Spis Treści

Rozdział 1.

Podstawy

Czym jest JavaScript?

JavaScript tak naprawdę narodził się w firmie Netscape jako LiveScript, język skryptowy rozszerzający standardowy HTML m.in. o możliwość interakcji z użytkownikiem przeglądającym stronę. Nieco później doszło do porozumienia między firmami Netscape i Sun Microsystems, w wyniku którego pod koniec 1995 roku światło dzienne ujrzał JavaScript. Język ten umożliwia tworzenie zagnieżdżonych bezpośrednio w kodzie HTML krótkich programów, które potrafią rozpoznać i odpowiednio zareagować na zdarzenia powodowane przez użytkownika. Zdarzenia te to np. kliknięcie myszą, wypełnianie formularza, czy nawigowanie między stronami. Przykładowo, można stworzyć skrypt, który będzie sprawdzał poprawność danych wprowadzonych przez użytkownika do formularza (np. czy wprowadzając jakąś datę, nie przekroczyliśmy dopuszczalnej liczby dni w danym miesiącu) i który będzie informował o ewentualnym błędzie. Sprawdzenie takie odbywać się będzie na komputerze przeglądającego stronę, nie nastąpi więc konieczność dodatkowej transmisji danych w sieci. Sprawdzaniem danych nie będzie musiał też zajmować się serwer.

JavaScript a Java

JavaScript, jak sama nazwa wskazuje, ma sporo wspólnego z językiem programowania Java, opracowanym w firmie Sun Microsystems. Niemniej nie należy tych dwóch języków mylić. Przede wszystkim zostały one stworzone do zupełnie różnych celów. Java to wywodzący się m.in. z C++ w pełni obiektowy język programowania, za pomocą którego można tworzyć skomplikowane aplikacje niezależne od platformy sprzętowej. JavaScript jest interpretowanym językiem skryptowym, służącym do tworzenia niewielkich programów rozszerzających możliwości HTML-u w zakresie opisu stron WWW. Krótkie zestawienie najważniejszych cech Javy i JavaScript znajduje się w tabeli 1.1.

Tabela 1.1. *Podstawowe różnice pomiędzy językami Java a JavaScript*

JavaScript	Java
Język interpretowany na komputerze klienta	Język kompilowany do tzw. b-kodu, wykonywanego następnie za pomocą wirtualnej maszyny Javy na komputerze klienta
Język oparty na predefiniowanych obiektach, niepozwalający jednak na stosowanie mechanizmów programowania obiektowego jak np. dziedziczenie	Język zorientowany obiektowo z obsługą wszystkich mechanizmów obiektowości
Kod programu jest zagnieżdżony w kodzie HTML	Kod programu jest niezależny od kodu HTML i znajduje się w oddzielnych plikach
Zmienne i ich typ nie muszą być deklarowane przed użyciem	Zmienne i ich typ muszą być zadeklarowane przed ich użyciem w programie
Odwołania do obiektów i funkcji są wykonywane podczas uruchamiania programu	Wszystkie odwołania do obiektów i funkcji są sprawdzane na etapie kompilacji
Ze względów bezpieczeństwa nie ma możliwości zapisu na dysk twardy	Ze względów bezpieczeństwa aplety, (w przeciwieństwie do aplikacji) nie mają możliwości zapisu na dysk twardy

Co nam będzie potrzebne?

Przede wszystkim dobre chęci. Oprócz tego żadne specjalne narzędzia nie będą przydatne. Musimy oczywiście mieć zainstalowaną przeglądarkę WWW. Najlepiej Microsoft Internet Explorer lub Netscape Navigator. Nie muszą to być najnowsze wersje, niemniej jednak nie niższe niż 3.0.

Do pisania samego kodu potrzebny będzie dowolny, najprostszy edytor tekstowy np. systemowy Notatnik. Będzie to nasz warsztat pracy.

Potrzebna będzie też przynajmniej podstawowa znajomość HTML-u, nie jest natomiast konieczna znajomość innych języków programowania.

Rozdział 2.

Pierwsze skrypty

Na początku zajmijmy się klasycznym przykładem, od którego zaczyna się większość kursów programowania. Postarajmy się wyświetlić na ekranie dowolny napis np. `Jaki miły mamy dzień!`. Aby tego dokonać, wpierw musimy dowiedzieć się, w jaki sposób umieszczać skrypty JavaScript w kodzie HTML oraz jaka instrukcja JavaScript pozwala pisać na ekranie.

Znacznik <SCRIPT>

Kod JavaScript musi być umieszczony pomiędzy znacznikami HTML `<SCRIPT>` i `</SCRIPT>`. Znaczniki te można umieszczać w dowolnym miejscu dokumentu, jednak przyjmuje się, że jeżeli jest to tylko możliwe, należy umieścić je na początku pliku HTML przed znacznikiem `<BODY>`.

Znacznik ten powinien zawierać parametr `LANGUAGE`, który może przyjmować dwie wartości: `LiveScript` lub `JavaScript`. Wartość `LiveScript` jest pozostałością po wczesnych wersjach języka i służy zachowaniu kompatybilności. Powinniśmy użyć wartości `JavaScript`.

Ćwiczenie 2.1.

Umieść w standardowym kodzie HTML znacznik `<SCRIPT>`.

```
<HTML>
<HEAD>
</HEAD>
<SCRIPT language = "JavaScript">
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Na listingu znajduje się poprawny kod HTML z zawartym znacznikiem `<SCRIPT>`. Jednak po wczytaniu tego pliku do przeglądarki otrzymamy pustą stronę. Brakuje nam instrukcji pozwalającej wyświetlać tekst.

Instrukcja document.write

Instrukcja `document.write()` pozwala na wyprowadzenie tekstu na ekran przeglądarki. Tekst, który chcemy wyświetlić, należy ująć w nawiasy i cudzysłowy i podać zaraz za `document.write()` np.

```
document.write ("Jaki miły mamy dzień!")
```

Ćwiczenie 2.2.

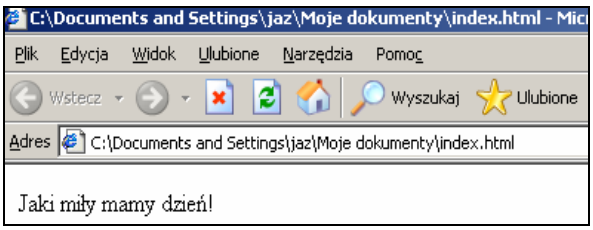
Napisz skrypt wyświetlający tekst „`Jaki miły mamy dzień!`” na ekranie przeglądarki.

```
<HTML>
<HEAD>
```

```
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT language = "JavaScript">
document.write ("Jaki miły mamy dzień!")
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Tak przygotowany kod spowoduje, że na ekranie pojawi się pożądany napis (rysunek 2.1). Warto zwrócić uwagę, że w celu poprawnej interpretacji polskich liter przez przeglądarkę dodaliśmy w sekcji `HEAD` znacznik `<META http-equiv="Content-Type" content="text/ html; charset=iso-8859-2">`

Rysunek 2.1. *Efekt działania instrukcji document.write()*



Przeanalizujmy nieco dokładniej fragment kodu odpowiedzialny za wyświetlanie tekstu na ekranie. Wszystkim, którzy mieli już wcześniej do czynienia z językiem C bądź C++, składnia wydaje się z pewnością znajoma:

```
document.write ("Jaki miły mamy dzień")
```

`document` to obiekt, który reprezentuje aktualną stronę. `write` to tzw. metoda, czyli pewna funkcja działająca na obiekcie `document` i, w tym przypadku, wyświetlająca na ekranie tekst. Tekst ten podajemy jako argument w nawiasach. Ogólnie można zapisać:

```
obiekt.metoda (argumenty metody)
```

Taki ciąg jest instrukcją i powinien zostać zakończony średnikiem. W JavaScript nie jest to jednak obligatoryjne, chyba że chcemy zapisać kilka instrukcji w jednej linii np.:

```
document.writeln (”Witamy”);document.write (”na naszej stronie”);
```

Wymieniona tutaj, nowa funkcja `writeln()` działa tak samo jak `write()`, z tym że na końcu wyświetlanego ciągu znaków dodaje znak przejścia do nowego wiersza. Niestety, nie zobaczymy tego efektu, jeżeli całość nie znajdzie się w bloku tekstu preformatowanego, tzn. pomiędzy znacznikami `<PRE>` i `</PRE>`.

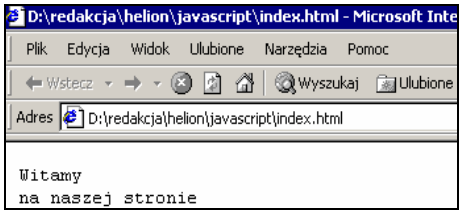
Ćwiczenie 2.3.

Użyj funkcji `write()` i `writeln()` do wyświetlenia tekstu w dwóch wierszach.

```
<HTML>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<HEAD>
<PRE>
<SCRIPT>
document.writeln ("Witamy");document.write ("na naszej stronie");
</SCRIPT>
</PRE>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Jak widać na rysunku 2.2, zadanie udało nam się wykonać znakomicie.

Rysunek 2.2. *Użycie instrukcji writeln() i znacznika <PRE>*



Komentarze

Komentarz HTML

Znacznik <SCRIPT>, niezbędny do umieszczania kodu JavaScript, niestety nie jest częścią specyfikacji HTML 2.0, ani wcześniejszych, więc niektóre przeglądarki mogą go nie rozpoznać. W takiej sytuacji mogą one wyświetlić tekst skryptu na stronie. Chcielibyśmy oczywiście tego uniknąć. Z pomocą przyjdą komentarze, które można umieszczać w kodzie HTML. Konstrukcja wygląda następująco:

```
<!--
Tekst komentarza
-->
```

Jeżeli zatem chcemy ukryć kod przed przeglądarkami nieobsługującymi JavaScript, powinniśmy ująć go w znaki komentarza, które są częścią standardu HTML.

Znacznik <SCRIPT>, niezbędny do umieszczania kodu JavaScript, niestety nie jest częścią specyfikacji HTML 2.0, ani wcześniejszych, więc niektóre przeglądarki mogą go nie rozpoznać. Co się stanie w takiej sytuacji? Otóż sam znacznik zostanie zignorowany, natomiast cały tekst skryptu znajdujący się między <SCRIPT> a </SCRIPT> zostanie wyświetlony na ekranie, zmieniając nam treść i strukturę strony. Chcielibyśmy oczywiście tego uniknąć. Z pomocą przyjdzie nam komentarz HTML, którego struktura wygląda następująco:

```
<!--
Tekst komentarza
-->
```

Jeżeli ujmiemy tekst skryptu w taką strukturę, przeglądarka nieobsługująca JavaScriptu pominie go, traktując właśnie jako zwykły komentarz.

Ćwiczenie 2.4.

Ukryj kod skryptu przed przeglądarkami nieobsługującymi JavaScript.

```
<HTML>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
document.write ("Jaki miły mamy dzień!")
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Powyższe ćwiczenie obrazuje użycie komentarzy znanych z języka HTML. W JavaScript mamy natomiast dwie nowe możliwości zastosowania komentarza. Obie są zapożyczone z języków programowania takich C, C++ czy Java. Pierwszy typ komentarza składa się z dwóch ukośników: // (komentarz ten został zastosowany w poprzednim przykładzie, bowiem wczesne wersje przeglądarki Netscape Navigator nie rozpoznawały poprawnie sekwencji --> umieszczonej między etykietami <SCRIPT>). Zaczyna się on wtedy od miejsca wystąpienia tych dwóch znaków i obowiązuje do końca danego wiersza.

Komentarz typu //

Ćwiczenie 2.5.

Użyj komentarza składającego się z dwóch ukośników do opisania kodu skryptu.

```
<HTML>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
// Wyświetlenie napisu w oknie przeglądarki
document.write ("Hello, Jaki miły mamy dzień!")
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
```

```
</BODY>
</HTML>
```

Komentarz blokowy

Komentarz może się również zaczynać od sekwencji `/*` i kończyć `*/`. W takim przypadku wszystko, co znajduje się pomiędzy tymi znakami, uznane zostanie za komentarz.

Ćwiczenie 2.6.

Użyj komentarza blokowego do opisania kodu skryptu.

```
<HTML>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nie osbługującymi JavaScript
/*
Komentarz blokowy
Wyświetlenie napisu w oknie przeglądarki
*/
document.write ("Hello, Jaki miły mamy dzień!")
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Znacznik <NOSCRIPT>

W jaki sposób jednak poinformować użytkownika przeglądarki nieobsługującej JavaScriptu, że strona taki skrypt zawiera, tylko nie został wykonany? Z pomocą przyjdą nam również komentarze.

Ćwiczenie 2.7.

Napisz kod, który po wczytaniu do przeglądarki nieobsługującej JavaScript wyświetli stosowny komunikat.

```
<HTML>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<HEAD>
<SCRIPT LANGUAGE = "JavaScript">
// Twoja przeglądarka nie obsługuje JavaScript
// Sugerujemy użycie przeglądarki Netscape Navigator
// lub Microsoft Internet Explorer!
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
document.write ("Jaki miły mamy dzień!")
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Przeglądarka nieobsługująca skryptów po napotkaniu nieznanej sobie etykiety `<SCRIPT>` ignoruje ją, następnie wyświetla dwa kolejne wiersze, traktując je jako zwykły HTML. Następne wiersze są dla niej komentarzem, więc je pomija. Z kolei dla przeglądarki obsługującej skrypty komentarzem są dwa wiersze następujące po etykiecie `<SCRIPT>` i to one są pomijane, natomiast kod z piątego wiersza skryptu (`document.write („Jaki miły mamy dzień!")`) jest interpretowany i wykonywany.

Jest też jeszcze inny sposób na wykonanie tego zadania. Przeglądarki Netscape Navigator oraz Internet Explorer, obie od wersji 3.0, akceptują dodatkowy znacznik `<NOSCRIPT>`. Dzięki niemu możemy osiągnąć podobny efekt. W tym przypadku tekst, który ma być wyświetlony, gdy wyłączymy skrypty w danej przeglądarce, umieszczamy pomiędzy znacznikami `<NOSCRIPT>` i `</NOSCRIPT>`.

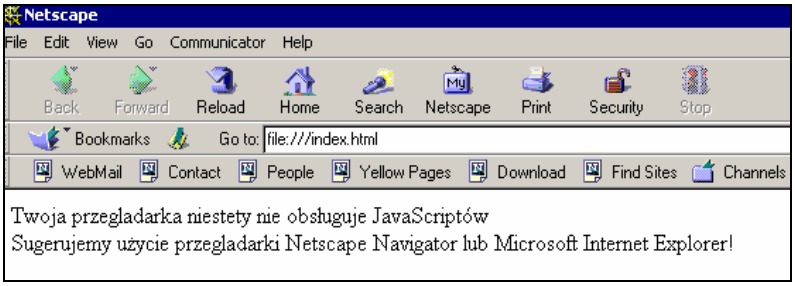
Ćwiczenie 2.8.

Użyj znacznika <NOSCRIPT> do poinformowania użytkownika, że jego przeglądarka nie obsługuje JavaScriptu.

```
<HTML>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
document.write ("Jaki miły mamy dzień!")
// Koniec kodu JavaScript -->
</SCRIPT>
<NOSCRIPT>
Twoja przeglądarka niestety nie obsługuje JavaScriptów.<BR>
Sugerujemy użycie przeglądarki Netscape Navigator lub Microsoft Internet Explorer!
</NOSCRIPT>
<BODY>
</BODY>
</HTML>
```

Na rysunku 2.3 widoczny jest efekt działania powyższego kodu w przeglądarce Netscape Navigator po wyłączeniu działania skryptów.

