JavaScript. Praktyczny kurs

Autor: Marcin Lis
ISBN: 83-246-2000-1
Format: 158x235, stron: 376

Zostań specjalistą w tworzeniu interaktywnych stron internetowych!

- Jak zapewnić interaktywne zachowanie stron WWW?
- Jak korzystać ze zmiennych, operatorów, instrukcji oraz pętli?
- Jak stworzyć atrakcyjną, bezawaryjną witrynę?

JavaScript od ponad dziesięciu lat jest jednym z podstawowych języków programowania, służących do tworzenia interaktywnych stron WWW. Jego wyjątkowo elastyczna struktura, pozwalająca m.in. na dodawanie animowanych lub dynamicznie rozwijanych elementów do witryn pisanych przy użyciu HTML oraz XHTML, sprawiła, że stał się on wręcz niezbędny przy projektowaniu nowoczesnych stron internetowych. Nie bez znaczenia pozostaje także i to, że JavaScript jest podstawą technologii AJAX, bez której nie sposób obejść się, jeśli strona WWW ma reagować na działania użytkownika tak, jak aplikacja komputerowa.

Książka „JavaScript. Praktyczny kurs" ma za zadanie przedstawić Ci możliwości kryjące się w języku JavaScript i pokazać, jak od razu można wykorzystać je w praktyce. Nie wymaga ona od Ciebie innych umiejętności, poza znajomością podstaw języka (X)HTML – wręcz przeciwnie, to dzięki niej zdobędziesz wiedzę w zakresie projektowania interesujących stron WWW. Dowiesz się, jak umieszczać skrypty w kodzie HTML, poznasz standardy i instrukcje JavaScriptu, a także zasady współpracy z różnymi przeglądarkami. Nauczysz się tworzyć pętle, funkcje, tablice i obiekty, zapewniać obsługę błędów i zdarzeń. Jeśli zależy Ci na szybkiej i efektywnej nauce, to idealna książka dla Ciebie!

- Skrypty w kodzie HTML i XHTML
- Instrukcje, zmienne i typy danych
- Operacje i operatory
- Instrukcje warunkowe
- Pętle
- Funkcje i zasięg zmiennych
- Obiekty i tablice
- Obsługa błędów i wyjątki
- Współpraca z przeglądarkami
- Zdarzenia
- Elementy witryny
- Style CSS
- Operacje na ciągach znaków
- Wprowadzanie danych przez użytkownika
- Wyrażenia regularne
- Cookies
- Obsługa daty i czasu
- Korzystanie z timerów

JavaScript – Twoja droga do projektowania oryginalnych witryn internetowych!

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

Spis treści

Rozdział 5.

Przetwarzanie danych

Lekcja 17. Operacje na ciągach znaków

W lekcji 2., w rozdziale 2. poznaliśmy typ łańcuchowy, czyli typ `string`. Dane tego typu to po prostu ciągi znaków — korzystaliśmy z nich wielokrotnie na łamach książki. W praktyce programistycznej bardzo często spotkamy się jednak z koniecznością najrozmaitszego przetwarzania ciągów znaków. Czasem trzeba będzie sprawdzić, czy w danym ciągu znajduje się jakiś podciąg, czasem wyodrębnić fragment tekstu, niekiedy zbadać, jaka jest pierwsza lub ostatnia litera. Takiej właśnie tematyce poświęcona jest lekcja 17.

Jak sprawdzić długość tekstu?

W lekcji 9., w rozdziale 3. dowiedzieliśmy się, że dane typów prostych (liczbowych, łańcuchowych itd.) w JavaScripcie traktowane są jak obiekty. Dokładnie tak samo jest z ciągami znaków. Jeśli gdzieś w kodzie skryptu wystąpi ciąg znaków, spowoduje to powstanie obiektu typu `string`. Taki obiekt ma własne właściwości i metody, z których możemy korzystać do woli. Właściwością charakterystyczną dla obiektu typu `string` jest `length`[1]. Zawiera ona długość ciągu, czyli liczbę zawartych w nim znaków — niewątpliwie jest więc bardzo użyteczna. Jeśli zatem w zmiennej `str` znajduje się ciąg znaków, jego długość uzyskamy, pisząc:

```
str.length
```

Oto przykład:

```
var str = "To jest przykładowy tekst.";
var długość = str.length;
alert("Długość tekstu to " + długość + " znaków.");
```

[1] Oprócz `length`, obiekt typu `string`, tak jak i każdy inny, zawiera również właściwości `constructor` i `prototype`. Były one omawiane w lekcji 9., w rozdziale 3.

Skoro jednak ciąg znaków w trakcie przetwarzania przez aparat wykonawczy JavaScript jest zamieniany na obiekt, odwołanie do jego właściwości (oraz metod) może być przeprowadzone bezpośrednio. Prawidłowy jest też zapis:

```
"ciąg znaków".length
```

Przykładowo:

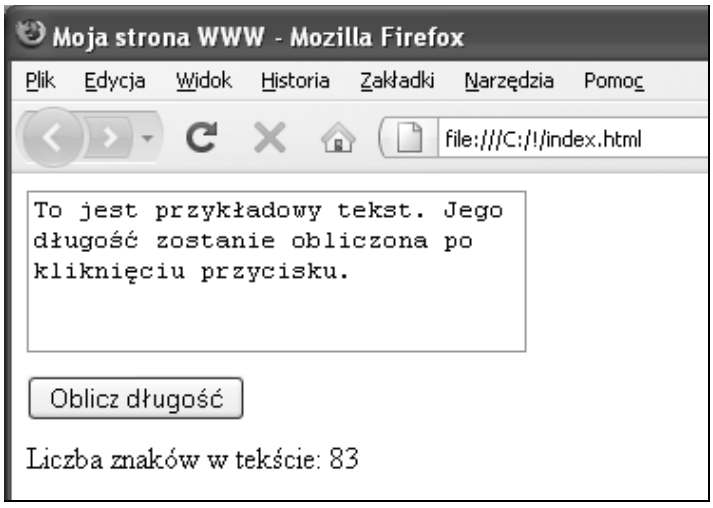
```
var długość = "To jest przykładowy tekst.".length;
alert("Długość tekstu to " + długość + " znaków.");
```

Napiszmy zatem skrypt, który umożliwi użytkownikowi wprowadzenie tekstu i poda jego długość. Kod (X)HTML został zaprezentowany na listingu 5.1.

