

<INPUT TYPE="BUTTON" value="Biały" onClick="javascript:
➥document.getElementById('zmiana').style.color='white'" />
<INPUT TYPE="BUTTON" value="Czerwony" onClick="javascript:
➥document.getElementById('zmiana').style.color='red'" />
<INPUT TYPE="BUTTON" value="Zielony" onClick="javascript:
➥document.getElementById('zmiana').style.color='green'" />
</BODY>
</HTML>
```

5.6. Optymalizacja działania skryptów JavaScript

W rozdziale 4. postulowałem, aby nie oszczędzać rozmiarów skryptów PHP przez skracanie nazw zmiennych bądź funkcji czy też np. przez pomijanie komentarzy. Zaciemniało to kod i nie dawało praktycznie żadnych zysków. Niestety nie jest to aż takie oczywiste w przypadku JavaScript, ponieważ im krótszy będzie dokument wysłany do przeglądarki, tym szybciej załaduje ona stronę (czas przesyłu danych często stanowi wąskie gardło). Interes użytkownika i programisty okazuje się więc niestety sprzeczny. Niektórzy programiści z tego powodu rezygnują z pisania komentarzy i zaczynają korzystać z jednoliterowych nazw zmiennych. Nie jest to jednak dobre rozwiązanie. Co więc można zrobić?

Jednym z rozwiązań jest praca na szerszej wersji kodu, tj. zawierającej komentarze i długie nazwy identyfikatorów zmiennych. Tuż przed wysłaniem na serwer wystarczy tę wersję poddać działaniu specjalnego programu, który nie tylko skraca identyfikatory i usuwa nieproduktywne fragmenty kodu (np. komentarze), ale także potrafi dokonać ograniczonej kompresji kodu. Jest to dobre rozwiązanie, niestety ma jedną wadę: jeśli już po wysłaniu na serwer w skrypcie objawi się błąd, trudno go będzie naprawić, dlatego że podgląd źródeł skryptu pokaże zmodyfikowaną wersję kodu, a nie oryginalną. Dlatego lepiej jest do momentu zakończenia prac i testowania serwisu pozostawić oryginalną wersję, a skompresowaną wysłać dopiero na serwer produkcyjny dostępny dla użytkowników. Niezwykle ważne jest, aby zawsze zachowywać oryginalną wersję kodu, by móc w każdej chwili dokonać analizy.

Zadania do samodzielnego wykonania

1. Napisz samodzielnie jakiś większy program w JavaScript. Może to np. być gra w kółko i krzyżyk.
2. Wykorzystując informacje z rozdziału 2. oraz 5., spróbuj napisać funkcje będące odpowiednikami `outerHTML` i `innerText` dla przeglądarki Firefox.

Pytania kontrolne

P5.1. Czy podmieniona za pomocą kodu JavaScript treść HTML jest widoczna w źródle dokumentu?

a) Tak.

b) Nie.

c) Tak, ale tylko w programie Internet Explorer.

d) Tak, ale tylko w programie Firefox.

P5.2. Czy można rozpoznać, na jakiej przeglądarce uruchamiany jest kod JavaScript?

a) Tak.

b) Nie.

P5.3. Jakie zdarzenie wywoływane jest po zatwierdzeniu (wysłaniu) formularza?

a) `onSelect`.

b) `onSend`.

c) `onSubmit`.

d) `onClick`.

PHP i HTML

Tworzenie **dynamicznych** stron WWW

W prasie codziennej i internecie pełno jest ogłoszeń z ofertami pracy dla programistów. Duża ich część jest skierowana do osób profesjonalnie zajmujących się tworzeniem i utrzymywaniem serwisów WWW. Nic dziwnego, technologie internetowe przeżywają bowiem obecnie prawdziwy boom, coraz więcej ludzi ma dostęp do sieci i mają oni coraz większe wymagania wobec tego, co w niej znajdują. Najbardziej popularnymi narzędziami używanymi do tworzenia serwisów WWW od dłuższego już czasu są — i pozostaną jeszcze bardzo długo — języki PHP i HTML. Standardem jest też zastosowanie języka JavaScript, kaskadowych arkuszy stylów i technologii AJAX. Od czego jednak należy rozpocząć naukę?

Jeśli chciałbyś zacząć projektować atrakcyjne serwisy WWW i szybko dołączyć do grona najlepszych profesjonalistów w tej dziedzinie, sięgnij po książkę **„PHP i HTML. Tworzenie dynamicznych stron WWW"**. Znajdziesz w niej przegląd najbardziej popularnych technik i nowoczesnych narzędzi, które pozwolą Ci odnaleźć się w skomplikowanym świecie technologii internetowych. Co ważniejsze, uda Ci się to bez konieczności wertowania grubych annałów informatycznych i przekopywania się przez niezrozumiałe dla przeciętnego człowieka specyfikacje techniczne. Książka ta ma szanse zastąpić kilka innych podręczników poświęconych tworzeniu serwisów WWW, a praktyczny sposób prezentacji wiedzy stanowi jeden z jej największych atutów. Lektura nie wymaga ukończenia wyższych studiów informatycznych, ponieważ zawarte w dodatkach podstawy umożliwią rozpoczęcie programowania nawet początkującym twórcom.

- Projektowanie serwisów WWW
- Korzystanie z języków PHP, XML i HTML
- Możliwości języka JavaScript i technologii DHTML
- Podstawy technologii AJAX
- Używanie kaskadowych arkuszy stylów
- Praktyczne zastosowania technologii internetowych

Sięgnij do kompetentnego źródła wiedzy o tworzeniu dynamicznych serwisów WWW!

Cena: 39,00 zł

Nr katalogowy: 5413

Księgarnia internetowa: http://helion.pl

Zamówienia telefoniczne: 0 801 339900, 0 601 339900

Wydawnictwo Helion

ul. Kościuszki 1c, 44-100 Gliwice
44-100 Gliwice, skr. poczt. 462
32 230 98 63
http://helion.pl
e-mail: helion@helion.pl

helion.pl księgarnia internetowa

ISBN 978-83-246-2597-0

Informatyka w najlepszym wydaniu