Rysunek 2.3. *Zastosowanie znacznika <NOSCRIPT> do poinformowania użytkownika, że jego przeglądarka nie obsługuje JavaScriptu*



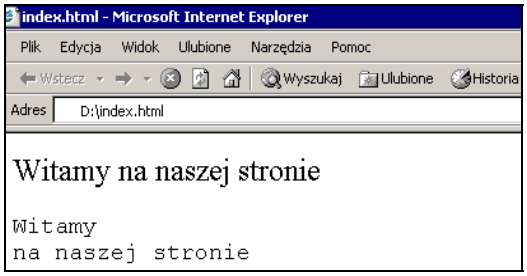
Formatowanie tekstu

Argumenty poznanych wyżej funkcji `write()` i `writeln()` są traktowane przez przeglądarkę jak tekst w HTML-u. Oznacza to, że możemy w łańcuchach wyświetlanych znaków wstawić praktycznie dowolne znaczniki formatujące tekst.

Ćwiczenie 2.9.

Użyj znaczników HTML formatujących tekst w argumentach funkcji `write()` i `writeln()`, tak by osiągnąć efekt jak na rysunku 2.4.

Rysunek 2.4. *Efekt użycia znaczników HTML w argumentach funkcji write() i writeln()*




```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScriptów
document.write ("<FONT SIZE=+2>Witamy ");
document.write ("na naszej stronie");
document.writeln ("<PRE>Witamy");
document.write ("na naszej stronie</PRE></FONT>");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Oprócz znaczników HTML w wyświetlanych łańcuchach znakowych mogą też pojawić się znaki specjalne, takie jak np. rozpoczęcie nowego wiersza. Jeśli chcemy wyświetlić znak specjalny, musimy zastosować sekwencję — ukośnik (backslash) plus litera symbolizująca dany znak. Sekwencje te przedstawione są w tabeli 2.1.

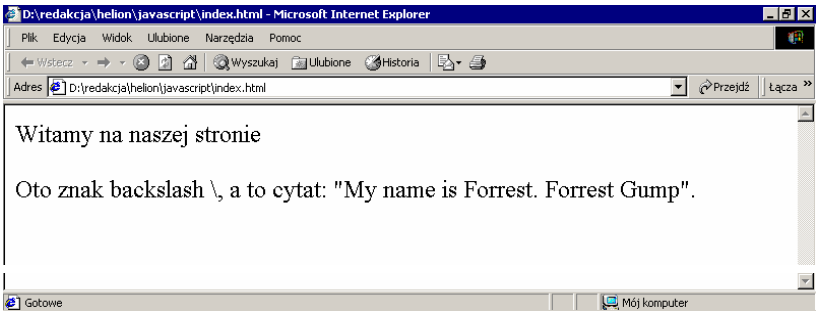
Tabela 2.1. *Sekwencje znaków specjalnych*

Sekwencja znaków specjalnych	Znaczenie
`\b`	Backspace
`\f`	wysunięcie kartki (ang. *form feed*)
`\n`	nowy wiersz (ang. *new line character*)
`\r`	enter (ang. *carriage return*)
`\t`	tabulator (ang. *tab character*)

Podobnie, jeżeli chcemy wyświetlić cudzysłów lub sam ukośnik (backslash \), musimy go poprzedzić znakiem backslash.

Ćwiczenie 2.10.

Używając funkcji `write()` wyprowadź na ekran tekst zawierający znak cudzysłowu oraz ukośnik (rysunek 2.5).



Rysunek 2.5. *Wyprowadzenie na ekran znaków specjalnych*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
document.write ("<FONT SIZE=+2>Witamy ");document.write ("na naszej stronie<BR><BR>");
document.write ("Oto znak backslash \\");document.write (", a to cytat: \"My name is Forrest. Forrest
Gump\".");
document.write ("</FONT>");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

W ten sam sposób możemy również pokusić się o wyświetlenie grafiki. Jeżeli argumentem funkcji `write()` będzie znacznik `<IMG>` z odpowiednim URL-em jako parametrem, przeglądarka wyświetli na stronie wskazany w ten sposób obrazek np.

```
document.write ("<IMG SRC = /javasc/gfx/grafika1.gif>");
```

Oczywiście, plik o lokalizacji */javasc/gfx/grafika1.gif* musi istnieć, abyśmy mogli zobaczyć efekt w oknie przeglądarki. Formalnie rzecz biorąc, powinniśmy wartość argumentu `SRC` ująć w cudzysłów, zatem zgodnie z tym, co wiemy już o znakach specjalnych, konstrukcja powinna wyglądać następująco:

```
document.write ("<IMG SRC = \"/javasc/gfx/grafika1.gif\">");
```

Ćwiczenie 2.11.

Użyj funkcji `write()` do wyświetlenia na ekranie pliku graficznego (rysunek 2.6).

Rysunek 2.6. *Przykład użycia funkcji write() do wyświetlenia pliku graficznego*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScriptu
document.write ("<IMG SRC = \"/javasc/gfx/grafika1.gif\">");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Okno dialogowe

Nauczymy się teraz, jak wyświetlić na ekranie najprostsze okienko dialogowe. Okno takie służy zwykle do poinformowania użytkownika o wystąpieniu jakiegoś zdarzenia. Najczęściej chodzi o sytuacje, w której wystąpił błąd. Na taki charakter prezentowanej metody wskazuje już sama nazwa: `alert()`. Może ona przyjmować jako parametr ciąg znaków, który zostanie wyświetlony na ekranie.

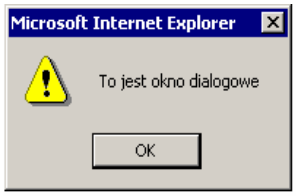
Ćwiczenie 2.12.

Wyświetl na ekranie okno dialogowe z dowolnym napisem.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScriptu
alert("To jest okno dialogowe");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Nasze okno wygląda jak na rysunku 2.7. Wykonywanie kodu jest wstrzymane do czasu, kiedy użytkownik kliknie przycisk *OK*. Dokładniej rzecz biorąc, w taki sposób powinna się zachować większość współczesnych przeglądarek. Tekst wyświetlany w oknie dialogowym możemy formatować, używając do tego celu znaków specjalnych (tabela 2.1), podobnie jak w przypadku funkcji `write()`.

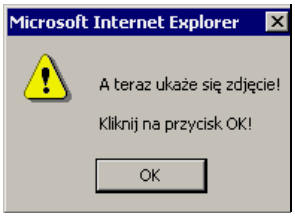
Rysunek 2.7.
Użycie funkcji alert() do wyświetlenia okna dialogowego



Ćwiczenie 2.13.

Wyświetl na ekranie okno dialogowe z tekstem w dwóch wierszach (jak na rysunku 2.8).

Rysunek 2.8.
Użycie znaków specjalnych formatujących tekst w oknie dialogowym



Spowoduj, aby po kliknięciu przez użytkownika przycisku *OK* na ekranie pojawił się plik graficzny.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
```

```
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScriptów
alert ("\nA teraz ukaże się zdjęcie!\n\nKliknij na przycisk OK!")
document.write ("<IMG SRC = \"obrazek1.gif\">");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Rozdział 3.

Elementy języka JavaScript

Typy danych JavaScript

Do dyspozycji mamy cztery typy danych:

- liczby,
- wartości logiczne,
- łańcuchy znaków,
- wartość `NULL`.

Typ liczbowy

Służy do reprezentowania wszelkiego rodzaju liczb. Odmiennie niż w innych językach programowania, jak np. C++, gdzie do reprezentacji liczb służy co najmniej kilka typów danych, tutaj używamy tylko jednego — liczbowego. Liczby całkowite — zarówno dodatnie, jak i ujemne — możemy przedstawić w postaci dziesiętnej, szesnastkowej lub ósemkowej. System dziesiętny jest nam wszystkim znany; używamy w nim dziesięciu cyfr. Przypomnijmy jednak pokrótce podstawy dwóch pozostałych systemów.

W systemie szesnastkowym podstawą jest oczywiście 16. Zatem do zapisu liczb w tym systemie nie wystarczy nam cyfr arabskich (0 – 9), których jest tylko dziesięć. Używamy więc dodatkowo sześciu liter: od A do F. Od zera do dziewięciu liczymy identycznie jak w znanym nam systemie dziesiętnym. Dalej jednak zamiast liczby dziesięć pojawia się litera A, potem B i tak dalej do F (co odpowiada 15 w systemie dziesiętnym). Po F następuje 10. W systemie ósemkowym natomiast, skoro podstawą jest liczba 8, używamy 8 cyfr — od 0 do 7. Po liczbie 7 następuje 10, potem 11 i tak dalej do 17, po której następuje 20. Odpowiadające sobie liczby w systemie dziesiętnym, szesnastkowym i ósemkowym przedstawia tabela 3.1.

Tabela 3.1. *Reprezentacja liczb w różnych systemach*

System ósemkowy	System dziesiętny	System szesnastkowy
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
10	8	8
11	9	9
12	10	A
13	11	B
14	12	C

15	13	D
16	14	E
17	15	F
20	16	10

Wróćmy jednak do reprezentacji liczb w JavaScripcie. Liczby dziesiętne zapisujemy w sposób tradycyjny, czyli np. 45, –124, 860. Liczby w systemie szesnastkowym musimy poprzedzić znakami **0x** lub **0X**, np. szesnaście zapiszemy jako 0x10 (lub 0X10), dwadzieścia jeden jako 0x15 (lub 0X15). W systemie ósemkowym przed liczbą dodajemy po prostu zero, np. 010 (osiem dziesiętnie), 024 (dwadzieścia dziesiętnie).

Drugi typ liczb reprezentowanych w JavaScripcie to liczby zmiennopozycyjne, czyli z częścią ułamkową. Może to być klasyczny sposób zapisu z kropką dziesiętną, np. 1.274, –56,8 bądź zapis wykładniczy typu 2E3, –123E–2 itp. Nie będziemy się tutaj zagłębiać w dalsze niuanse reprezentacji liczb. Warto tylko pamiętać, że zakres liczb, tzn. największa i najmniejsza wartość, które mogą one przyjmować, jest zależny od systemu, na którym pracujemy.

Wartości logiczne

Zmienne tego typu mogą przyjmować tylko dwie wartości: `TRUE` i `FALSE` (prawda i fałsz). Będą one używane przy konstruowaniu wyrażeń logicznych, porównywania danych, wskazania, czy dana operacja zakończyła się sukcesem. Dla osób znających C czy C++ uwaga: wartości `TRUE` i `FALSE` nie mają przełożenia na wartości liczbowe, jak w przypadku wymienionych języków.

Łańcuchy znaków

Są to oczywiście dowolne ciągi znaków zawartych pomiędzy znakami cudzysłowów lub apostrofów. Mogą zawierać znaki specjalne. Przykładami mogą być: `"Kot ma Alę"`, `"liczby pierwsze: 1 3 5... "`.

Wartość NULL

Jest to pewien specjalny typ danych, który oznacza po prostu nic (`null`). Wartość ta jest zwracana przez niektóre funkcje. Bliżej zapoznamy się z nią w dalszej części książki.

Zmienne

Poznaliśmy już typy danych, czas zapoznać się ze sposobami deklarowania i wykorzystania zmiennych. Zmienne są to konstrukcje programistyczne, które pozwalają nam przechowywać dane. Każda zmienna na swoją nazwę, która ją jednoznacznie identyfikuje. Zwykle (w większości języków programowania) zmienna musi mieć określony typ danych, który jest jej przypisany na stałe. Czasami, np. w języku programowania Pascal, musi być jawnie zadeklarowana przed użyciem. Ponieważ jednak JavaScript jest stosunkowo prostym językiem skryptowym, nie ma narzuconych takich ograniczeń. Zmiennych bowiem nie musimy (aczkolwiek możemy) deklarować przed użyciem; każda zmienna może też przyjmować dane z dowolnego typu opisanego wyżej. Co więcej, typ danych przypisywanych zmiennej może się również zmieniać. Wszystko stanie się jaśniejsze po wykonaniu pierwszego ćwiczenia.

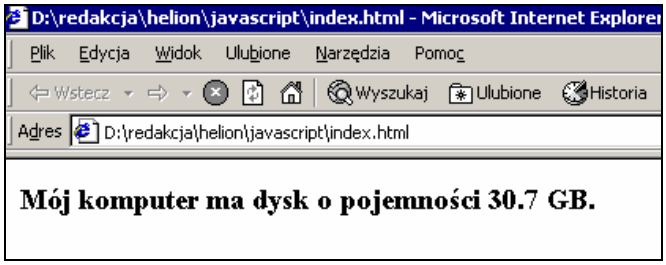
Ćwiczenie 3.1.

Zadeklaruj dwie zmienne, przypisz im dowolne ciągi znaków i wyprowadź je na ekran za pomocą funkcji `write()`.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var zmienna1 = "Mój komputer";
var zmienna2 = 30.7;
document.write ("<H3>" + zmienna1 + " ma dysk o pojemności " + zmienna2 + " GB.</H3>");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
```

```
</HTML>
```

Po wczytaniu takiej strony na ekranie ukaże się napis „Mój komputer ma dysk o pojemności 30.7 GB” (rysunek 3.1). Przeanalizujmy więc, co się tutaj stało. Zadeklarowaliśmy dwie zmienne o nazwach `zmienna1` i `zmienna2`. Zmiennej `zmienna1` przypisaliśmy ciąg znaków „Mój komputer”, zmiennej `zmienna2` natomiast wartość liczbową, dodatnią liczbę zmiennoprzecinkową `30.7`. Zmiennych tych użyliśmy jako argumentów funkcji `write()`. Musieliśmy również tak połączyć poszczególne łańcuchy tekstowe, aby otrzymać jeden, który ukazał się na ekranie. Do tego celu użyliśmy operatora + (plus). Nazywa się to łączeniem lub bardziej fachowo konkatenacją łańcuchów znakowych.



Rysunek 3.1. *Wyprowadzenie na ekran wartości dwóch zmiennych*

Przekonajmy się teraz, że w JavaScripcie naprawdę można zmieniać typ danych przechowywanych w zmiennej.

Ćwiczenie 3.2.

Zadeklaruj jedną zmienną. Przypisz do niej dowolny łańcuch znaków i wyprowadzić na ekran. Następnie przypisz tej samej zmiennej wartość liczbową i również wyprowadź na ekran.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var zmienna1 = "Mój komputer";
document.write ("<H3>" + zmienna1 + " ma dysk o pojemności ");
zmienna1 = 30.7;
document.write (zmienna1 + " GB.</H3>");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Efekt będzie taki sam jak na rysunku 3.1. W stosunku do skryptu z poprzedniego ćwiczenia obecny kod jest dłuższy i chyba mniej przejrzysty, niemniej w niektórych sytuacjach użycie jednej zmiennej do kilku różnych funkcji może być przydatne. Warto zwrócić też uwagę, że lepiej byłoby nadawać zmiennym nazwy, które w jakiś sposób symbolizowałyby ich przeznaczenie. Przy tak prostych skryptach, jak w dwóch powyższych ćwiczeniach, nie ma to zapewne wielkiego znaczenia. Jednak przy bardziej skomplikowanych programach, z większą ilością zmiennych, takie nazewnictwo doprowadzi do sytuacji, w której nie będziemy w stanie zorientować się, o co nam w programie chodziło.

Wprowadzanie danych

Ćwiczenie 3.3.