Listing 5.1. *Strona podająca długość wprowadzonego tekstu*

```
<body>
  <div style="margin-bottom:10px;">
    <textarea id="ta1" cols="30" rows="4"></textarea>
  </div><div style="margin-bottom:10px;">
    <input type="button" value="Oblicz długość"
           id="btn1"
           onclick="obliczDługość();" />
  </div>
  <div id="div3"></div>
</body>
```

Mamy tu trzy warstwy. Pierwsza zawiera rozszerzone pole tekstowe, druga — przycisk, a trzecia (o identyfikatorze `div3`) służy do prezentacji wyników. Pole testowe zostało zdefiniowane za pomocą znacznika `<textarea>` i został mu nadany identyfikator `ta1`. Przyciskowi została przypisana procedura obsługi zdarzenia `onclick` w postaci funkcji `obliczDługość`. A zatem po wprowadzeniu tekstu do pola będzie można kliknąć przycisk *Oblicz długość* i wtedy na ekranie pojawi się informacja o liczbie znaków, co zostało zaprezentowane na rysunku 5.1. Treść funkcji `obliczDługość` znajduje się na listingu 5.2.



Rysunek 5.1. *Po kliknięciu przycisku została wyświetlona długość tekstu*

Listing 5.2. *Funkcja obliczająca długość tekstu*

```
<script type="text/javascript">
  function obliczDługość()
  {
    var ta1 = document.getElementById("ta1");
    var div3 = document.getElementById("div3");
    var długość = ta1.value.length;
    div3.innerHTML = "Liczba znaków w tekście: " + długość;
  }
</script>
```

Funkcja za pomocą metody `getElementById` pobiera odwołania do pola tekstowego oraz warstwy `div3` i zapisuje je w zmiennych `ta1` i `div3`. Następnie odczytuje długość tekstu. Robi to, odwołując się do właściwości `length`. Skoro bowiem ciąg znaków znajdujący się polu `ta1` znajduje się we właściwości `value`, jego długość odczytamy przy użyciu odwołania:

```
ta1.value.length
```

Liczba znaków znajdujących się we wprowadzonym tekście (po odczytaniu) jest wyświetlana na warstwie `div3`:

```
div3.innerHTML = "Liczba znaków w tekście: " + długość;
```

Powyższy przykład można w łatwy sposób zmodyfikować tak, by użytkownik na bieżąco był informowany o tym, ile znaków wprowadził do pola tekstowego. Gdy zajrzymy do tabeli 4.11 z lekcji 14., znajdziemy kilka zdarzeń związanych z obsługą klawiatury. Są to `onkeydown`, `onkeypress` i `onkeyup`. Najodpowiedniejsze do naszego celu wydaje się być `onkeyup`, powstaje bowiem, gdy zostanie puszczony (rzecz jasna po uprzednim wciśnięciu) dowolny klawisz. Wystarczy wtedy sprawdzić liczbę znaków i wyświetlić tę informację na ekranie. Nie musimy przy tym wcale modyfikować skryptu z listingu 5.2. Wystarczą zmiany w kodzie (X)HTML z listingu 5.1. Treść sekcji `body` będzie teraz wyglądała tak, jak na listingu 5.3.

Listing 5.3. *Strona obliczająca liczbę znaków na bieżąco*

```
<body>
  <div style="margin-bottom:10px;">
    <textarea id="ta1" cols="30" rows="4"
              onkeyup="obliczDługość();"
    ></textarea>
  </div>
  <div id="div3"></div>
</body>
```

Z kodu została usunięta warstwa zawierająca przycisk — nie jest w tym przykładzie do niczego potrzebny. Zdarzeniu `onkeyup` pola tekstowego została natomiast przypisana procedura obsługi w postaci funkcji `obliczDługość`. Jest to ta sama funkcja, której treść widnieje na listingu 5.2. Tym razem będzie po prostu wykonywana po każdym puszczeniu klawisza. A więc po każdym takim zdarzeniu na ekranie pojawi się informacja o liczbie znaków w tekście znajdującym się w polu tekstowym. Oznacza to, że wyświetlana informacja będzie uaktualniana na bieżąco w trakcie pisania na klawiaturze.

Metody formatujące ciągi znaków

Metody dostępne w obiektach typu `string` możemy podzielić na takie, które formatują ciągi znaków, oraz takie, które w jakiś sposób ciągi przetwarzają lub udostępniają różne informacje. Zajmiemy się najpierw pierwszą grupą metod. Co prawda, obecnie straciły na znaczeniu (metody te dostępne są od pierwotnych wersji JavaScriptu), ale warto wiedzieć, że istnieją i jak ich używać. Metody formatujące zostały zebrane w tabeli 5.1.

Użyjmy kilku metod przedstawionych w tabeli metod. Napiszemy kod witryny zawierającej cztery pola wyboru typu radio, pole tekstowe oraz przycisk. Za pomocą pól wyboru będzie można wybrać efekt, który zostanie zastosowany w stosunku do tekstu wprowadzonego w polu tekstowym. Witryna będzie więc wyglądała tak, jak na rysunku 5.2. Kod (X)HTML takiej strony został zaprezentowany na listingu 5.4.

Kod witryny składa się z trzech warstw. Pierwsza zawiera cztery pola wyboru zdefiniowane za pomocą znaczników `<input>` z parametrem `type` ustawionym na `radio`. Właściwość `name` wszystkich pól została ustawiona na `grp1`, co sprawiło, że stanowią one jedną grupę i na raz będzie mogło być zaznaczone tylko jedno z nich (są polami wzajemnie się wykluczającymi). Każde pole ma również przypisany identyfikator określający, jaki efekt włącza: `rbBlink` (znacznik `<blink>`, tekst migający, metoda `blink`), `rbBold` (znacznik `<b>`, tekst pogrubiony, metoda `bold`), `rbItalics` (znacznik `<i>`, tekst pochylony, metoda `italics`), `rbStrike` (znacznik `<strike>`, tekst przekreślony, metoda `strike`).

Druga warstwa zawiera pole tekstowe oraz przycisk. Polu został nadany identyfikator `tf1`, za którego pomocą będzie się można odwołać do tego elementu w skrypcie i odczytać wprowadzony tam tekst. Zdarzeniu `onclick` przycisku została natomiast przypisana procedura obsługi w postaci funkcji `przetwórzTekst`. Trzecia warstwa ma identyfikator `div3` i początkowo jest pusta. Będzie się na niej pojawiał tekst wprowadzony do pola tekstowego i przetworzony przez wybraną w polach wyboru metodę.

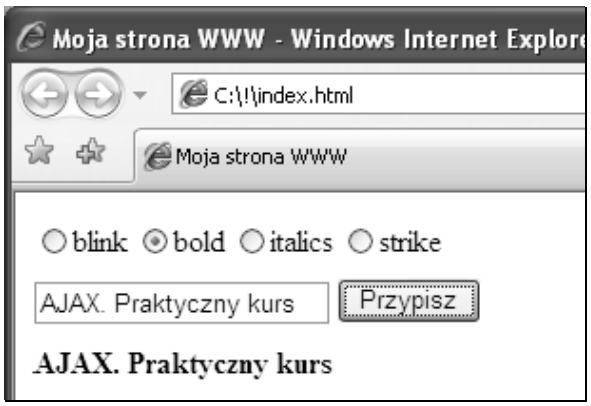
Treść funkcji `przetwórzTekst` została przedstawiona na listingu 5.5.