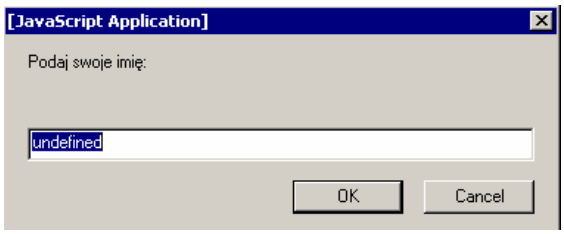
Wyświetl na ekranie okno pozwalające użytkownikowi na podanie np. jego imienia.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
prompt ("Podaj swoje imię: ");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Po wczytaniu tego skryptu na ekranie pojawi się okno dialogowe z pytaniem o imię. Okno to będzie miało nieco inny wygląd w różnych przeglądarkach, ale zawsze będzie pozwalało na wprowadzenie danych przez użytkownika. Wygląd okna w przeglądarce Internet Explorer widoczny jest na rysunku 3.2, natomiast w przeglądarce Netscape Navigator na rysunku 3.3.

Rysunek 3.2. *Efekt działania funkcji prompt() w przeglądarce Internet Explorer*

Rysunek 3.3. *Efekt działania funkcji prompt() w przeglądarce Netscape Navigator*



Niestety w takiej wersji, jak w powyższym ćwiczeniu, skrypt nie potrafi nic zrobić z odpowiedzią użytkownika. Należy więc go odpowiednio zmodyfikować.

Ćwiczenie 3.4.

Wyświetl na ekranie okno dialogowe pozwalające na podanie przez użytkownika imienia. Następnie wyprowadź na ekran napis powitalny zawierający podane imię.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var imie = prompt ("Podaj swoje imię:");
document.write ("Cześć " + imie + "!");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Zadeklarowaliśmy zmienną o nazwie `imie`, której przypisaliśmy wartość zwracaną przez funkcję `prompt()`. Następnie wypisaliśmy wartość tej zmiennej, razem z tekstem powitania na ekran. Nie musimy jednak wcale deklarować zmiennej, aby uzyskać taki sam efekt. Wystarczy, że wywołanie funkcji `prompt()` umieścimy w argumencie funkcji `write()`.

Ćwiczenie 3.5.

Wyświetl na ekranie okno dialogowe pozwalające na podanie przez użytkownika swojego imienia. Następnie, nie używając zmiennych, wyprowadź na ekran napis powitalny zawierający podane imię.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
document.write ("Cześć " + prompt ("Podaj swoje imię:") + "!");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

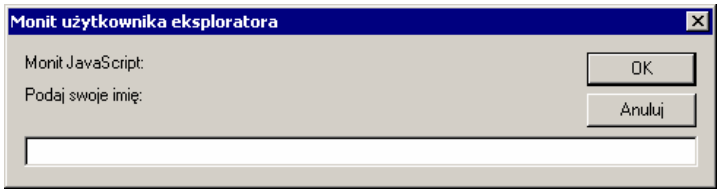
Kod ten jest co prawda mniej czytelny, niż gdybyśmy użyli zmiennej pomocniczej, uzyskaliśmy jednak bardziej zwięzły zapis.

We wszystkich powyższych przykładach w okienku dialogowym jako wartość domyślna pojawia się napis „`Undefined`" (rysunki 3.1, 3.2, 3.3). Nie jest to pożądany przez nas efekt; na szczęście można to zmienić. Jeśli chcemy, aby wartością domyślną było coś innego, dowolny napis lub liczba, musimy podać ją jako drugi argument funkcji `prompt()`.

Ćwiczenie 3.6.

Spowoduj, aby w oknie dialogowym wywoływanym przez funkcję `prompt()` nie pojawiała się wartość „`Undefined`", lecz pusty łańcuch znaków (rysunek 3.4).

Rysunek 3.4. *Okno wprowadzania danych z pustym łańcuchem znaków jako wartością domyślną*

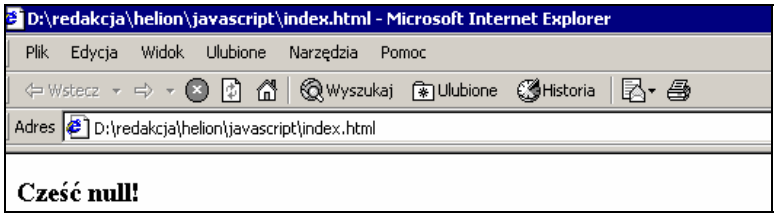



```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
document.write ("Cześć " + prompt ("Podaj swoje imię:", "") + "!");
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Instrukcje warunkowe

Funkcja `prompt()` zwraca wartość podaną przez użytkownika lub jeśli nie podał on żadnej, wartość domyślną, tylko w przypadku naciśnięcia przycisku *OK*. Jeżeli jednak użytkownik wcisnął przycisk *Anuluj* (*Cancel*), zwrócona zostanie wartość `null`. W taki przypadku na ekranie ukaże się napis `Cześć null!`. Widać to na rysunku 3.5. Tego jednak chcielibyśmy uniknąć. Pomogą nam w tym instrukcje warunkowe.

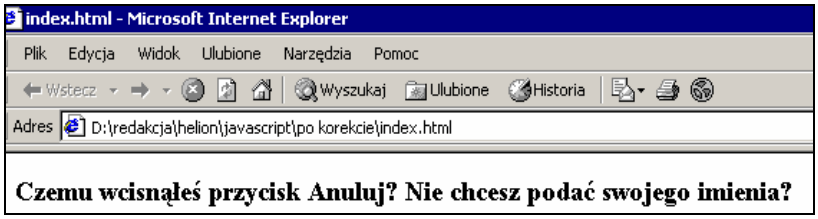
Rysunek 3.5. *Efekt wciśnięcia przycisku Anuluj (Cancel)*



Ćwiczenie 3.7.

Wyświetl na ekranie okno dialogowe pozwalające na podanie przez użytkownika imienia. Wyprowadź na ekran napis powitalny zawierający podane imię. W przypadku gdy użytkownik naciśnie przycisk *Anuluj* (*Cancel*), ma pojawić się stosowny komunikat (rysunek 3.6).

Rysunek 3.6. *Komunikat pojawia się po wciśnięciu przycisku Anuluj (Cancel) po wywołaniu funkcji prompt()*




```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var imie = prompt ("Podaj swoje imię:", "");
if (imie == null) {
    document.write ("<H3>Czemu wcisnąłeś przycisk Anuluj? Nie chcesz podać swojego imienia?");
}
else {
    document.write ("Cześć " + imie + "!");
}
```

```
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Mamy tu kilka nowych elementów wymagających wyjaśnienia. Na początku deklarujemy zmienną `imie` i przypisujemy jej od razu wartość zwróconą przez funkcję `prompt()`. Jest to ciąg znaków wprowadzony przez użytkownika lub wartość `null`. Następnie za pomocą konstrukcji `if...else` sprawdzamy, czy wartość zmiennej `imie` jest równa `null`. Jeśli tak, wykonujemy ciąg instrukcji zapisany pomiędzy nawiasami `{ }` po instrukcji `if`. W przeciwnym przypadku wykonujemy instrukcje z bloku po występującego po instrukcji `else`. Ogólna konstrukcja `if...else` wygląda następująco:

```
if (warunek logiczny) {
    instrukcje do wykonania, jeśli warunek jest prawdziwy
}
else {
    instrukcje do wykonania, jeśli warunek nie jest prawdziwy
}
```

W powyższym ćwiczeniu warunkiem logicznym jest (`imie == null`). Jest to porównanie wartości zmiennej `imie` i wartości `null`. Jeżeli są one równe, warunek jest prawdziwy. Tzn. ma on wartość logiczną `TRUE` i wykonywany jest blok instrukcji występujący po `if`. Jeżeli wartości te nie są równe (np. `imie = „Anna"`), warunek ten ma wartość logiczną `FALSE` i wykonywany jest blok instrukcji występujący po `else`.

W tym miejscu małe wyjaśnienie dla osób, które do uruchomienia ostatniego skryptu użyły jednej ze starszych wersji przeglądarki Internet Explorer (wersje poniżej 4). Otóż skrypt w tym przypadku nie zadziałał tak jak powinien. Wczesne wersje tego programu po wciśnięciu przycisku *Anuluj* (*Cancel*) nie zwracały wartości `null`, ale pusty łańcuch znaków (zapisujemy go jako ""). Oczywiście po kliknięciu przycisku *OK*, gdy nic nie zostało wpisane w okienku dialogowym, również zwracany był pusty ciąg znaków. Mieliśmy więc identyczną reakcję na dwa różne zdarzenia, co uniemożliwiało wykrycie wciśnięcie przycisku *Anuluj* (*Cancel*). Na szczęście ten problem został usunięty i w nowych wersjach przeglądarki już nie występuje.

Operacje na zmiennych

Operacje dokonywane na zmiennych możemy podzielić na:

- operacje arytmetyczne,
- operacje bitowe,
- operacje logiczne,
- operacje przypisania,
- operacje porównania,
- operacje na łańcuchach znaków.

Operacje arytmetyczne

Operacje arytmetyczne to standardowe dodawanie, odejmowanie, mnożenie oraz dzielenie. Zapisujemy je za pomocą znanych z matematyki znaków: +, –, *, /.

Ćwiczenie 3.8.

Zadeklaruj kilka zmiennych, wykonaj na ich standardowe operacje arytmetyczne i wyprowadź wyniki na ekran.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var zmienna1 = 12;
var zmienna2 = 5;
var zmienna3 = zmienna1 + zmienna2
document.write (zmienna3 + " " + (zmienna1 + 4) + " " );
zmienna3 = zmienna1 / 3;
document.write (zmienna3 + " " + zmienna1 * zmienna2);
```

```
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Wynikiem działania tego skryptu, nikt nie powinien chyba mieć żadnych wątpliwości, będzie ciąg liczb: 17 16 4 60. Podkreślenia wymaga jedynie to, że wyrażenie `zmienna1 + 4` musi być ujęte w nawiasy. Dlaczego tak się dzieje, stanie się jasne przy wykonywaniu ćwiczeń dotyczących operacji na łańcuchach znakowych.

Do operatorów arytmetycznych należy również znak %. Nie oznacza on jednak obliczania procentów, ale tzw. dzielenie modulo (resztę z dzielenia). Zakładając dane jak w ćwiczeniu 3.8, wyrażenie `zmienna1 % zmienna2` przyjmie wartość `2` (12%5 = 2). Do dyspozycji mamy również operator zmiany znaku oraz inkrementacji i dekrementacji. Zmiany znaku dokonujemy poprzez dostawienie znaku minusa (–), np. po wykonaniu wyrażenia `zmienna1 = -zmienna1` wartością zmiennej zmienna `1` stanie się `-12` (również przy założeniu danych jak w ćwiczeniu 3.8).

Dużo ciekawszym operatorem jest operator inkrementacji, czyli zwiększenia wartości. Powoduje on przyrost wartości zmiennej o jeden. Operator ten, zapisywany jako „++”, może występować w dwóch formach: przyrostkowej bądź przedrostkowej. Tzn. jeśli mamy zmienną, która nazywa się np. `x`, forma przedrostkowa będzie wyglądać: `++x`, natomiast przyrostkowa `x++`. Oba te wyrażenia zwiększą wartość zmiennej `x` o jeden, jednak wcale nie są sobie równoważne. Otóż operator `x++` zwiększa wartość zmiennej po jej wykorzystaniu, natomiast `++x` przed jej wykorzystaniem. I w brew pozorom takie rozróżnienie może być bardzo pomocne podczas pisania programów.

Następne ćwiczenie, związane z operatorem inkrementacji, będzie nietypowe.

Ćwiczenie 3.9.

Przeanalizuj poniższy kod. Nie wczytuj skryptu do przeglądarki, ale zastanów się, jaki będzie wyświetlony ciąg liczb. Następnie, po uruchomieniu skryptu, sprawdź swoje przypuszczenia.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var x = 12;
var y;
/*1*/document.write (++x);
/*2*/document.write (" ");
/*3*/document.write (x++);
/*4*/document.write (" ");
/*5*/document.write (x);
/*6*/document.write (" ");
/*7*/y = x++;
/*8*/document.write (y);
/*9*/document.write (" ");
/*10*/y = ++x;
/*11*/document.write (y);
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Wynikiem będzie ciąg znaków „13 13 14 14 16”. Zatem, jak to wszystko działa (dla ułatwienia opisu wiersze skryptu zostały ponumerowane)? Otóż w wierszu oznaczonym numerem 1 najpierw zwiększana jest wartość zmiennej `x` o 1 (czyli `x = 13`), a następnie ten wynik jest wyświetlany. W linii numer 3 najpierw jest wyświetlana aktualna wartość zmiennej `x` (czyli 13), a następnie jest ona zwiększana o 1 (czyli `x = 14`). W wierszu nr 5 jest wyświetlana aktualna wartość zmiennej `x`, czyli 14. W wierszu siódmym zmiennej `y` jest przypisywana wartość zmiennej `x`, a następnie zmienna `x` jest zwiększana o 1 (czyli `y = 14`, `x = 15`). W wierszu dziesiątym najpierw jest zwiększana wartość zmiennej `x` o 1 (czyli `x =16`), a następnie wartość ta jest przypisywana zmiennej `y` (czyli `y = 16` i `x = 16`). Na początku może wydawać się to trochę skomplikowane, ale po dokładnym przeanalizowaniu i samodzielnym wykonaniu kilku własnych ćwiczeń operator ten nie powinien sprawiać żadnych kłopotów.

Operator dekrementacji działa analogicznie, z tym że zamiast zwiększać wartości zmiennych, zmniejsza je. Oczywiście zawsze o jeden.

Ćwiczenie 3.10.

Zmień kod z ćwiczenia 3.9 tak, aby operator ++ został zastąpiony operatorem --. Następnie przeanalizuj jego działanie i sprawdź, czy otrzymany wynik jest taki sam jak na ekranie przeglądarki.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var x = 12;
var y;
document.write (--x);
document.write (" ");
document.write (x--);
document.write (" ");
document.write (x);
document.write (" ");
y = x--;
document.write (y);
document.write (" ");
y = --x;
document.write (y);
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Operacje na bitach

Operacje tego typu są wykonywane oczywiście na bitach, w arytmetyce binarnej. Zestawienie tych operacji przedstawione jest w tabeli 3.2.

Tabela 3.2. *Operacje bitowe*

Rodzaj działania	Symbol w JavaScript
bitowe AND	&
bitowe OR	|
XOR	^
przesunięcie bitowe w lewo	<<
przesunięcie bitowe w prawo	>>
przesunięcie bitowe w prawo z wypełnieniem zerami	>>>

Operacje przypisania

Operacje przypisania są dwuargumentowe i powodują przypisanie wartości argumentu prawostronnego do argumentu lewostronnego. Najprostszym operatorem tego typu jest oczywiście klasyczny znak równości. Oprócz niego mamy jeszcze do dyspozycji operatory łączące klasyczne przypisanie z innym operatorem arytmetycznym bądź bitowym.

Tabela 3.3. *Operacje przypisania*

Argument 1	Operator	Argument 2	Znaczenie
X	+=	Y	X = X +Y
X	–=	Y	X = X – Y
X	*=	Y	X = X * Y
X	/=	Y	X = X / Y
X	%=	Y	X = X % Y
X	<<=	Y	X = X << Y
X	>>=	Y	X = X >> Y
X	>>>=	Y	X = >>> Y
X	&=	Y	X = X & Y

X	|=	Y	X = X | Y
X	^=	Y	X = X ^ Y

Operacje logiczne i porównania

Argumentami operacji logicznych tego typu muszą być wyrażenia posiadające wartość logiczną, czyli `TRUE` lub `FALSE` (prawda i fałsz). Np. wyrażenie 10 < 20 jest niewątpliwie prawdziwe (10 jest mniejsze od 20), zatem jego wartość logiczna jest równa `TRUE`.