Najpierw odczytywane są i zapisywane w zmiennych `tf1` i `div3` odwołania pola tekstowego `tf1` i warstwy `div3`, a tekst zapisany w polu `tf1` jest przypisywany zmiennej `tekst`. Następnie w złożonej instrukcji warunkowej `if...else if` badane jest, które z pól wyboru jest zaznaczone, czyli ma właściwość `checked` ustawioną na `true`. W zależności od tego, wywoływana jest jedna z metod obiektu `tekst`, czyli `blink`, `bold`, `italice` lub `strike`, a efekt jej wywołania jest przypisywany z powrotem zmiennej `tekst`. Tym samym, jeżeli jedno z pól wyboru jest zaznaczone, tekst z pola tekstowego zostanie przetworzony przez odpowiadającą polu metodę. Jeżeli zaś żadne z pól nie jest zaznaczone (w praktyce taka sytuacja nie powinna mieć miejsca), tekst pozostanie niezmieniony.

Na końcu kodu wartość zmiennej `tekst`, niezależnie od tego, czy została przetworzona, czy też nie, jest przypisywana właściwości `innerHTML` warstwy `div3`, dzięki czemu zostaje wyświetlona na ekranie.

Tabela 5.1. *Metody formatujące ciągi*

Metoda	Wywołanie	Opis	Przykład wywołania	Efekt wywołania
anchor	*str*.anchor ↪(*param*)	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik <a> z parametrem name równym *param*.	"abc".anchor ↪("def")	abc
big	*str*.big()	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik <big>.	"abc".big ↪("def")	<big>abc</big>
blink	*str*.blink()	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik <blink>.	"abc".blink ↪("def")	<blink>abc ↪</blink>
bold	*str*.bold()	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik .	"abc".bold ↪("def")	abc
fixed	*str*.fixed()	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik <tt>.	"abc".fixed ↪("def")	<tt>abc</tt>
fontcolor	*str*.fontcolor ↪(*kolor*)	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik , z parametrem color równym argumentowi *kolor*.	"abc".fontcolor ↪("red")	abc ↪
fontsize	*str*.fontsize ↪(*rozmiar*)	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik , z parametrem size równym argumentowi *rozmiar*.	"abc". ↪fontsize(7)	abc
italics	*str*. ↪italics()	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik .	"abc".italics()	abc
link	*str*.link ↪(*param*)	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik <a> z parametrem href równym *param*.	"abc".link ↪("http://nazwa. ↪domeny")	abc
small	*str*.small()	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik <small>.	"abc".small()	abc ↪
strike	*str*.strike()	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik <strike>.	"abc".strike()	<strike>abc ↪</strike>
sub	*str*.sub()	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik <sub>.	"abc".sub()	_{abc}
sup	*str*.sup()	Zwraca ciąg znaków, w którym ciąg reprezentowany przez *str* został ujęty w znacznik <sup>.	"abc".sup()	^{abc}



Rysunek 5.2. *Pola wyboru pozwalają na wybór rodzaju tekstu*

Listing 5.4. *Strona zawierająca pola wyboru, pole tekstowe i przycisk*

```
<body>
  <div style="margin-bottom:10px;">
    <input type="radio" id="rbBlink" name="grp1">blink
    <input type="radio" id="rbBold" name="grp1">bold
    <input type="radio" id="rbItalics" name="grp1">italics
    <input type="radio" id="rbStrike" name="grp1">strike
  </div>
  <div style="margin-bottom:10px;">
    <input type="text" id="tf1">
    <input type="button" id="btn1"
           value="Przypisz" onclick="przetwórzTekst();">
  </div>
  <div id="div3"></div>
</body>
```

Listing 5.5. *Funkcja przetwarzająca tekst z pola tekstowego*

```
<script type="text/javascript">
  function przetwórzTekst()
  {
    var tf1 = document.getElementById("tf1");
    var div3 = document.getElementById("div3");
    tekst = tf1.value;
    if(document.getElementById("rbBlink").checked)
      tekst = tekst.blink();
    else if(document.getElementById("rbBold").checked)
      tekst = tekst.bold();
    else if(document.getElementById("rbItalics").checked)
      tekst = tekst.italics();
    else if(document.getElementById("rbStrike").checked)
      tekst = tekst.strike();
    div3.innerHTML = tekst;
  }
</script>
```

Przetwarzanie ciągów

Działanie metod przedstawionych w poprzedniej części lekcji można w bardzo prosty sposób zasymulować ręcznie, wprowadzając instrukcje łączące ciągi. I tak występujące na listingu 5.5 wywołanie:

```
var tekst = tekst.bold();
```

można z powodzeniem zastąpić przez:

```
var tekst = "<b>" + tekst + "</b>";
```

Efekt działania tych konstrukcji będzie identyczny.

Efektów działania omawianych w tej części lekcji metod przetwarzających ciągi nie da się już tak łatwo zasymulować, a wykonują one bardzo wiele zadań wyjątkowo przydatnych przy pracy z ciągami znaków. Metody te zostały zebrane w tabeli 5.2.

Tabela 5.2. *Metody przetwarzające ciągi*

Metoda	Wywołanie	Opis
charAt	*str*.charAt(*indeks*)	Zwraca znak ciągu znajdujący się pod indeksem *indeks*.
charCodeAt	*str*.charCodeAt(*indeks*)	Zwraca kod znaku znajdującego się pod indeksem *indeks*. W wersji JavaScript 1.2 jest to kod ISO-Latin1, a od wersji 1.3 (oraz w JScript) — kod Unicode.
concat	*str*.concat(*str2*, *str3*, ... , *strN*)	Łączy łańcuchy znakowe. Wynikiem działania metody jest ciąg powstały z połączenia łańcucha *str* i wszystkich łańcuchów przekazanych w postaci argumentów.
fromCharCode	String.fromCharCode ↪(*kod1*, *kod2*, ... , ↪*kodN*)	Zwraca ciąg znaków, których kody zostały przekazane w postaci argumentów. Jest to metoda statyczna, do jej wykorzystania nie jest zatem konieczne tworzenie obiektu typu String.
indexOf	*str*.indexOf(*value* ↪[, *index*])	Zwraca *indeks* wystąpienia ciągu *value* w ciągu *str*. Jeżeli podany zostanie opcjonalny drugi argument, przeszukiwanie ciągu *str* rozpocznie się od znaku znajdującego się pod indeksem *index*.
lastIndexOf	*str*.lastIndexOf ↪(*value* [, *index*])	Działa podobnie do metody indexOf, z tą różnicą, że ciąg *str* przeszukiwany jest od końca.
match	*str*.match(*wyrReg*)	Zwraca część ciągu str pasującego do wyrażenia regularnego *wyrReg*. Jeżeli żadna część ciągu nie pasuje do wyrażenia, zwraca wartość null.
replace	*str*.replace ↪(*wyrReg*, *tekst*)	Zwraca ciąg, w którym podciągi opisane przez wyrażenie regularne *wyrReg* zostały zamienione na ciąg znaków reprezentowany przez *tekst*.
search	*str*.search(*wyrReg*)	Sprawdza, czy ciąg opisany przez wyrażenie regularne *wyrReg* występuje w ciągu *str*. Jeśli tak, zwracany jest indeks wystąpienia, jeśli nie, zwracana jest wartość -1.

Tabela 5.2. *Metody przetwarzające ciągi (ciąg dalszy)*

Metoda	Wywołanie	Opis
slice	*str*.slice(*start*[, *end*])	Zwraca podciąg ciągu *str* rozpoczynający się od indeksu wskazywanego przez *start* i kończący się wraz z końcem ciągu *str*, lub też, jeśli został podany parametr *end*, w indeksie wskazywanym przez ten parametr.
split	*str*.split([*separator*[, *limit*]])	Dzieli ciąg znaków *str* na podciągi względem parametru *separator* i zwraca je w postaci tablicy. Parametr *separator* może być ciągiem znaków lub wyrażeniem regularnym, jeżeli nie zostanie podany, zwrócony zostanie pełny ciąg znaków. Parametr *limit* jest wartością całkowitą wskazującą maksymalną liczbę elementów tablicy wynikowej.
substr	*str*.substr(*indeks*[, *ile*])	Zwraca podciąg ciągu *str* rozpoczynający się od znaku o indeksie *indeks*, zawierający liczbę znaków wskazywanych przez parametr *ile*. Jeżeli parametr *ile* nie zostanie podany, zwracane są wszystkie znaki od indeksu *indeks* do końca ciągu *str*.
substring	*str*.substring(*indeks1*, *indeks2*)	Zwraca podciąg ciągu *str* rozpoczynający się od znaku o indeksie *indeks1* i kończący się przed znakiem o indeksie *indeks2*.
toLowerCase	*str*.toLowerCase()	Zwraca ciąg, w którym wszystkie znaki ciągu *str* zostały zamienione na małe litery. Wywołanie metody nie wpływa na zawartość ciągu *str*.
toLowerCase	*str*.toLowerCase()	Zwraca ciąg, w którym wszystkie znaki ciągu *str* zostały zamienione na duże litery. Wywołanie metody nie wpływa na zawartość ciągu *str*.

Spójrzmy na przykłady użycia tych metod.

Działanie metody `charAt` wydaje się oczywiste. Skoro zwraca znak znajdujący się pod wskazanym indeksem, to przykładowe wywołanie:

```
var c = "łąka".charAt(2);
```

spowoduje przypisanie zmiennej `c` znaku `k`. Znak ten znajduje się bowiem pod indeksem 2 (indeksowanie ciągu zaczyna się od 0). Należy jedynie pamiętać, że sekwencje znaków specjalnych tworzą w rzeczywiści jeden znak, czyli w wyniku wykonania instrukcji:

```
var c1 = "\\t".charAt(0);
var c2 = "\\t".charAt(1);
```

otrzymamy przypisanie zmiennej `c1` znaku `\`, a zmiennej `c2` — znaku tabulacji poziomej.

Metoda `charCodeAt` powoduje pobranie kodu znaku znajdującego się pod wskazanym indeksem. Wykonanie przykładowych instrukcji:

```
var c1 = "łąka".charCodeAt(0);
var c2 = "łąka".charCodeAt(2);
```

spowoduje przypisanie zmiennej `c1` wartości `322` (kod znaku *ł* w postaci dziesiętnej), a zmiennej `c2` — wartości `107` (kod znaku *k* w postaci dziesiętnej). Oczywiście, dla sekwencji znaków specjalnych obowiązują te same zasady, co dla metody `charAt`, czyli wywołanie:

```
var c3 = "\\t".charCodeAt(0);
```

spowoduje przypisanie zmiennej `c3` wartości `92` (kod znaku \).

Metoda `concat` zwraca ciąg będący połączeniem ciągu bieżącego i wszystkich ciągów przekazanych w postaci argumentów. A zatem przykładowe wywołanie:

```
var s1 = "abc".concat("123", "def");
```

spowoduje przypisanie zmiennej `s1` ciągu `abc123def`.

Metoda `fromCharCode` jest nieco inna od dotychczas opisanych. Można ją wywołać, nie tworząc żadnego obiektu typu `String` (takie metody nazywa się statycznymi). W wyniku jej działania powstaje ciąg z kodów znaków przekazanych w postaci argumentów. Przykładowe wywołanie:

```
var s1 = String.fromCharCode(322, 261, 107, 97);
```

spowoduje przypisanie zmiennej `s1` ciągu `łąka`.

Metoda `indexOf` pozwala stwierdzić, czy w ciągu istnieje dany podciąg, a zatem sprawdza, czy w ciągu podstawowym istnieje inny ciąg, wskazany za pomocą argumentu. Jeżeli istnieje, zwracany jest indeks wystąpienia, jeśli nie — wartość –1. To oznacza, że instrukcja:

```
var indeks = "piękna łąka".indexOf("na");
```

spowoduje przypisanie zmiennej `indeks` wartości 4 — bowiem ciąg `na` w ciągu `piękna łąka` zaczyna się w pozycji o indeksie 4 (*p* — 0, *i* — 1, *ę* — 2, *k* — 3, *n* — 4). Z kolei instrukcja:

```
var indeks = "piękna łąka".indexOf("one");
```

spowoduje przypisanie zmiennej `indeks` wartości –1, bowiem w ciągu `piękna łąka` nie występuje ciąg `one`.

Omawiana metoda umożliwia użycie drugiego argumentu. Określa on indeks znaku, od którego ma się zacząć przeszukiwanie ciągu podstawowego. Znaczy to, że przykładowe wywołanie:

```
var indeks = "piękna łąka".indexOf("ą", 4);
```

spowoduje przypisanie zmiennej indeks wartości `8` — bowiem ciąg `ą` zaczyna się na 8 pozycji, a przeszukiwanie rozpoczyna od 4. Natomiast wywołanie:

```
var indeks = "piękna łąka".indexOf("ą", 9);
```

spowoduje przypisanie zmiennej `indeks` wartości –1 — bowiem ciąg `ą` zaczyna się na 8 pozycji, a przeszukiwanie zaczyna się od pozycji 9.