W grupie tej wyróżniamy trzy operatory: logiczne: `AND` (&&), logiczne `OR` (||) i logiczna negacja — `NOT` (!). `AND` (iloczyn logiczny) działa w taki sposób, że daje wynik `TRUE` tylko wtedy, jeśli oba argumenty mają wartość `TRUE`. Logiczne `OR` daje natomiast wynik `TRUE` wtedy, gdy przynajmniej jeden z argumentów ma wartość `TRUE`. Logiczne `NOT` jest po prostu negacją, tzn. zmienia wartość argumentu na przeciwny:

Operacje porównania porównują dwa argumenty. Wynikiem porównania jest wartość `TRUE` (jeśli jest ono prawdziwe) lub `FALSE` (jeśli jest fałszywe). Argumentami mogą być zarówno wartości numeryczne, jak i łańcuchy znaków. Do dyspozycji mamy operatory porównania przedstawione w tabeli 3.4.

Tabela 3.4. *Operacje porównania*

Operator	Znaczenie
==	Zwraca `TRUE`, jeśli argumenty są sobie równe.
!=	Zwraca `TRUE`, jeśli argumenty są różne.
<	Zwraca `TRUE`, jeśli argument prawostronny jest większy od lewostronnego.
>	Zwraca `TRUE`, jeśli argument prawostronny jest mniejszy od lewostronnego.
>=	Zwraca `TRUE`, jeśli argument prawostronny jest mniejszy lub równy lewostronnemu.
<=	Zwraca `TRUE`, jeśli argument prawostronny jest większy lub równy lewostronnemu.

Ćwiczenie 3.11.

Wyświetl na ekranie okno dialogowe pozwalające na podanie przez użytkownika swojego imienia. Wyprowadź na ekran napis powitalny zawierający podane imię. W przypadku, gdy użytkownik naciśnie przycisk *Anuluj* (*Cancel*) lub nie poda imienia i wciśnie przycisk *OK*, ma się pojawić stosowny komunikat.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var imie = prompt ("Podaj swoje imię:", "");
if ((imie == null) || (imie == "")){
    document.write ("Dlaczego nie podałeś swojego imienia?");
}
else{ document.write ("Cześć " + imie + "!");
    }
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Znaczenie konstrukcji `if ((imie == null) || (imie == ""))` jest następujące: jeżeli zawartość zmiennej `imie` równa jest wartości `null` lub wartość zmiennej `imie` równa jest pustemu łańcuchowi znaków, całe wyrażenie przyjmuje wartość `TRUE`, zatem jest wykonywany blok instrukcji po `if`. Jeżeli jednak użytkownik podał jakąś i wartość zmiennej `imie` nie jest równa ani `null`, ani pustemu ciągowi znaków, wykonujemy blok instrukcji występujący po `else`. W ten sposób jednocześnie wykorzystaliśmy instrukcje porównania i instrukcje logiczne.

Operacje na łańcuchach znaków

Występuje tu tylko jeden operator, mianowicie + (plus). Powoduje on znaną już nam z wcześniejszych ćwiczeń, konkatenację, czyli łączenie łańcuchów znakowych. Np. wynikiem działania operacji:

```
var napis = "Idziemy do " + "kina"
```

będzie oczywiście przypisanie zmiennej napis ciągu znaków „Idziemy do kina". Co jednak ciekawe, a tej właściwości zwykle nie posiadają zwykle inne języki programowania, można również do siebie dodawać zmienne tekstowe oraz liczbowe np. po wykonaniu

```
var x = "Mam " + 100 + " złotych"
```

zmienna x będzie zawierała ciąg znaków „Mam 100 złotych".

Ćwiczenie 3.12.

Dokonaj konkatenacji łańcucha znakowego oraz dowolnej liczby. Wynik wyświetl na ekranie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
x = "Mam " + 100 + " złotych"
document.write (x);
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Ponownie wróćmy teraz do ćwiczenia 3.8. Pojawił się w nim m.in. następujący fragment kodu:

```
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var zmienna1 = 12;
var zmienna2 = 5;
var zmienna3 = zmienna1 + zmienna2
document.write (zmienna3 + " " + (zmienna1 + 4) + " " );
zmienna3 = zmienna1 / 3;
document.write (zmienna3 + " " + zmienna1 * zmienna2);
// Koniec kodu JavaScript -->
</SCRIPT>
```

Pisałem wtedy, iż wyrażenie zmienna 1 + 4 musi być ujęte w nawiasy. Dzieje się tak dlatego, że w przeciwnym wypadku zmienna1 najpierw zostanie przekształcona na łańcuch znaków, a następnie do tego łańcucha zostanie dodana czwórka. Dałoby na ekranie wynik 124 (12 i 4) zamiast spodziewanych 16. Można tego uniknąć poprzez zrezygnowanie z tworzenia jednego łańcucha znakowego jako argumentu funkcji write(), zastępując go listą argumentów.

Ćwiczenie 3.13.

Używając listy argumentów funkcji write() przekształcić kod z ćwiczenia 3.8, tak by nie następowało niepożądane łączenie łańcuchów znakowych.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var zmienna1 = 12;
var zmienna2 = 5;
var zmienna3 = zmienna1 + zmienna2
document.write (zmienna3, " ", zmienna1 + 4, " " );
zmienna3 = zmienna1 / 3;
document.write (zmienna3 + " " + zmienna1 * zmienna2);
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Instrukcja przetwarzania warunkowego

Instrukcja przetwarzania warunkowego pozwala, w niektórych przypadkach, na wygodne zastąpienie bloku if...else. Konstrukcja wygląda następująco:

```
(wyrażenie warunkowe)?wartość1:wartość2
```

Należy rozumieć to w sposób następujący: jeżeli warunek jest spełniony, tzn. wyrażenie warunkowe ma wartość `TRUE` — całość przyjmuje wartość `wartość1`, w przeciwnym przypadku `wartość2`. Najłatwiej zrozumieć ten zapis, wykonując kolejne ćwiczenie.

Ćwiczenie 3.14.

Wyświetl okno dialogowe umożliwiające podanie przez użytkownika dowolnej liczby. Korzystając z instrukcji przetwarzania warunkowego, sprawdź, czy liczba jest ujemna czy nieujemna. Wynik wyświetl na ekranie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var x = prompt ("Podaj liczbę:", "");
var jaka = (x < 0)? "ujemna":"nieujemna";
document.write ("Ta liczba jest " + jaka);
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

W powyższym przykładzie najpierw prosimy użytkownika o podanie dowolnej liczby. Następnie stosujemy poznane właśnie wyrażenie warunkowe, które, o ile x < 0, przypisuje zmiennej `jaka` ciąg znaków `ujemna`, a w przeciwnym przypadku ciąg znaków `nieujemna`.

Ćwiczenie 3.15.

Zmień kod z ćwiczenia 3.14 w taki sposób, aby zamiast wyrażenia warunkowego użyć bloku instrukcji `if...else`.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var x = prompt ("Podaj liczbę:", "");
var jaka;
if (x < 0) jaka = "ujemna";
else jaka = "nieujemna";
document.write ("Ta liczba jest " + jaka);
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Jeśli ktoś lubi nieco bardziej skomplikowany kod, może rozwiązać zadania z powyższych dwóch ćwiczeń bez użycia jakichkolwiek zmiennych. Całość zapisać w jednego tylko wiersza kodu (nie licząc oczywiście znaczników HTML), właśnie dzięki użyciu wyrażenia warunkowego.

Ćwiczenie 3.16.

Wyświetl okno dialogowe umożliwiające podanie przez użytkownika dowolnej liczby. Nie używając zmiennych, ani bloku `if...else` sprawdź, czy liczba jest ujemna czy nieujemna. Wynik wyświetl na ekranie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
document.write ("Ta liczba jest " + (((prompt ("Podaj liczbę","",""))<0)?"ujemna":"nieujemna"));
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

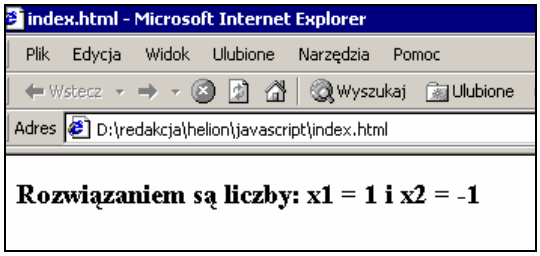
Powyższy skrypt w pierwszej chwili może wydawać się jest jasny. Wystarczy go jednak spokojnie przeanalizować. Wszystkie elementy potrzebne do zrozumienia zostały już omówione, a funkcjonalnie odpowiada on programom z dwóm poprzednich ćwiczeń. Oczywiście z takim upraszczaniem zapisu nie należy przedobrzyć. Może się bowiem okazać się, że po kilku dniach sami nie będziemy mogli go zrozumieć. Niemniej w tym przypadku, stopień skomplikowania nie jest duży, natomiast zdecydowanie oszczędzamy na ilości zmiennych.

Wykonajmy teraz skrypt obliczający pierwiastki równania kwadratowego o zadanych przez użytkownika parametrach. Jak pamiętamy ze szkoły deltę równania w postaci `A*x^2+B*x+C = 0` otrzymujemy ze wzoru: `B^2-4*A*C`. Jeżeli delta jest większa od zera, mamy dwa pierwiastki: `x1 = (-B+√delta)/2*A` i `x2 = (-B-√delta)/2*A`. Jeżeli delta jest równa zero, istnieje tylko jedno rozwiązanie, mianowicie `x = -B/2*A`. W przypadku trzecim, delta mniejsza od zera, w zbiorze liczb rzeczywistych rozwiązań nie ma.

Ćwiczenie 3.17.

Napisz skrypt obliczający pierwiastki równania kwadratowego o parametrach zadanych przez użytkownika. Wyniki wyświetl na ekranie (rysunek 3.7).

Rysunek 3.7.
Efekt działania skryptu rozwiązującego równania kwadratowe



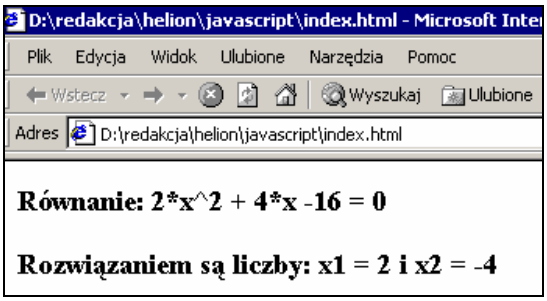

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptów
var A = prompt ("Podaj parametr A równania kwadratowego: Ax^2 + Bx + C = 0","");
var B = prompt ("Podaj parametr B równania kwadratowego: Ax^2 + Bx + C = 0","");
var C = prompt ("Podaj parametr C równania kwadratowego: Ax^2 + Bx + C = 0","");
var delta = B * B - 4 * A * C;
if (delta < 0){
    document.write ("To równanie nie ma rozwiązań w zbiorze liczb rzeczywistych!");
}
else{
    if (delta == 0){
        document.write ("Rozwiązaniem jest liczba: x = ", - B / 2 * A)
    }
    else{
        document.write ("<H3>Rozwiązaniem są liczby: x1 = ", ((- B + Math.sqrt (delta)) / (2 * A)));
        document.write (" i x2 = ", ((- B - Math.sqrt (delta)) / (2 * A)));
    }
}
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Powyższy skrypt działa prawidłowo, warto może jednak pokusić się o wyświetlenie, oprócz wyniku, także samego równania, którego parametry wprowadził użytkownik. Nie jest to wcale banalne, gdyż jeśli chcemy, aby całość wyglądała porządnie, trzeba będzie zastosować kilka instrukcji warunkowych.

Ćwiczenie 3.18.

Napisz skrypt obliczający pierwiastki równania kwadratowego o parametrach zadanych przez użytkownika. Wyświetl na ekranie równanie oraz jego rozwiązania (rysunek 3.8).

Rysunek 3.8.
Skrypt wyświetlający równanie kwadratowe o zdanych parametrach oraz jego rozwiązania




```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptów
var A = prompt ("Podaj parametr A równania kwadratowego: Ax^2 + Bx + C = 0","");
var B = prompt ("Podaj parametr B równania kwadratowego: Ax^2 + Bx + C = 0","");
var C = prompt ("Podaj parametr C równania kwadratowego: Ax^2 + Bx + C = 0","");
var delta = B * B - 4 * A * C;

var rownanie = "";
if (A != 0){
    rownanie = (A == 1)?"x^2 ":A + "*x^2 ";
}
if (B != 0) {
    if (A != 0)
        rownanie += (B == 1)?"+ x ":((B < 0)?B + "*x ":"+ " + B + "*x ");
    else
        rownanie += (B == 1)?"x ":((B < 0)?B + "*x ":B + "*x ");
}
if (C != 0){
    if ((A == 0) && (B == 0))
        rownanie += C;
    else
        rownanie += (C < 0)?C:"+ " + C;
}
rownanie += " = 0";

document.write("Równanie: " + rownanie +"<BR><BR>");

if (delta < 0){
    document.write ("To równanie nie ma rozwiązań w zbiorze liczb rzeczywistych!");
}
else{
    if (delta == 0){
        document.write ("Rozwiązaniem jest liczba: x = ", - B / 2 * A)
    }
    else{
        document.write ("Rozwiązaniem są liczby: x1 = ", ((- B + Math.sqrt (delta)) / (2 * A)));
        document.write (" i x2 = ", ((- B - Math.sqrt (delta)) / (2 * A)));
    }
}
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Jak widać, skrypt jest teraz bardziej skomplikowany. Wyjaśnienia może wymagać wiersz:

```
rownanie += (B == 1)?"+ x ":((B < 0)?B + "*x ":"+ " + B + "*x ");
```

Pierwsze wrażenie — bardzo skomplikowanie, niemniej jest to jedynie zagnieżdżone wyrażenie warunkowe poznane w ćwiczeniu 3.14. Ogólna konstrukcja takiego wyrażenia wyglądała następująco:

```
(wyrażenie warunkowe)?wartość1:wartość2
```

W naszym przypadku `wartość2` zastąpiona została po prostu przez kolejne wyrażenie warunkowe. Schematycznie można przedstawić jako:

```
(wyrażenie warunkowe 1)?wartość1:((wyrażenie warunkowe 2)?wartość3:wartość4)
```

Mimo że skrypt jest teraz dosyć skomplikowany, nie oznacza to, że nie można go jeszcze usprawnić. Przede wszystkim należałoby sprawdzać, czy na pewno równanie podane przez użytkownika jest równaniem kwadratowym, tzn. czy parametr `A` nie jest równy zero. Jeśli tak, nie należy obliczać delty. Warto też zainteresować się, czy użytkownik nie wcisnął przy po-

dawaniu parametrów przycisku *Anuluj* (*Cancel*). Kod wyświetlający samo równanie też można usprawnić. W tej chwili, np. jeśli parametr `B` równy jest jeden, nie wyświetlamy jedynki, a tylko „`+B`”, jeśli jednak `B` jest równe `-1`, wyświetlane jest `-1 * B`. Te poprawki pozostawmy jednak jako zadanie do samodzielnego wykonania.

Pętle

Pętle to wyrażenia programowe służące do wykonywania powtarzających się czynności. Np. jeśli chcielibyśmy 150 razy wypisać na stronie `To jest moja strona domowa`, pomijając sensowność takiej czynności, wklepanie w skrypcie 150 razy `document.write („To jest moja strona domowa”)` byłoby niewątpliwie uciążliwe. Pętle pozwalają na automatyzację takiej czynności.

Pętla for

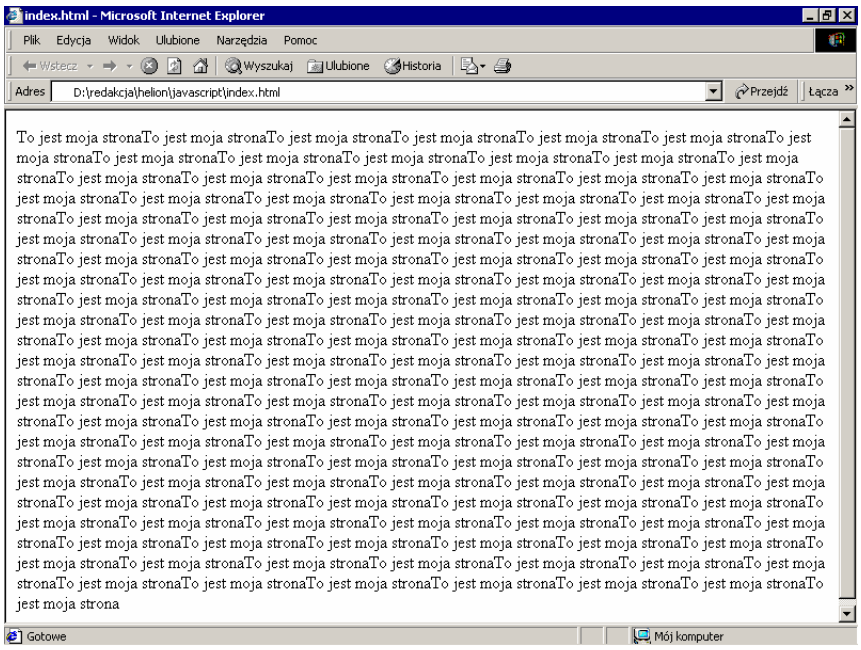
Pętla typu `for` ma składnię następującą:

```
for (wyrażenie początkowe; wyrażenie warunkowe; wyrażenie modyfikujące){
    blok instrukcji
}
```

Wyrażenie początkowe jest stosowane do zainicjalizowania zmiennej używanej jako licznik ilości wykonać pętli. *Wyrażenie warunkowe* określa warunek, jaki musi być spełniony, aby dokonać kolejnego przejścia w pętli, *wyrażenie modyfikujące* używane jest zwykle do modyfikacji zmiennej będącej licznikiem.

Ćwiczenie 3.19.

Użyj pętli typu `for` do wyświetlenia na ekranie dowolnej sekwencji napisów (rysunek 3.9).



Rysunek 3.9. *Zastosowanie pętli typu for do wyświetlania sekwencji napisów*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
for (var i = 1; i <= 150;i++){
  document.write ("To jest moja strona");
}
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Kod należy rozumieć następująco: dopóki wartość zmiennej `i` jest mniejsza bądź równa `150`, zwiększaj `i` o `1` oraz wykonuj instrukcję `document.write („To jest moja strona")`. Efekt działania widoczny jest na rysunku 3.9.

Pętle tego typu można zmodyfikować, tak aby pozbyć się wyrażenia modyfikującego. Dokładniej przenieść je do wnętrza pętli w sposób następujący:

```
for (wyrażenie początkowe; wyrażenie warunkowe;){ blok instrukcji
     wyrażenie modyfikujące
}
```

Ćwiczenie 3.20.

Zmodyfikuj pętle typu `for` tak, aby wyrażenie modyfikujące znalazło się w bloku instrukcji.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
for (var i = 1; i <= 150;){
  document.write ("To jest moja strona");
  i++;
}
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Ważną rzeczą jest, aby pamiętać o średniku występującym po wyrażeniu `i <= 150` — jest on bowiem niezbędny dla prawidłowego funkcjonowania całego skryptu.

W podobny sposób można też „pozbyć się" wyrażenia początkowego, przenosząc je jednak nie do wnętrza pętli, a przed nią.

Ćwiczenie 3.21.

Usuń wyrażenie początkowe poza pętlę `for`.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var i = 1;
for (;i <= 150;){
  document.write ("To jest moja strona");
  i++;
}
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Skoro zaszliśmy już tak daleko w pozbywaniu się wyrażeń sterujących, usuńmy również wyrażenie warunkowe. Jest to jak najbardziej możliwe.

Ćwiczenie 3.22.

Umieść wyrażenia: warunkowe i modyfikujące we wnętrzu pętli, natomiast wyrażenie początkowe przenieś poza pętlę.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var i = 1;
for (;;){
  document.write ("To jest moja strona");
  if (i++ >= 150) break;
}
```

```
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

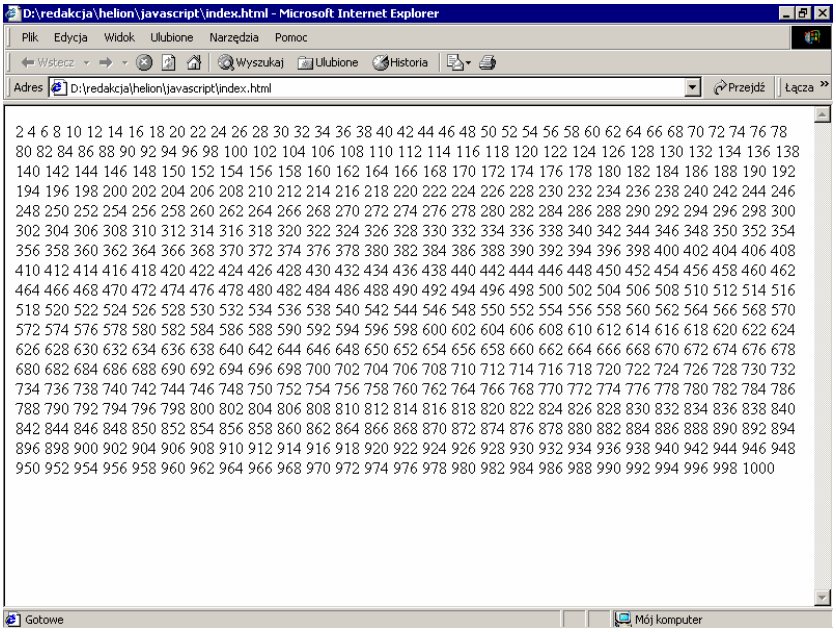
Należy zwrócić tutaj uwagę na dwie sprawy. Po raz kolejny zauważ średniki w wyrażeniu `for` — tym razem dwa, oba są niezbędne, inaczej skrypt nie zadziała. Ważne jest również odwrócenie kierunku nierówności. W ćwiczeniu 3.20 było bowiem `i++ <=150`, obecnie jest `i++ >= 150`. W pierwszym przypadku sprawdzaliśmy warunek, czy pętla ma być dalej wykonywana, w drugim sprawdzamy, czy pętla ma się zakończyć. Na to zakończenie pozwala nowa instrukcja `break`. Po jej napotkaniu następuje przerwanie wykonywanie pętli, niezależnie od stanu zmiennej sterującej (w naszym przypadku `i`).

Drugą instrukcją pozwalającą na modyfikację zachowania pętli jest `continue`. Po jej napotkaniu następuje przerwanie bieżącej iteracji i rozpoczęcie kolejnej. Mówiąc prościej następuje przeskok na początek pętli.

Ćwiczenie 3.23.

Wyświetl na ekranie wszystkie liczby pomiędzy 1 a 1 000 podzielne przez 2 (rysunek 3.10). Skorzystaj z pętli `for` i instrukcji `continue`.

Rysunek 3.10. *Efekt działania skryptu wypisującego liczby podzielne przez dwa*




```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">

<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
for (var i = 1; i <= 1000; i++){
  if ((i % 2) != 0)
    continue;
  document.write (i + " ");
}
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Oczywiście do realizacji zadania z ćwiczenia 3.22 nie jest konieczna instrukcji `continue`. Wystarczy skorzystać tylko z instrukcji `if`. Sama pętla wyglądałaby wtedy następująco:

```
for (var i = 1; i <= 1000; i++){
  if ((i % 2) == 0)
    document.write (i + " ");
}
```

Pętla while

Pętla typu `while` ma składnię następującą:

```
while (wyrażenie warunkowe){
  blok instrukcji
}
```

Wyrażenie warunkowe musi być spełnione, żeby wykonana została kolejna iteracja.

Ćwiczenie 3.24.

Użyj pętli typu `while` do wyświetlenia na ekranie dowolnej sekwencji napisów (tak jak na rysunku 3.9).

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
i = 0;
while(i++ < 150){
  document.write ("To jest moja strona");
}
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

W pętli `while` można oczywiście również stosować instrukcje `break` i `continue`.

Rozdział 4.

Obiekty i funkcje

Funkcje

Funkcje można określić jako zgrupowane ciągi instrukcji służące do wykonywania określonych, powtarzających się zadań. Funkcje powiązane z obiektami i na nich operujące nazywamy natomiast metodami, o czym jednak nieco dalej. Do funkcji możemy przekazać pewne wartości, może ona również pewną wartość zwracać. Funkcją, już poznaną jest np. `prompt()`, którą w poprzednich przykładach wywoływaliśmy m.in. następująco:

```
var A = prompt ("Podaj imię","");
```

Parametrami są tu dwa ciągi znaków. Pierwszy to napis, który ukaże się w oknie, drugi jest wartością domyślną ukazującą się w oknie edycyjnym. Zwraca ona ciąg znaków, który podamy po otrzymaniu zapytania. Definicja funkcji musi zatem zawierać:

- słowo kluczowe `function`,
- nazwę funkcji,
- listę argumentów,
- implementację.

Lista argumentów musi być zawarta między nawiasami okrągłymi `(` i `)`, poszczególne argumenty oddzielane są przecinkami. Implementacja, czyli ciąg instrukcji, które funkcja będzie wykonywać, zawarta jest pomiędzy nawisami klamrowymi `{` i `}`. Ogólnie definicja wygląda następująco:

```
function nazwa_funkcji (agrument1, argument2, ... argument n){
    instrukcje JavaScript
}
```

Ćwiczenie 4.1.

Napisz kod funkcji wyświetlającej na ekranie napis przekazany jako parametr.

```
function wyswietl_powitanie (imie){
  document.write ("Cześć " + imie);
}
```

Ćwiczenie 4.2.

Umieść definicję funkcji z ćwiczenia 4.1 w nagłówku kodu HTML. Wywołaj funkcję w sekcji `<BODY>`.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function wyswietl_powitanie (imie){
  document.write ("Cześć " + imie);
}
// Koniec kodu JavaScript -->
```

```
</SCRIPT>
</HEAD>
<BODY>
<H3><CENTER>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
wyswietl_powitanie ("Marcin");
// Koniec kodu JavaScript -->
</SCRIPT>
</H3></CENTER>
</BODY>
</HTML>
```

Oczywiście raz stworzoną funkcję możemy wywoływać dowolną ilość razy. Nic nie stoi też na przeszkodzie, aby dodać element interaktywności, stosując wspomnianą wcześniej instrukcję `prompt()`. Wystarczy zmienić linię:

```
wyswietl_powitanie ("Marcin")
```

na:

```
wyswietl_powitanie (prompt ("Podaj swoje imie: ",""));
```

Skrypt poprosi wtedy użytkownika o podanie imienia, a następnie wyświetli je na ekranie.

Jak widać, na liście parametrów funkcji może się znaleźć wywołanie innej funkcji, pod warunkiem, że zwróci ona jakiś wynik. W jaki sposób zmusić funkcję do zwrotu wyniku? Należy użyć instrukcji `return`.

Ćwiczenie 4.3.

Napisz funkcję obliczającą drugą potęgę liczby podanej jako argument. Wywołaj ją w skrypcie umożliwiającym użytkownikowi podanie dowolnej liczby. Wynik działania wyświetl na ekranie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function druga_potega (x){
  x = x * x;
  return x;
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H3><CENTER>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var x = prompt ("Podaj liczbę", "");
document.write (x + " do kwadratu = " + druga_potega (x));
// Koniec kodu JavaScript -->
</SCRIPT>
</H3></CENTER>
</BODY>
</HTML>
```

Rekurencja

Rekurencja jest to wywoływanie funkcji przez samą siebie. Choć brzmi to może dziwnie, taka możliwość ułatwia rozwiązywanie niektórych problemów. Prostym przykładem może być zmodyfikowanie funkcji `druga_potega()` w taki sposób, aby można było podnosić liczby do dowolnej całkowitej nieujemnej potęgi. Na początku jednak napiszmy nierekurencyjną wersję takiej funkcji.

Ćwiczenie 4.4.

Napisz funkcję podnoszącą liczbę do zadanej potęgi. Wywołaj funkcję w przykładowym skrypcie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
```

```
function potega (liczba, do_potegi){
  if (do_potegi == 0)
          return 1;
  var temp = liczba;
  for (i = 0; i < do_potegi - 1; i++){
    liczba = liczba * temp;
        }
        return liczba;
}

// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H3><CENTER>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var x = potega(2, 5);
document.write ("dwa do potęgi piątej = " + x);
// Koniec kodu JavaScript -->
</SCRIPT>
</H3></CENTER>
</BODY>
</HTML>
```

Jak zatem będzie wyglądać rekurencyjna wersja funkcji `potega()`?

Ćwiczenie 4.5.

Napisz rekurencyjną wersję funkcji podnoszącej liczbę do zadanej potęgi. Wywołaj funkcję w przykładowym skrypcie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function potega (liczba, do_potegi){
  if (do_potegi == 0)
    return 1;
  else
    return (liczba * potega (liczba, do_potegi - 1));
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H3><CENTER>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var x = potega(2, 5);
document.write ("dwa do potęgi piątej = " + x);
// Koniec kodu JavaScript -->
</SCRIPT>
</H3></CENTER>
</BODY>
</HTML>
```

Zapis ten wydaje się być prostszy i bardziej czytelny niż ten z poprzedniego ćwiczenia. Funkcja wywołuje samą siebie, za każdym razem zmniejszając wartość `do_potegi` o jeden. Trwa to tak długo, aż `do_potegi` stanie się zerem. Dalej rozpoczynają się powroty z funkcji. Po pierwszym powrocie jest to 1, po drugim 1 razy liczba, po trzecim 1 razy liczba razy liczba itd. Ostatecznie otrzymujemy liczbę podniesioną do zadanej potęgi.

Postarajmy się wykonać jeszcze jeden przykład. Niech będzie to rekurencyjna wersja funkcji silnia. Już sama definicja silni jest rekurencyjna. Przypomnijmy: 0! = 1, 1! = 1, n! = n* (n – 1)!. Czyli np. 4! = 1*2*3*4 = 24.

Ćwiczenie 4.6.