Metoda `lastIndexOf` działa na tej samej zasadzie, co `indexOf`, ale przeszukuje ciąg od końca. Jeśli więc wykonamy serię instrukcji:

```
var i1 = "błękitne niebo".lastIndexOf("ne");
var i2 = "błękitne niebo".lastIndexOf("na");
var i3 = "błękitne niebo".lastIndexOf("ki", 6);
var i4 = "błękitne niebo".lastIndexOf("ki", 2);
```

okaże się, że:

- zmienna `i1` zawiera wartość `6`, bowiem ciąg `ne` rozpoczyna się w indeksie 6;
- zmienna `i2` zawiera wartość `-1`, bowiem ciąg `na` nie występuje w ciągu `błękitne niebo`;
- zmienna `i3` zawiera wartość `3`, bowiem przeszukiwanie rozpoczyna się w indeksie 6 (licząc od początku), a ciąg `ki` rozpoczyna się w indeksie 3;
- zmienna `i4` zawiera wartość `-1`, bowiem ciąg `ki` rozpoczyna się w indeksie 3, a przeszukiwanie rozpoczyna się w indeksie 2 (i dąży do indeksu 0);

Metoda `match` pozwala stwierdzić, czy w danym ciągu występuje podciąg odpowiadający wyrażeniu regularnemu przekazanemu w postaci argumentu. Wyrażeniami regularnymi zajmiemy się dopiero w lekcji 19. Funkcja pozwala również na pracę ze zwykłymi ciągami, a zatem możemy stwierdzić, czy w danym ciągu występuje pewien podciąg. Jeśli tak jest, metoda zwróci ten podciąg, jeśli nie — zwróci wartość `null`. Znaczy to, że po wykonaniu instrukcji:

```
var str = "błękitne niebo".match("nie");
```

zmienna `str` będzie zawierała ciąg `nie` (bowiem ciąg `nie` jest podciągiem ciągu `błękitne niebo`), a po wykonaniu instrukcji:

```
var str = "błękitne niebo".match("abc");
```

zmienna `str` będzie zawierała wartość `null` (bowiem ciąg `abc` nie jest podciągiem ciągu `błękitne niebo`).

Metoda `replace` zamienia wszystkie podciągi zgodne z wyrażeniem regularnym (lekcja 19.) podanym jako pierwszy argument na ciąg przekazany jako drugi argument. Jeśli nie chcemy korzystać z zaawansowanych wyrażeń, ale mamy zamiar zamieniać zwykłe ciągi znaków, możemy ciąg poszukiwany umieścić między znakami `/`. Przykładowo po wykonaniu instrukcji:

```
var str = "Cześć %IMIE%. Miło Cię spotkać.".replace(/%IMIE%/, "Adam");
```

zmienna `str` będzie zawierała ciąg znaków:

```
Cześć Adam. Miło Cię spotkać.
```

bowiem ciąg `%IMIE%` zostanie zamieniony na ciąg `Adam`. Uwaga: w ten sposób zostanie zamienione tylko pierwsze wystąpienie szukanego ciągu. Jeśli chcemy zamienić wszystkie wystąpienia, należy skorzystać z opcji `g`, np. po wykonaniu instrukcji:

```
var str1 = "Ala ma kota. Ala ma też psa.".replace(/Ala/, "Ola");
var str2 = "Ala ma kota. Ala ma też psa.".replace(/Ala/g, "Ola");
```

zmienna str1 będzie zawierała ciąg znaków:

```
Ola ma kota. Ala ma też psa.
```

a zmienna str2 ciąg:

```
Ola ma kota. Ola ma też psa.
```

Drugi argument metody może zawierać sekwencje znaków specjalnych (dostępne w JavaScript od wersji 1.2 i JScript od wersji 5.5) przedstawionych w tabeli 5.3.

Tabela 5.3. *Sekwencje znaków specjalnych dla metody replace*

Sekwencja	Znaczenie
$$	znak $
$&	podciąg pasujący do wyrażenia *wyrReg*
$`	podciąg znajdujący się przed ciągiem $&
$'	podciąg znajdujący się za ciągiem $&
$n lub $nn	n-ty podciąg wyrażenia *wyrReg*

Metoda search działa tak samo jak indexOf, z tą różnicą, że argumentem jest wyrażenie regularne. Odszukuje więc pierwszy podciąg pasujący do tego wyrażenia i zwraca indeks jego wystąpienia. Jeżeli w ciągu nie istnieje żaden podciąg pasujący do wyrażenia, zwracana jest wartość –1. Przykładowe wywołania:

```
var i1 = "Ten serwer ma 2GB RAM.".search(/ser[vw]er/);
var i2 = "This server has 2GB RAM.".search(/ser[vw]er/);
```

spowodują przypisanie zmiennej i1 wartości 4, a zmiennej i2 — wartości 5. Wyrażenie /ser[vw]er/ pasuje bowiem zarówno do ciągu serwer (a występuje on w ciągu głównym na 4. pozycji), jak i server (a występuje on w ciągu głównym na 5. pozycji).

Metoda slice pozwala na pobranie fragmentu danego ciągu. Jeżeli zostanie wywołana z jednym argumentem, zwrócony podciąg będzie się rozpoczynał w indeksie wskazanym przez ten argument i kończył w ostatnim znaku ciągu głównego. Jeżeli zostaną podane dwa argumenty, zwrócony zostanie podciąg przez nie wyznaczany. Znaczy to, że wywołanie:

```
var str1 = "świt w świecie świerszcza".slice(7);
```

spowoduje przypisanie zmiennej str1 wartości świecie świerszcza, a wywołanie:

```
var str2 = "świt w świecie świerszcza".slice(8, 14);
```

spowoduje przypisanie zmiennej str2 wartości wiecie.

Możliwe jest również stosowanie argumentów o wartościach ujemnych. Wtedy są one interpretowane jako przesunięcia względem końca ciągu głównego. Przykładowo wywołanie:

```
str = "niebieska waza".slice(5, -5);
```

spowoduje przypisanie zmiennej `str` ciągu `eska`. Jest to ciąg rozpoczynający się w znaku o indeksie 5, licząc od początku, i kończący się w znaku o indeksie 5, licząc od końca. Identyczny efekt da wywołanie:

```
str = "niebieska waza".slice(-9, -5);
```

Będzie to powiem ciąg rozpoczynający się w znaku o indeksie 9, licząc od końca, i kończący się w znaku o indeksie 5, również licząc od końca.

Metoda `split` umożliwia podzielenie ciągu względem znaków separatora przekazanego jako pierwszy argument. Podzielony ciąg jest zwracany w postaci tablicy. Opcjonalny drugi argument pozwala określić maksymalną liczbę ciągów wynikowych (a tym samym, rozmiar tablicy wynikowej). Separator może być określony jako pojedynczy znak, ciąg znaków lub wyrażenie regularne. Wykonanie przykładowych instrukcji:

```
var tab1 = "a b c".split(" ");
var tab2 = "a,b,c".split(",", 2);
var tab3 = "a, b, c".split(", ", 2);
var tab4 = "dwa".split("");
```

spowoduje utworzenie następujących tablic:

- `tab1` — zawierającej trzy komórki z ciągami `a`, `b` i `c` — znakiem separatora jest bowiem spacja, a liczba ciągów wynikowych nie jest ograniczona,
- `tab2` — zawierającej dwie komórki z ciągami `a`, `b` — znakiem separatora jest bowiem przecinek, a liczba ciągów wynikowych jest ograniczona do dwóch,
- `tab3` — zawierającej dwie komórki z ciągami `a` i `b` — separatorem jest bowiem ciąg składający się z przecinka i spacji, a liczba ciągów wynikowych jest ograniczona do dwóch,
- `tab4` — zawierającej trzy komórki z ciągami `d`, `w` i `a` — separatorem jest bowiem pusty ciąg znaków, co powoduje podzielenie ciągu na poszczególne litery (znaki).

Metoda `substr`, podobnie jak `slice` i `substring`, pozwala wyodrębnić fragment ciągu. Pierwszy argument określa indeks początkowy, a drugi — liczbę znaków do pobrania. Drugi argument można pominąć — wtedy pobierany fragment rozpocznie się od indeksu wskazywanego przez pierwszy argument, a skończy w końcu ciągu głównego. Znaczy to, że przykładowe wywołanie:

```
str = "wspaniały świat".substr(2, 4);
```

spowoduje przypisanie zmiennej `str` ciągu `pani` (4 znaki, począwszy od znaku o indeksie 2 w ciągu głównym), a wywołanie:

```
str = "wspaniały świat".substr(9);
```

spowoduje przypisanie zmiennej `str` ciągu `świat` (wszystkie znaki, począwszy od tego o indeksie 9, aż do końca ciągu głównego).

Metoda `substring` pobiera wybrany fragment ciągu wyznaczany przez indeksy wskazywane przez pierwszy i drugi argument. Jeżeli nie zostanie podany drugi argument, przyjmuje się, że pobrane zostaną wszystkie znaki, począwszy od indeksu wskazywanego

przez pierwszy argument, aż do końca ciągu. Pod tym względem zachowuje się tak samo jak `slice`. Jednak, w odróżnieniu od `slice`, jeżeli pierwszy argument jest większy od drugiego (czyli indeks początkowy jest większy od końcowego), ich kolejność zostanie zamieniona. Zatem wywołanie `substring(4, 2)` zostanie potraktowane jak `substring(2, 4)`. Przykładowe wywołania:

```
str1 = "wizja pięknego świata".substring(15);
str2 = "wizja pięknego świata".substring(11, 14);
str3 = "wizja pięknego świata".substring(20, 15);
```

spowodują:

- przypisanie zmiennej `str1` ciągu `świata`;
- przypisanie zmiennej `str2` ciągu `ego`;
- przypisanie zmiennej `str3` ciągu `świat`.

Działanie metod `toLowerCase` i `toUpperCase` jest analogiczne, choć działają przeciwnie. Pierwsza zamienia wszystkie litery ciągu na małe, a druga — na wielkie. Dzieje się to, niezależnie od tego, jaka była wielkość liter w ciągu oryginalnym. Zwracany jest ciąg przetworzony, a ciąg oryginalny nie jest zmieniany. Znaczy to, że instrukcja:

```
str1 = "Wielki Zderzacz Hadronów".toLowerCase();
```

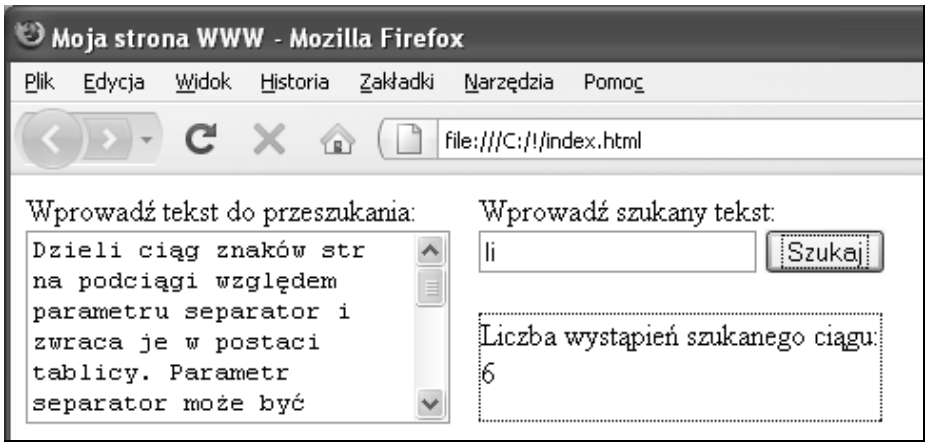
spowoduje przypisanie zmiennej `str1` ciągu `wielki zderzacz hadronów`, a instrukcja

```
str2 = "Wielki Zderzacz Hadronów".toUpperCase();
```

przypisanie zmiennej `str2` ciągu `WIELKI ZDERZACZ HADRONÓW`.

Użycie metod operujących na ciągach

Skoro poznaliśmy już cały zestaw metod operujących na ciągach znaków, użyjmy choć jednej z nich w praktycznie działającym przykładzie. Umożliwimy użytkownikowi witryny wprowadzenie dłuższego tekstu i automatyczne wliczenie, ile razy występuje w nim dowolna fraza — czyli dowolny ciąg znaków. Strona będzie wyglądała tak, jak zostało to zaprezentowane na rysunku 5.3. Znajduje się na niej rozszerzone pole tekstowe, zwykłe pole tekstowy, przycisk oraz warstwa prezentująca wyniki. Kod (X)HTML takiej witryny umieszczono na listingu 5.6.



Rysunek 5.3. *Witryna zliczająca liczbę wystąpień dowolnej frazy w danym tekście*

Listing 5.6. *Sekcja body kodu HTML opisanej witryny*

```
<body>
  <div style="margin-right:15px;float:left;">
    <div>Wprowadź tekst do przeszukania:</div>
    <div><textarea id="ta1" cols="25" rows="5"
         ></textarea></div>
  </div>
  <div style="float:left;">
    <div>Wprowadź szukany tekst:</div>
    <div style="margin-bottom:20px;">
       <input type="text" id="tf1">
       <input type="button" id="btn1"
           value="Szukaj" onclick="szukaj();">
    </div>
    <div id="divWynik"
         style="width:210px;height:55px;border:1px dotted black;">
    </div>
  </div>
</body>
```

Do formatowania układu strony zostały użyte warstwy — dwie główne tworzą układ kolumnowy. Na pierwszej znajduje się rozszerzone pole tekstowe zdefiniowane za pomocą znacznika `<textarea>`, któremu został nadany identyfikator `ta1`. Na drugiej umieszczone zostało zwykłe pole tekstowe o identyfikatorze `tf1`, przycisk oraz warstwa `divWynik` służąca do prezentacji wyników. Warstwa `divWynik` została wyróżniona przez zastosowanie kropkowanego (`dotted`) obramowania oraz ustalenie konkretnych rozmiarów (210×55 pikseli). Przyciskowi została przypisana procedura zdarzenia `onclick` w postaci funkcji `szukaj`, która zajmie się przeszukaniem tekstu wprowadzonego do pola `ta1`. Jej treść jest widoczna na listingu 5.7.