Stwórz skrypt obliczający silnię liczby podanej przez użytkownika. Posłuż się rekurencyjną wersją funkcji silnia.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function silnia (liczba){
    if (liczba == 0){
        return 1;
    }
```

```
    else{
        return (liczba * silnia (liczba - 1));
    }
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H3><CENTER>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var liczba = prompt ("Podaj liczbę", 0);
var wynik = silnia (liczba);
document.write ("Wynik: " + liczba + "! = " + wynik);
// Koniec kodu JavaScript -->
</SCRIPT>
</H3></CENTER>
</BODY>
</HTML>
```

Używając funkcji rekurencyjnych należy pamiętać, aby zawsze umieścić warunek wyjścia z rekurencji. Inaczej funkcja taka będzie wywoływać sama siebie bez końca. W przypadku klasycznych języków programowania spowoduje to zawsze przepełnienie stosu programu i tym samym jego „awarie". Co się stanie w przypadku gdy JavaScript zależy od implementacji mechanizmów bezpieczeństwa w przeglądarkach?

Ćwiczenie 4.7.

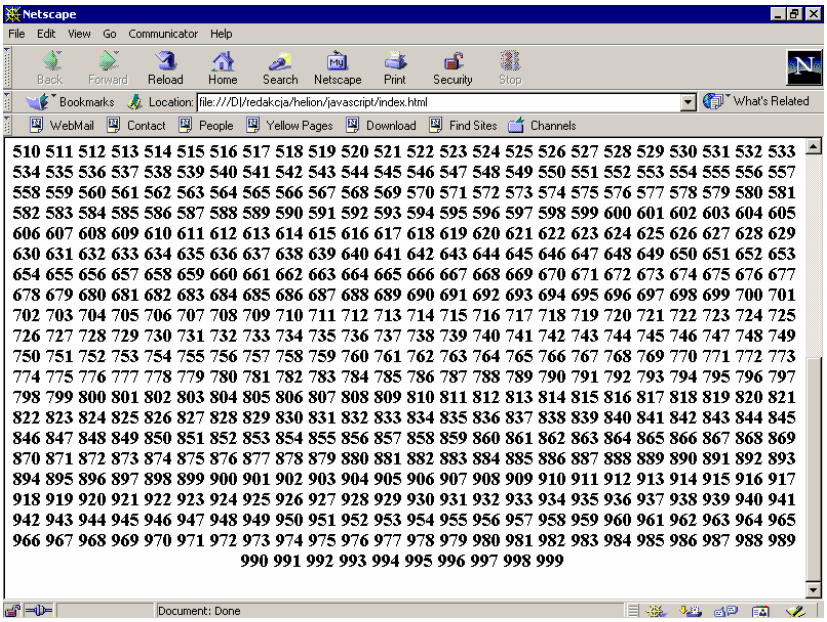
Sprawdzić efekt wykonania „nieskończonej" funkcji rekurencyjnej w różnych przeglądarkach.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScript
function a (i){
  document.write (i++ + " ");
  a (i);
}

// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H3><CENTER>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
a (1);
// Koniec kodu JavaScript -->
</SCRIPT>
</H3></CENTER>
</BODY>
</HTML>
```

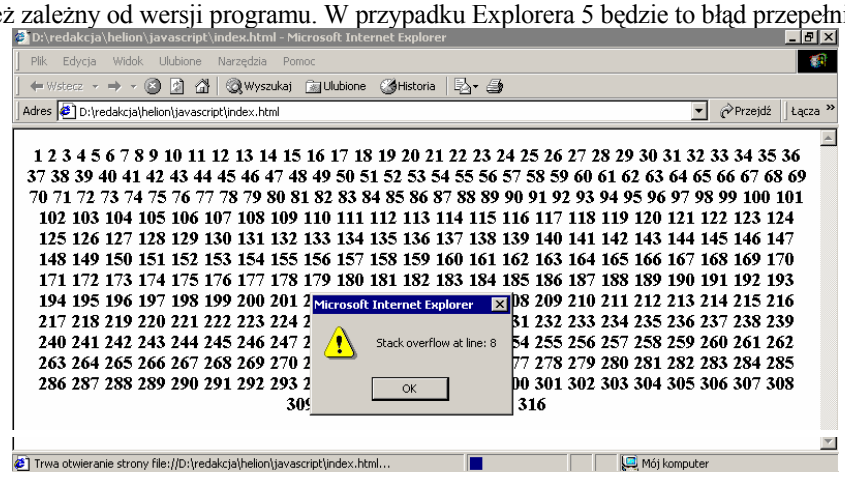
Liczba możliwych do wykonania wywołań rekurencyjnych zależy od użytej przeglądarki. Konkretnie od implementacji zawartego w niej interpretera języków skryptowych. Dzięki powyższemu ćwiczeniu można przekonać się, że Netscape Navigator akceptuje do 999 wywołań rekurencyjnych (rysunek 4.1). Po osiągnięciu tej liczby wykonywanie skryptu jest przerywane, przy czym użytkownik nie jest o tym fakcie informowany.

Rysunek 4.1.
Wynik działania „nieskończonej” funkcji rekurencyjnej w przeglądarce Netscape Navigator



Internet Explorer zachowuje się tu zupełnie inaczej. Liczba możliwych wywołań jest różna i zależy od wersji przeglądarki oraz systemu operacyjnego na jakim jest uruchomiona. Po przekroczeniu granicznej liczby wywołań generowany jest błąd — również zależny od wersji programu. W przypadku Explorera 5 będzie to błąd przepełnienia stosu (rysunek 4.2).

Rysunek 4.2.
Wynik działania „nieskończonej” funkcji rekurencyjnej w przeglądarce Internet Explorer



Obiekty

Obiekty są to konstrukcje programistyczne posiadające tzw. właściwości, którymi mogą być zmienne JavaScript lub też inne obiekty. Z obiektami powiązane są funkcje wykonujące operacje na właściwościach obiektu, które nazywamy metodami. Do właściwości danego obiektu możemy się dostać, używając następującej notacji:

```
nazwa_obiektu.nazwa_właściwości
```

Np. jeśli mamy obiekt „komputer”, może on mieć właściwości: procesor, zegar, cena. Aby więc „wypełnić” obiekt odpowiednimi wartościami, będziemy musieli wykonać następującą sekwencję instrukcji:

```
komputer.procesor = "Celeron"
komputer.zegar = 800
komputer.cena = "2500 zł"
```

Identyczny efekt uzyskamy również przy zastosowaniu innej notacji:

```
nazwa_obiektu[nazwa_właściwości], np. komputer[procesor]
```

Pozostaje pytanie, jak taki obiekt stworzyć? Najpierw musimy go zdefiniować, tworząc funkcję zwaną konstruktorem, a następnie utworzyć za pomocą operatora `new`.

Ćwiczenie 4.8.

Napisz funkcję konstruktora obiektu o nazwie „komputer”. Utwórz obiekt za pomocą operatora `new`, wyświetl jego właściwości na ekranie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function komputer (procesor, zegar, cena){
  this.procesor = procesor;
  this.zegar = zegar;
  this.cena = cena;
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H3><CENTER>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
moj_pc = new komputer ("Celeron", 800, "2500 zł");
document.write("Procesor: " + moj_pc.procesor);
document.write("<BR>Zegar: " + moj_pc.zegar);
document.write("<BR>Cena: " + moj_pc.cena);
// Koniec kodu JavaScript -->
</SCRIPT>
</H3></CENTER>
</BODY>
</HTML>
```

Częściami składowymi obiektu nie muszą być tylko „zwykłe” zmienne. Mogą nimi również być inne obiekty. Jeśli chcielibyśmy do obiektu `komputer` dodać informację o jego właścicielu, warto stworzyć nowy obiekt — `właściciel` — i potraktować go jako właściwość komputera. Brzmi to może nieco śmiesznie, wykonajmy jednak takie ćwiczenie.

Ćwiczenie 4.9.

Napisz konstruktor i utwórz obiekt `osoba`. Wykorzystaj go jako właściwość obiektu `komputer` z ćwiczenia 4.8.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function komputer (procesor, zegar, cena, wlasciciel){
  this.procesor = procesor;
  this.zegar = zegar;
  this.cena = cena;
  this.wlasciciel = wlasciciel;
}
function osoba(imie, nazwisko){
  this.imie = imie;
  this.nazwisko = nazwisko;
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H3><CENTER>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
ja = new osoba("Marek", "Kowalski");
moj_pc = new komputer ("Celeron", 800, "2500 zł", ja);
document.write("Procesor: " + moj_pc.procesor);
document.write("<BR>Zegar: " + moj_pc.zegar);
document.write("<BR>Cena: " + moj_pc.cena);
document.write("<BR>Imię właściciela: " + moj_pc.wlasciciel.imie);
document.write("<BR>Nazwisko właściciela: " + moj_pc.wlasciciel.nazwisko);
// Koniec kodu JavaScript -->
</SCRIPT>
</H3></CENTER>
</BODY>
</HTML>
```

Taka realizacja ma jednak pewną wadę. Otóż wyświetlanie odbywa się tutaj niezależnie od obiektów. Dużo lepszym pomysłem jest napisanie dla każdego obiektu funkcji wyświetlającej jego właściwości i zadeklarowanie tych funkcji jako metod danego obiektu. Chcąc wyświetlić taki obiekt na ekranie będziemy po prostu pisać:

```
nazwa_obiektu.wyswietl;
```

Ćwiczenie 4.10.

Stwórz metody wyświetlające właściwości obiektów `komputer` i `osoba`.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function wyswietl_komputer(){
    document.write("<BR>Procesor: " + this.procesor);
    document.write("<BR>Zegar: " + this.zegar);
    document.write("<BR>Cena: " + this.cena);
    this.wlasciciel.wyswietl();
}
function wyswietl_osoba(){
    document.write("<BR>Imię właściciela: " + this.imie);
    document.write("<BR>Nazwisko właściciela: " + this.nazwisko);
}
function komputer (procesor, zegar, cena, wlasciciel){
    this.procesor = procesor;
    this.zegar = zegar;
    this.cena = cena;
    this.wyswietl = wyswietl_komputer;
    this.wlasciciel = wlasciciel;
}
function osoba(imie, nazwisko){
    this.imie = imie;
    this.nazwisko = nazwisko;
    this.wyswietl = wyswietl_osoba;
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H3><CENTER>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
ja = new osoba("Marek", "Kowalski");
moj_pc = new komputer ("Celeron", 800, "2500 zł", ja);
moj_pc.wyswietl();
// Koniec kodu JavaScript -->
</SCRIPT>
</H3></CENTER>
</BODY>
</HTML>
```

Łańcuchy znaków (obiekt string)

Obiekt `string` służy do przechowywania łańcuchów znakowych. Wykorzystywaliśmy go już wielokrotnie, tworząc zmienne będące łańcuchami znaków. Ma on tylko jedną właściwość, o nazwie `length`, która jest liczbą całkowitą reprezentującą ilość znaków w łańcuchu. Niezbyt rozbudowana ilość właściwości jest rekompensowana przez pokaźną ilość metod możliwych do zastosowania w przypadku tego obiektu. Większość z nich służy do dodawania do łańcuchów znakowych etykiet HTML w celu wyświetlenia ich na stronie WWW.

Metody dodające znaczniki oraz odpowiadające im kody HTML zawarte są w tabeli 4.1.

Tabela 4.1. *Lista metod dodających znaczniki HTML operujących na obiekcie string.*
Uwaga: Przykłady zakładają istnienie obiektu o nazwie „napis" zawierającego ciąg znaków „przykładowy tekst"

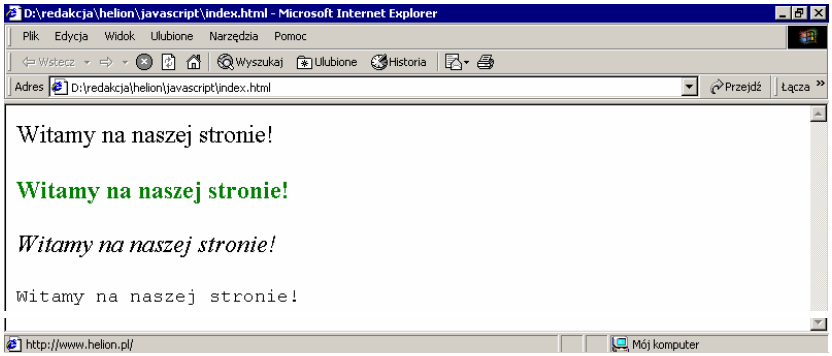
Nazwa metody	Znacznik HTML	Przykład wywołania	Odpowiadający kod HTML
anchor	<A>	napis.anchor ("odnosnik_1")	<ANAME = "odnosnik_1"> przykładowy tekst </A>
big	<BIG>	napis.big()	<BIG> przykładowy tekst </BIG>
blink	<BLINK>	napis.blink()	<BLINK> przykładowy tekst </BLINK>
bold	<BOLD>	napis.bold()	<BOLD> przykładowy tekst </BOLD>
fixed	<TT>	napis.fixed()	<TT> przykładowy tekst </TT>

fontcolor	<FONT>	napis.fontcolor("red")	<FONT COLOR = "red"> przykładowy tekst </FONT>
fontsize	<FONT>	napis.fontsize("wielkość")	<FONT SIZE = "wielkość"> przykładowy tekst </FONT>
italics	<I>	napis.italics()	<I> przykładowy tekst </I>
link	<A>	napis.link ("http://www.helion.pl")	<AHREF = "http://www.helion.pl"> przykładowy tekst </A>
small	<SMALL>	napis.small()	<SMALL> przykładowy tekst </SMALL>
strike	<STRIKE>	napis.strike()	<STRIKE> przykładowy tekst </STRIKE>
sub	<SUB>	napis.sub()	_{przykładowy tekst}
Sup	<SUP>	napis.sup()	^{przykładowy tekst}

Ćwiczenie 4.11.

Wykorzystaj metody operujące na obiekcie `string` do zmiany wyglądu tekstu wyświetlanego na ekranie przeglądarki (rysunek 4.3).

Rysunek 4.3. *Zmiana wyglądu tekstu za pomocą metod operujących na obiekcie string*




```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<HEAD>
<BODY>
<FONT SIZE = "5">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptów
var napis = "Witamy na naszej stronie!<BR><BR>"
document.write (napis);
document.write (napis.bold().fontcolor("green"));
document.write (napis.italics());
document.write (napis.fixed());
document.write (napis.bold().italics().fixed().link("http://www.helion.pl"));
document.write (napis.fontcolor("yellow"));
// Koniec kodu JavaScript -->
</SCRIPT>
</BODY>
</HTML>
```

Powyższy przykład pokazuje, że można zastosować wiele metod w jednym ciągu. Jest to możliwe, bowiem wywołanie każdej metody zwraca obiekt typu `string`, czyli nasz łańcuch znaków, uzupełniony o dodane znaczniki. Skoro zwrócony został obiekt, możemy do niego zastosować kolejną metodę, która doda kolejne znaczniki HTML itd. Zatem wiersz:

```
document.write (napis.bold().italics().fixed().link("http://www.helion.pl"));
```

można zapisać, używając znaczników HTML jako:

```
<AHREF = "http://www.helion.pl"><TT><I><B>Witamy na naszej stronie!</B></I></TT></A>
```

Istnieje jeszcze kilka innych metod operujących na obiekcie `string`, są to:

`charAt()`

składnia:

```
string.charAt(indeks);
```

Metoda ta zwraca znak o podanym indeksie, przy czym pierwsza litera łańcucha znaków ma indeks 0. Przykładowo:

```
var napis = "Helion"
document.write (napis.charAt(3));
```

Wyświetli na ekranie literę `i`.

```
indeksOf()
```

składnia:

```
string.indeksOf (wartość, [indeks]);
```

Metoda ta poszukuje ciągu znaków podanych jako `wartość`, poczynając od pozycji w łańcuchu wskazywanej przez `indeks`. Zwraca natomiast indeks pierwszego wystąpienia poszukiwanego łańcucha.

```
toUpperCase()
```

składnia:

```
string.toUpperCase()
```

Zamienia wszystkie litery w danym ciągu znaków na wielkie.

```
toLowerCase()
```

składnia:

```
string.toLowerCase()
```

Zamienia wszystkie litery w danym ciągu znaków na małe.

```
lastIndexOf()
```

składnia:

```
string.lastIndeksOf (wartość, [indeks])
```

Metoda ta jest bardzo podobna do *`indeksOf()`*, z tą tylko różnicą, że obiekt jest przeszukiwany od końca i zwracana jest pozycja ostatniego wystąpienia podanego ciągu znaków.