Listing 5.7. *Treść funkcji szukaj*

```
<script type="text/javascript">
  function szukaj()
  {
    var ta1 = document.getElementById('ta1');
    var tf1 = document.getElementById('tf1');
    var divWynik = document.getElementById('divWynik')
    var komunikat = "Liczba wystąpień szukanego ciągu: ";

    var tekst = ta1.value;
    var szukanyTekst = tf1.value;

    if((tekst == "") || (szukanyTekst == ""))
      return;

    var indeks = 0;
    var indeks_wystąpienia = 0;
    var liczba_wystąpień = 0;
```

```
    while (indeks_wystąpienia != -1){
      indeks_wystąpienia = tekst.indexOf(szukanyTekst, indeks);
      if (indeks_wystąpienia != -1){
        indeks = indeks_wystąpienia + 1;
        liczba_wystąpień++;
      }
    }
    divWynik.innerHTML = komunikat + liczba_wystąpień;
  }
</script>
```

Funkcja pobiera odwołania do wszystkich niezbędnych elementów strony: rozszerzonego pola tekstowego `ta1`, pola tekstowego `tf1` oraz warstwy `divWynik` i zapisuje je w zmiennych o takich samych nazwach. Definiuje również zmienną `komunikat` zawierającą podstawową treść wyświetlanego jako wynik komunikatu. W dalszej części skryptu zostanie ona uzupełniona o dane dotyczące liczby wystąpień poszukiwanego ciągu. Następnie funkcja odczytuje tekst do przeszukania (właściwość `value` pola `ta1`) oraz tekst poszukiwany (właściwość `value` pola `tf1`). Dane te zapisuje w zmiennych `tekst` oraz `szukanyTekst`.

W kolejnym kroku za pomocą instrukcji warunkowej `if` bada, czy któraś z tych zmiennych zawiera pusty ciąg znaków (`if((tekst == "") || (szukanyTekst == ""))`). Jeśli tak jest, oznacza to, że albo nie ma czego przeszukiwać, albo nie ma czego poszukiwać. W takiej sytuacji działanie funkcji jest przerywane za pomocą instrukcji `return`. Jeżeli jednak istnieją teksty zarówno poszukiwany, jak i przeszukiwany, zerowane są zmienne pomocnicze: `indeks`, `indeks_wystąpienia` i `liczba_wystąpień`. Ich znaczenie jest następujące:

- `indeks` — indeks znaku, od którego w danym kroku ma się zacząć przeszukiwanie tekstu głównego;
- `indeks_wystąpienia` — indeks, w którym w danym kroku został znaleziony poszukiwany ciąg znaków;
- `liczba_wystąpień` — całkowita liczba wystąpień poszukiwanego ciągu.

Przeszukiwanie tekstu odbywa się w pętli `while`. Jest wykonywana tak długo, jak długo zmienna `indeks_wystąpienia` jest różna od –1, czyli dopóty, dopóki w tekście przeszukiwanym można odnaleźć tekst poszukiwany. Pierwszą instrukcją wnętrza pętli jest:

```
indeks_wystąpienia = tekst.indexOf(szukanyTekst, indeks);
```

To wywołanie metody `indexOf` dla ciągu zawartego w zmiennej `tekst`, czyli dla tekstu przeszukiwanego. Pierwszym argumentem jest wartość zmiennej `szukanyTekst`, czyli poszukiwany tekst, a drugim — wartość zmiennej `indeks`, czyli indeks znaku, od którego ma się rozpocząć przeszukiwanie (początkowo — 0).

Jeżeli indeks wystąpienia jest różny od –1:

```
if (indeks_wystąpienia != -1){
```

oznacza to, że tekst poszukiwany został odnaleziony na pozycji wskazywanej przez zmienną `indeks_wystąpienia`. Skoro tak, poszukiwanie kolejnego wystąpienia należy rozpocząć w następnym znaku, a więc trzeba ustawić zmienną `indeks` tak, by pokazywała kolejny znak za wskazywanym przez `indeks_wystąpienia`. (Czy aby na pewno jest to optymalne rozwiązanie? Można to sprawdzić w rozwiązaniu ćwiczenia 17.5). Ustawmy zatem zmienną `indeks`:

```
indeks = indeks_wystąpienia + 1;
```

oraz zwiększmy wartość zmiennej `liczba_wystąpień` o jeden:

```
liczba_wystąpień++;
```

Po zakończeniu pętli w zmiennej `liczba_wystąpień` będzie się znajdowała całkowita liczba wystąpień ciągu poszukiwanego w ciągu przeszukiwanym. Jest ona dodawana do komunikatu zapisanego w zmiennej `komunikat`, a całość jest przypisywana właściwości `innerHTML` warstwy `divWynik`. Dzięki temu wynik poszukiwań pojawia się na ekranie, co zostało zaprezentowane na znajdującym się na początku tej części lekcji rysunku 5.3.

Ćwiczenia do samodzielnego wykonania

Ćwiczenie 17.1.

Zmień kod z listingu 5.3, tak by zamiast zdarzenia `onkeyup` było używane zdarzenie `onkeydown`. Sprawdź, jak zmieniło się działanie skryptu. Zastanów się, dlaczego wyniki nie są poprawne.

Ćwiczenie 17.2.

Zmień kod z listingu 5.5 tak, by nie korzystał z metod formatujących ciągi z tabeli 5.1, ale też nie zmienił się efekt działania skryptu.

Ćwiczenie 17.3.

W oparciu o przykład z listingów 5.4 i 5.5 napisz kod strony, która umożliwi zastosowanie do tekstu wprowadzonego do pola tekstowego dowolnej kombinacji czterech efektów generowanych przez metody formatujące. Zamiast pól wyboru typu radio użyj pól typu `checkbox`.

Ćwiczenie 17.4.

Napisz własną wersję metody `substring`. Nie używaj innych metod niż `substr`.

Ćwiczenie 17.5.