```
substring ()
```

składnia:

```
string.substring (indeks1, indeks2)
```

Metoda ta zwraca łańcuch znaków rozpoczynający się w pozycji `indeks1`, a kończący na pozycji `indeks2`. Wynika z tego że `indeks1` powinien być mniejszy od `indeks2`. Jeżeli jednak podamy liczby odwrotnie tzn. `indeks 2` będzie mniejszy od `indeks1`, zostaną one zamienione miejscami.

Ćwiczenie 4.12.

Stwórz skrypt, który umożliwi użytkownikowi wprowadzenie dowolnego tekstu i zamieni wszystkie litery podanego tekstu na wielkie. Wynik należy wyświetlić na ekranie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<HEAD>
<BODY>
<FONT SIZE = "5">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var napis = prompt("Podaj dowolny tekst","");
document.write (napis.toUpperCase());
// Koniec kodu JavaScript -->
</SCRIPT>
</BODY>
</HTML>
```

Obiekt Math

Obiektu `Math` używaliśmy już w jednym z ćwiczeń. Konkretnie skorzystaliśmy z jednej z jego metod — mianowicie `sqrt()` (pierwiastek kwadratowy). Obiekt ten, jak sama nazwa wskazuje, umożliwia dokonywanie różnych obliczeń matematycznych w JavaScript. Udostępnia on osiem stałych, zebranych tabeli 4.2 oraz 17 metod przedstawionych tabeli 4.3.

Tabela 4.2. *Stałe udostępniane przez obiekt Math*

Nazwa stałej	Znaczenie	Przybliżona wartość
E	stała Eulera (e)	2,718
LN2	logarytm naturalny z 2	0,693
LN10	logarytm naturalny z 10	2,302
LN2E	logarytm o podstawie 2 z e	1,442
LN10E	logarytm o podstawie 10 z e	0,434
PI	liczba ∏	3,14159
SQRT1_2	pierwiastek kwadratowy z 1/2	0,707
SQRT2	pierwiastek kwadratowy z 2	1,414

Tabela 4.3. *Metody udostępniane przez obiekt Math*

Nazwa metody	Znaczenie
abs()	zwraca wartość bezwzględną argumentu
acos()	zwraca arcus cosinus argumentu
asin()	zwraca arcus sinus argumentu
atan()	zwraca tangens sinus argumentu
ceil()	zwraca najmniejszą liczbę całkowitą większą bądź równą argumentowi
cos()	zwraca cosinus argumentu

Tabela 4.3. *Metody udostępniane przez obiekt Math — ciąg dalszy*

Nazwa metody	Znaczenie
exp()	zwraca wartość e do potęgi równej argumentowi
floor()	zwraca największą liczbę całkowitą mniejszą bądź równą argumentowi
log()	zwraca logarytm dziesiętny argumentu
max()	zwraca większy z podanych dwóch argumentów
min()	zwraca mniejszy z podanych dwóch argumentów
pow()	zwraca wartość będącą podniesieniem argumentu pierwszego do potęgi równej argumentowi drugiemu
random()	zwraca wartość pseudolosową z zakresu 0 – 1
round()	zwraca wartość argumentu zaokrągloną do najbliższej liczby całkowitej
sin()	zwraca sinus argumentu
sqrt()	zwraca pierwiastek kwadratowy argumentu
tan()	zwraca tangens argumentu

Należy pamiętać, że stałe z tabeli 4.2 to wartości tylko do odczytu, co wydaje się oczywiste oraz że funkcje trygonometryczne z tabeli 4.3 (sin, cos, tan, asin, acos, atan) wymagają podawania argumentów w radianach. Widać też, że pisane przez nas w ćwiczeniach 4.4 i 4.5 funkcje do potęgowania nie były niezbędnie konieczne, bowiem wystarczyłoby zastosować metodę pow(). Co więcej, dzięki metodzie pow() możemy także podnieść liczbę do potęgi nie będącej liczbą całkowitą. Inaczej mówiąc wyciągać pierwiastki dowolnego stopnia. Np. pierwiastek trzeciego stopnia z 8 uzyskamy na ekranie pisząc:

```
document.write (Math.pow(8, 1/3));
```

Trzeba jednak uważać, gdyż próba podniesienia wartości 8 do potęgi 2/3 bądź też próba wyciągnięcia pierwiastka trzeciego stopnia z 64 bynajmniej nie da w wyniku wartości 4 — jak wiemy z matematyki, ale pewną liczbę zmiennoprzecinkową. Co ciekawe w przypadku przeglądarki Netscape Navigator będzie 3,9999999999999996 (rysunek 4.4). Internet Explorer natomiast podaje 3,9999999999999995 (rysunek 4.5).

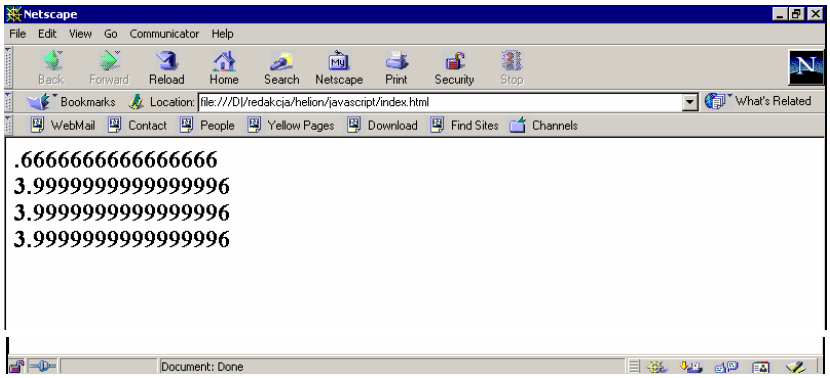
Ćwiczenie 4.13.

Napisz skrypt demonstrujący działanie funkcji potęgującej pow() przy podnoszeniu do potęgi nie będącej liczbą rzeczywistą. Sprawdzić wynik działania kodu w różnych przeglądarkach.

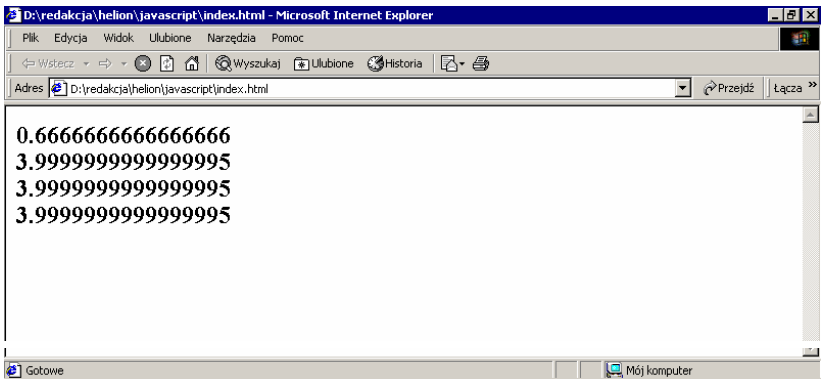
```
<HTML>
<HEAD>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<H2>
<BODY>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
document.write (2/3 + "<BR>");
document.write (Math.pow(8, .6666666666666666) + "<BR>");
document.write (Math.pow(8, 2/3) + "<BR>");
document.write (Math.pow(64, 1/3) + "<BR>");
// Koniec kodu JavaScript -->
</SCRIPT>
</BODY>
</HTML>
```

Rysunek 4.4. *Efekt wykonania skryptu z ćwiczenia 4.13 w przeglądarce Netscape Navigator*



Rysunek 4.5. *Efekt wykonania skryptu z ćwiczenia 4.13 w przeglądarce Internet Explorer*



Obiekt Date

Obiekt `Date` umożliwia wykonywanie operacji na dacie i czasie. Obiekty tego typu możemy tworzyć następująco:

1. `nazwa = new Date();`
2. `nazwa = new Date(„miesiąc dzień, rok godziny:minuty:sekundy”);`
3. `nazwa = NewDate(rok, miesiąc, dzień);`
4. `nazwa = NewDate(rok, miesiąc, dzień, godziny, minuty, sekundy).`

Dostępne metody obiektu `Date` przedstawia tabela 4.4 przedstawiona na następnej stronie.

Tabela 4.4. *Metody udostępniane przez obiekt date*

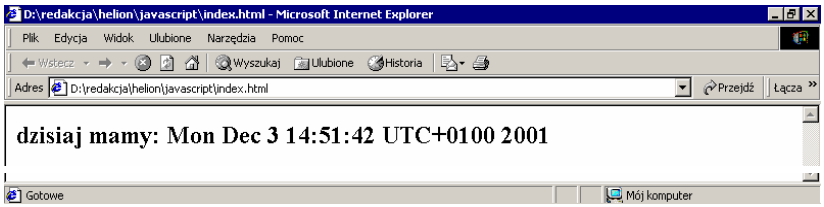
Nazwa metody	Opis
`getDate()`	zwraca dzień miesiąca
`getDay()`	zwraca dzień tygodnia (poczynając od niedzieli, która ma przypisany numer 0)
`getHours()`	zwraca godzinę
`getMinutes()`	zwraca minutę
`getMonth()`	zwraca numer miesiąca (styczeń ma numer 0)

`getSeconds()`	zwraca sekundę
`getTime()`	zwraca liczbę sekund, które upłynęły od 01.01.1970 00:00:00
`getYear()`	zwraca rok
`parse()`	zwraca liczbę sekund, które upłynęły od 01.01.1970 00:00:00 do daty podanej jako argument (`string`)
`setDate()`	ustawia dzień miesiąca
`setHours()`	ustawia godzinę
`setMinutes()`	ustawia minuty
`setMonth()`	ustawia numer miesiąca (styczeń ma numer 0)
`setSeconds()`	ustawia sekundę
`setTime()`	inicjuje obiekt liczbą sekund, które upłynęły od 01.01.1970 00:00:00
`setYear()`	ustawia rok
`toGMTString()`	zwraca datę w postaci czasu GMT
`toLocaleString()`	zwraca datę w postaci lokalnej
`UTC()`	zwraca liczbę sekund, które upłynęły od 01.01.1970 00:00:00 do daty podanej jako zestaw argumentów (liczby)

Ćwiczenie 4.14.

Użyj obiektu `date` do wyświetlania na ekranie aktualnego czasu (rysunek 4.6).

Rysunek 4.6. *Efekt użycia obiektu date do wyświetlenia aktualnej daty*

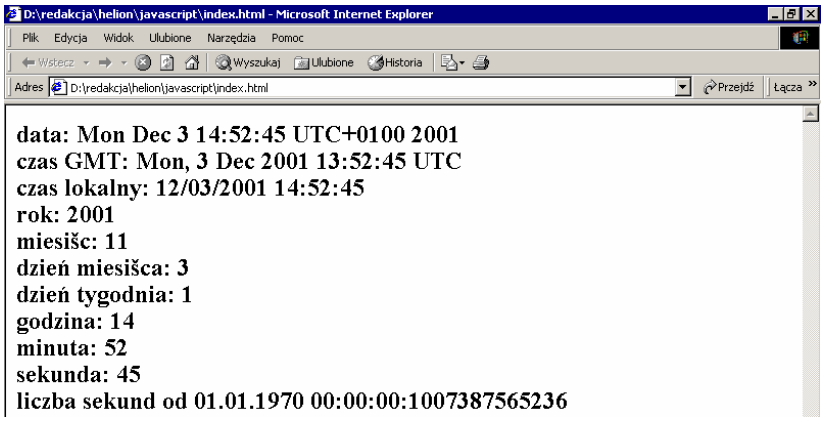


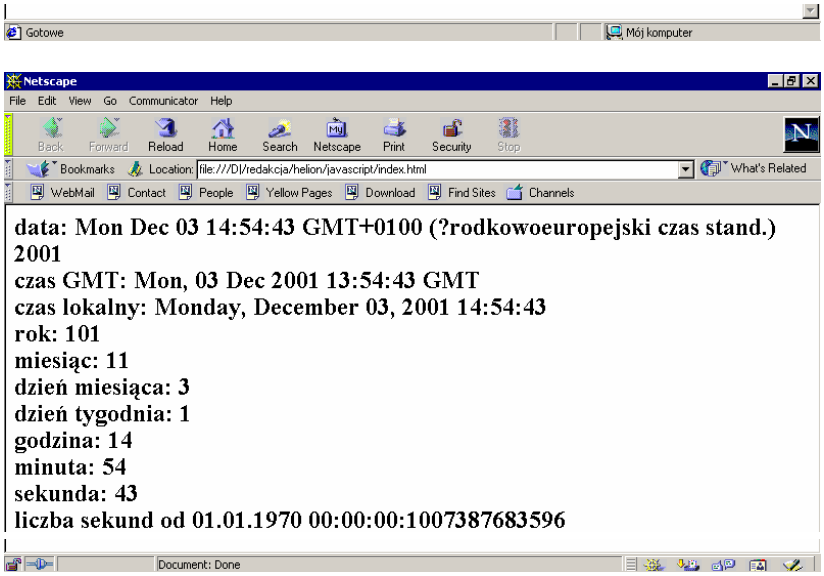

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<H2>
<BODY>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
data = new Date();
document.write ("dzisiaj mamy: " + data + "<BR>");
// Koniec kodu JavaScript -->
</SCRIPT>
</BODY>
</HTML>
```

Ćwiczenie 4.15.

Wyświetlić na ekranie aktualną datę, bieżący czas lokalny, czas GMT oraz dzień tygodnia i dzień miesiąca (rysunki 4.7 i 4.8).

Rysunek 4.7. *Wyświetlenie danych dotyczących daty i czasu w przeglądarce Internet Explorer*





Rysunek 4.8. *Wyświetlenie danych dotyczących daty i czasu w przeglądarce Internet Explorer*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<H2>
<BODY>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptów
data = new Date();
document.write ("data: " + data + "<BR>");
document.write ("czas GMT: " + data.toGMTString() + "<BR>");
document.write ("czas lokalny: " + data.toLocaleString() + "<BR>");
document.write ("rok: " + data.getYear() + "<BR>");
document.write ("miesiąc: " + data.getMonth() + "<BR>");
document.write ("dzień miesiąca: " + data.getDate() + "<BR>");
document.write ("dzień tygodnia: " + data.getDay() + "<BR>");
document.write ("godzina: " + data.getHours() + "<BR>");
document.write ("minuta: " + data.getMinutes() + "<BR>");
document.write ("sekunda: " + data.getSeconds() + "<BR>");
document.write ("liczba sekund od 01.01.1970 00:00:00:" + data.getTime() + "<BR>");
// Koniec kodu JavaScript -->
</SCRIPT>
</BODY>
</HTML>
```

Warto zwrócić uwagę na różnice w wyświetlanych danych po wczytaniu skryptu do różnych przeglądarek. Efekt działania kodu w Internet Explorerze jest widoczny na rysunku 4.7, natomiast w Nestscape Navigatorze na rysunku 4.8. Widać też wyraźnie, że Navigator niepoprawnie podaje rok, jeśli użyjemy metody `getYear()`. W ten właśnie sposób objawia się w tej przeglądarce słynny problem roku 2000.

Obiekt document

Z obiektem `document` spotykaliśmy się już wielokrotnie, choćby pisząc w skrypcie instrukcje `document.write()`. Zawiera on informacje na temat dokumentu znajdującego się w każdym oknie bądź ramce oraz metody na wyświetlanie danych w formacie HTML. Obiekt `document` definiujemy, używając standardowej składni języka HTML w postaci:

```
<BODY
BACKGROUND = ”plik tła”
BGCOLOR = ”kolor tła”
TEXT = ”kolor tekstu”
LINK = ”kolor nieużytego odnośnika”
ALINK = ”kolor aktywowanego odnośnika”
VLINK = ”kolor użytego odnośnika”
[onLoad = ”procedura obsługi”]
[onUnload = ”procedura obsługi”]>
zawartość dokumentu
</BODY>
```

Parametr `BACKGROUND` specyfikuje URL pliku graficznego, który zostanie użyty jako tło, `BGCOLOR` podaje kolor tła, `TEXT` — kolor tekstu, `LINK`, `ALINK` i `VLINK` — kolory przypisane odnośnikom w różnych ich stanach. `onLoad` i `onUnload` spe-

cyfikują natomiast nazwy procedur, które mają zostać wykonane podczas ładowania oraz opuszczania strony przez użytkownika.