Kod z listingu 5.7 realizuje rodzaj algorytmu przeszukiwania liniowego. Zastanów się, czy jest optymalny, czy też można zwiększyć jego wydajność. Podpowiedź: przyjrzyj się dokładniej instrukcji `indeks = indeks_wystąpienia + 1;`, być może da się ją w prosty sposób usprawnić.

Lekcja 18. Wprowadzanie danych przez użytkownika

W lekcji 17. poznaliśmy rozmaite elementy witryny pozwalające na wybór różnych opcji bądź też wprowadzenie danych przez użytkowników. Wszystkie mogą być używane w formularzach WWW. Co prawda, dane z formularzy najczęściej przesyłane są do serwera, co już wykracza poza ramy tematyczne tej książki, jednak każdy programista JavaScript powinien wiedzieć, jak obsługiwać te elementy i sprawdzać poprawność uzyskiwanych za ich pomocą danych. W lekcji 18. omówimy więc formularze, sposoby walidacji danych, a także przetwarzanie informacji uzyskiwanych od użytkowników witryny.

Formularze

Formularz to typowy element witryny, który tworzymy, używając znacznika `form` z odpowiednimi atrybutami. Schematycznie wygląda to następująco:

```
<form name="nazwa"
     target="okno"
     action="url"
     method="metoda"
     enctype="typ_kodowania">
<!-- tutaj elementy formularza -->
</form>
```

gdzie `nazwa` jest po prostu nazwą formularza. Parametr `target` określa nazwę okna, w którym ma się pojawić odpowiedź otrzymana z serwera, może on zawierać nazwę okna lub ramki, `action` ustala lokalizację serwera, do którego mają zostać wysłane dane zebrane z formularza, zwykle jest to adres skryptu PHP, ASP lub podobnego, `method` — sposób wysłania informacji do serwera (zwykle `post` lub `get`), natomiast `enctype` — sposób kodowania `MIME`.

Dostęp do obiektów formularzy zawartych na stronie HTML można uzyskać, odwołując się do właściwości `forms` obiektu `document` (tablica 4.4 w lekcji 12.). Całkowitą liczbę formularzy uzyskamy, odczytując właściwość `lenght`, np.:

```
var liczba_formularzy = document.forms.length;
```

Natomiast dostęp do poszczególnych formularzy można uzyskać poprzez odwołania w postaci:

```
document.forms["nazwa_formularza"]
document.forms.nazwa_formularza
```

lub:

```
document.forms[indeks_formularza]
```

gdzie *nazwa_formularza* to nazwa zdefiniowana za pomocą atrybutu `name`, natomiast *indeks_formularza* to indeks kolejnego formularza w dokumencie (z reguły formularze indeksowane są w kolejności ich umieszczania w dokumencie HTML, indeks pierwszego formularza to 0). Jeżeli w definicji formularza użyjemy atrybutu `id` (nadamy mu identyfikator), odwołanie do obiektu formularza, oprócz wyżej wymienionych metod, będzie można również uzyskać przy użyciu metody `getElementById` (tak samo jak w przypadku każdego innego elementu strony), np.:

```
var formularz = dokument.getElementById('identyfikator');
```

Obiekt formularza udostępnia właściwości zebrane w tabeli 5.4 i metody zebrane w tabeli 5.5.

Tabela 5.4. *Właściwości obiektu formularza*

Nazwa	Znaczenie
`acceptCharset`	Pobiera lub ustala listę typów kodowań znaków możliwych do zastosowania dla danych wprowadzanych do formularza (lista obsługiwanych stron kodowych).
`action`	Pobiera lub ustala wartość atrybutu `action` (określenie, gdzie mają zostać wysłane dane z formularza).
`className`	Pobiera lub ustala wartość atrybutu `class`.
`dir`	Pobiera lub ustala określenie kierunku tekstu.
`encoding`	Pierwotna nazwa właściwości `enctype` w starszych wersjach przeglądarek. Obecnie nie należy jej stosować.
`enctype`	Pobiera lub ustala typ `MIME` używany do kodowania zawartości formularza (wartość atrybutu `enctype`).
`elements`	Tablica zawierająca elementy składowe formularza.
`id`	Pobiera lub ustala identyfikator formularza (wartość atrybutu `id`).
`lang`	Pobiera lub ustala kod języka danego elementu.
`length`	Zawiera liczbę elementów składowych formularza.
`method`	Określa sposób wysyłania danych do serwera (wartość atrybutu `method`).
`name`	Pobiera lub ustala nazwę formularza (wartość atrybutu `name`).
`target`	Wartość atrybutu `target`, określenie okna lub ramki, do której mają zostać wysłane dane.
`title`	Pobiera lub ustala treść podpowiedzi dla formularza.

Tabela 5.5. *Metody obiektu form*

Metoda	Wywołanie	Opis
reset	reset()	Wykonuje reset formularza, czyli przywraca go do stanu wyjściowego. Działa więc tak samo jak przycisk *reset*.
submit	submit()	Powoduje wysłanie zawartości formularza do serwera. Działa tak samo jak przycisk *submit*.

Utwórzmy stronę zawierającą prosty formularz i napiszmy skrypt odczytujący podstawowe informacje o nim. Kod witryny przyjmie postać widoczną na listingu 5.8.

Listing 5.8. *Strona zawierająca formularz*

```
<body>
  <form name="form_nr_1"
        action="http://moja.domena/form.php"
        method="get"
        style="width:250px;">
  <div style="border: 1px solid black;margin:10px;
              width:250px;padding:10px;float:left;">
    <input type="radio" name="grp1" value="1" />Opcja nr 1
    <input type="radio" name="grp1" value="2" />Opcja nr 2
    <br>
    Wpisz tekst: <input type="text" name="txt1" value="" />
  </div>
  </form>
  <div style="margin-bottom:10px;">
    <input type="button"
           value="Odczytaj dane dotyczące formularza"
          onclick="odczytaj();" />
  </div>
  <div id="dataDiv" style="float:left;"></div>
</body>
```

Formularz zdefiniowano za pomocą znacznika <form>. Określona została jego nazwa (atrybut name — form_nr_1), adres hipotetycznego skryptu PHP przetwarzającego dane (atrybut action — http://moja.domena/form.php) oraz metoda wysyłania danych do serwera (atrybut method — get). W formularzu znalazły się trzy elementy: dwa pola wyboru typu radio oraz zwykłe pole tekstowe. Wszystkie elementy zostały zdefiniowane za pomocą znacznika <input>.

Za formularzem znajduje się warstwa zawierająca przycisk. Kliknięcie przycisku będzie powodowało odczytanie informacji o formularzu oraz wyświetlenie ich na przeznaczonej do tego celu warstwie (ostatnia warstwa o identyfikatorze dataDiv). W związku z tym, zdarzeniu onclick przycisku została przypisana procedura obsługi w postaci funkcji odczytaj. Treść tej funkcji została przedstawiona na listingu 5.9.