Wartości parametrów znacznika `<BODY>` możemy uzyskać, odwołując się do właściwości obiektu `document`. Ich lista znajduje się w tabeli 4.5.

Tabela 4.5. *Właściwości obiektu document*

Nazwa właściwości	Znaczenie
`AlinkColor`	wartość koloru, jaki przyjmuje aktywowany odnośnik
`LinkColor`	wartość koloru, jaki przyjmuje jeszcze nie użyty i nie aktywowany odnośnik
`vlinkColor`	wartość koloru, jaki przyjmuje użyty odnośnik
`bgColor`	kolor podkładu
`fgColor`	kolor tekstu
`anchors`	tablica zawierająca wszystkie „kotwice” w dokumencie
`cookie`	specyfikacja obiektu `cookie`
`forms`	tablica zawierająca wszystkie formularze w dokumencie
`lastModified`	data ostatniej modyfikacji dokumentu
`links`	tablica zawierająca wszystkie linki znajdujące się w danym dokumencie
`location`	kompletny URL bieżącego dokumentu
`referrer`	URL dokumentu, z którego został wywołany bieżący dokument
`title`	tytuł dokumentu podany znacznikiem HTML `<TITLE>`

Ćwiczenie 4.16.

Wyświetlić dostępne właściwości obiektu `document`.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<TITLE> Przykładowy skrypt </TITLE>
</HEAD>
<H2>
<BODY
  LINK = "blue"
  ALINK = "red"
  VLINK = "white"
  BGCOLOR = "silver"
  TEXT = "black"
>
<A HREF = "http://helion.pl"></A>
<A HREF = "http://www.google.com"></A>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
document.fgColor = "white"
document.write ("ALINK = " + document.alinkColor + "<BR>");
document.write ("LLINK = " + document.linkColor + "<BR>");
document.write ("VLINK = " + document.vlinkColor + "<BR>");
document.write ("BGCOLOR = " + document.bgColor + "<BR>");
document.write ("FGCOLOR = " + document.fgColor + "<BR>");
document.write ("LOCATION = " + document.location + "<BR>");
document.write ("REFERRER = " + document.referrer + "<BR>");
document.write ("TITLE = " + document.title + "<BR>");
document.write ("LAST MODIFIED = " + document.lastModified + "<BR>");
document.write ("LINKS:" + "<BR>");
for (var i in document.links){
  document.write (document.links[i] + "<BR>");
}
document.write ("FORMS:" + "<BR>");
for (var i in document.forms){
  document.write (document.forms[i] + "<BR>");
}
// Koniec kodu JavaScript -->
</SCRIPT>
</BODY>
</HTML>
```

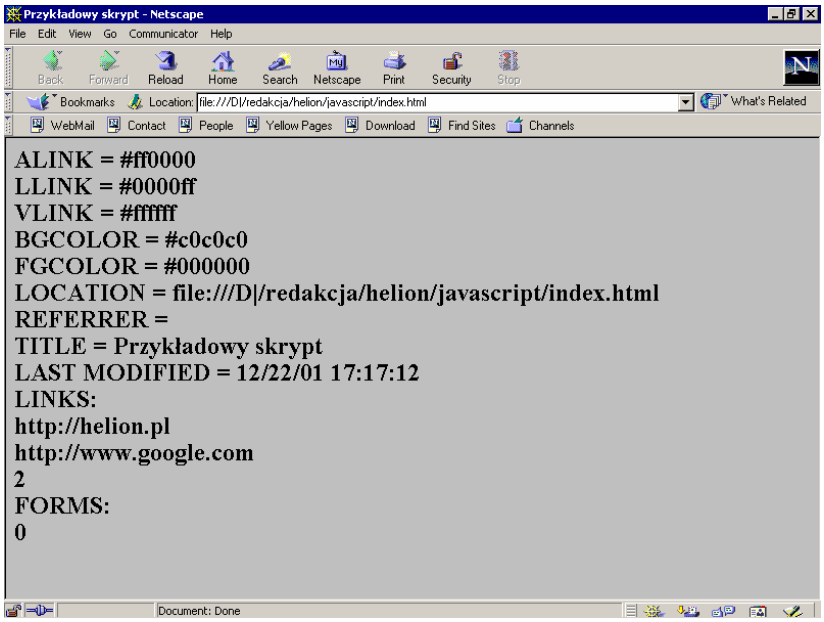
W skrypcie wykorzystaliśmy pętlę przeglądającą wszystkie właściwości danego obiektu (w tym przypadku obiektów: `links` oraz `forms`). Ma ona składnię:

```
for (var zmienna in obiekt){
  instrukcje
}
```

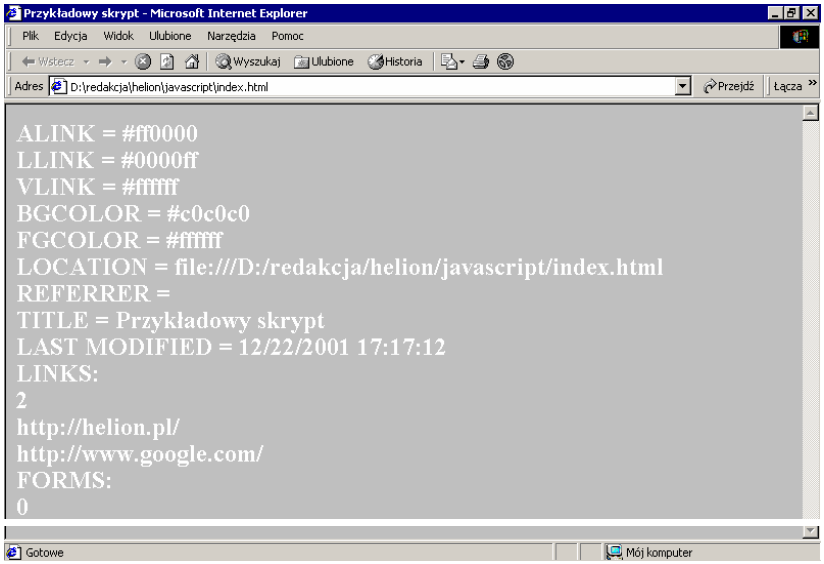
Pętla ta pozwala na przejście po wszystkich właściwościach podanego obiektu. Dokładnie jak w powyższym przykładzie.

Właściwość `referrer` zawiera pusty ciąg znaków, jako że strona została wczytana bezpośrednio, a nie poprzez wywołanie odnośnika z innej strony. Warto zwrócić uwagę, że mimo iż zawarte w stronie dwa odnośniki nie są widoczne na ekranie, znajdują się we właściwościach obiektu `document` oraz że w przypadku Netscape Navigatora skrypt został wykonany... błędnie. Proszę zwrócić uwagę, że w kodzie znajduje się wiersz `document.fgColor = „white"`, która powinna spowodować przypisanie wyświetlanemu tekstowi koloru białego, podczas gdy pozostaje on (według specyfikacji w sekcji `<BODY>`) czarny (rysunku 4.9). Błąd ten nie występuje w Internet Explorerze, który stronę wyświetla zgodnie z założeniami (rysunek 4.10).

Rysunek 4.9. *Efekt działania skryptu z ćwiczenia 4.16 w przeglądarce Netscape Navigator*



Rysunek 4.10. *Efekt działania skryptu z ćwiczenia 4.16 w przeglądarce Internet Explorer*



Każdy element tablicy zawierającej odnośniki (z wyjątkiem ostatniej pozycji, która jest liczbą całkowitą odzwierciedlającą ilość odnośników na stronie) jest obiektem typu `link`, który ma następujące właściwości:

- `host` — fragment adresu w postaci: `nazwahosta:port`;
- `hostname` — nazwa hosta oraz domena lub adres IP;
- `pathname` — adres ścieżki dostępu do pliku oraz nazwę pliku;
- `port` — numer portu (standardowo dla `http` jest to port 80);
- `protocol` — nazwa protokołu (`http`, `ftp`, `gopher`);
- `search` — zapytanie przesyłane z adresem URL;
- `target` — wartość parametru `TARGET` znacznika `<A>`.

Ćwiczenie 4.17.

Wyświetl parametry odnośnika znajdującego się w dokumencie HTML.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<TITLE> Przykładowy skrypt </TITLE>
</HEAD>
<H2>
<BODY>
<A HREF = "http://www.pewien.adres.com:80/sciezka/index.htm"
   TARGET = "ramka1">
Pewien odnośnik
</A>
<BR><BR>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptów
document.write ("HOST = " + document.links[0].host + "<BR>");
document.write ("HOSTNAME = " + document.links[0].hostname + "<BR>");
document.write ("HREF = " + document.links[0].href + "<BR>");
document.write ("PATHNAME = " + document.links[0].pathname + "<BR>");
document.write ("PORT = " + document.links[0].port + "<BR>");
document.write ("PROTOCOL = " + document.links[0].protocol + "<BR>");
document.write ("TARGET = " + document.links[0].target + "<BR>");
// Koniec kodu JavaScript -->
</SCRIPT>
</BODY>
</HTML>
```

Obiekt window

Obiekt `window` znajduje się na samym szczycie hierarchii obiektów w JavaScript. Do jego metod możemy się więc odwoływać, pisząc `window.metoda()` lub po prostu `metoda()`. Trzecim sposobem jest napisanie `self.metoda()`. Wszystkie właściwości zebrane są w tabeli 4.6.

Tabela 4.6. *Właściwości obiektu window*

Właściwość	Opis
`defaultStatus`	tekst domyślnie wyświetlany na pasku stanu okna przeglądarki
`frames`	tablica ramek zawartych w bieżącym oknie
`length`	liczba ramek
`location`	bieżący adres URL
`self`	synonim dla nazwy bieżącego okna
`status`	tekst wyświetlany na pasku stanu
`window`	synonim dla nazwy bieżącego okna

Właściwość `location`, czyli bieżący adres URL, nie jest tak naprawdę ciągiem znaków określających odnośnik, ale obiektem typu o typie `location`. Obiekt ten ma swoje własne właściwości przedstawione w tabeli 4.7.

Tabela 4.7. *Właściwości obiektu location*

Właściwość	Opis
`host`	fragment adresu w postaci: `nazwa hosta:port`
`hostname`	nazwa hosta oraz domena lub adres IP
`href`	pełny adres URL
`pathname`	adres ścieżki dostępu do pliku oraz nazwa pliku
`port`	numer portu (standardowo dla http jest to port 80)
`protocol`	nazwa protokołu (http, ftp, gopher)
`search`	zapytanie przesyłane z adresem URL
`target`	wartość parametru `TARGET` znacznika `<A>`

Jak widać, jest to zestaw bardzo podobny do właściwości obiektu `link`. Różni się jednak m.in. tym, że wartości te można w przypadku obiektu `location` modyfikować. Możemy zatem wymusić np. załadowanie innej strony WWW. Wystarczy, że napiszemy gdzieś w dokumencie wiersz:

```
window.location.href = ”http://helion.pl”
```

a przeglądarka spróbuje połączyć się ze stroną WWW Wydawnictwa Helion. Jest to jeden ze sposobów realizacji przeadresowywania, tzn. po połączeniu się z danym adresem użytkownik automatycznie zostanie połączony z innym.

Ćwiczenie 4.18.

Napisać skrypt, który po wczytaniu automatycznie połączy się z inną stroną WWW.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<TITLE> Automatyczne przeadresowanie </TITLE>
</HEAD>
<BODY>
<H2><P ALIGN = "center">Strona została przeniesiona w inne miejsce</P>
<P ALIGN = "center"> Zostaniesz automatycznie połączony z nową lokalizacją<P>
</H2>
</BODY>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
window.location.href = "http://helion.pl";
// Koniec kodu JavaScript -->
</SCRIPT>
</HTML>
```

Oczywiście realizacja automatycznego przeadresowania jest również możliwa bez używania skryptów. Wystarczy w sekcji `<HEAD>` umieścić znacznik `<META>` z odpowiednią zawartością:

```
<META HTTP-EQUIV="refresh" CONTENT="n; URL=url">
```

gdzie `n` oznacza czas (w sekundach), po jakim nastąpi załadowanie nowej strony, a `url` to adres tej strony.

Ćwiczenie 4.19.

Napisz procedurę automatycznego przeadresowania bez użycia JavaScript.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<META HTTP-EQUIV="refresh" CONTENT="1; URL=http://helion.pl">
<TITLE> Automatyczne przeadresowanie </TITLE>
</HEAD>
<BODY>
<H2>
<P ALIGN = "center">Strona została przeniesiona w inne miejsce</P>
<P ALIGN = "center"> Zostaniesz automatycznie połączony z nową lokalizacją>
</H2>
</BODY>
</HTML>
```

Ćwiczenie 4.20.

Wyświetl na pasku stanu przeglądarki dowolny napis.

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<SCRIPT language = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
status = "Moja strona WWW";
// Koniec kodu JavaScript -->
</SCRIPT>
<BODY>
</BODY>
</HTML>
```

Obiekt `window` udostępnia również kilka metod, pokazanych w tabeli 4.8. Będą one przydatne przy omawianiu formularzy i zdarzeń w rozdziale piątym.

Tabela 4.8. *Metody udostępniane przez obiekt window*

Nazwa metody	Działanie
`alert`	wyświetla okno z wiadomością dla użytkownika
`close`	zamyka okno przeglądarki
`confirm`	wyświetla okno dialogowe z przyciskami *OK* i *Cancel*
`open`	otwiera nowe okno przeglądarki
`prompt`	wyświetla okno umożliwiające wprowadzenie wartości przez użytkownika
`setTimeout`	umożliwia wykonanie zadanego polecenia po określonym czasie
`clearTimeout`	anuluje licznik czasu ustawiony poleceniem `setTimeout`

Rozdział 5.

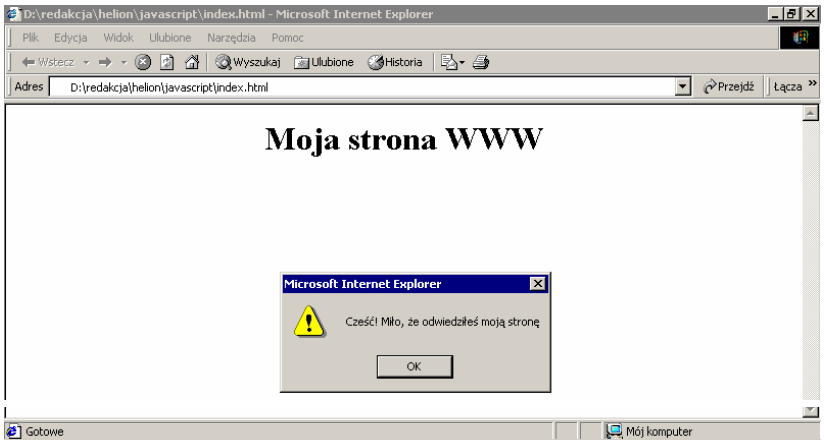
Zdarzenia i formularze

Zdarzenia onLoad i onUnload

Na stronie mogą zachodzić różne zdarzenia, np. użytkownik kliknie myszą lub zacznie wprowadzać dane do formularza. JavaScript pozwala na oprogramowanie tego typu zdarzeń poprzez procedury ich obsługi. Jedną z takich procedur jest `onLoad`, która zostanie wykonana po załadowaniu strony do przeglądarki. Możemy ją wykorzystać np. do powitania użytkownika.

Ćwiczenie 5.1.

Napisz skrypt wyświetlający po załadowaniu strony WWW okno powitalne (rysunek 5.1).



Rysunek 5.1. *Okno powitalne wykorzystujące zdarzenie onLoad*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<BODY onLoad = "alert ('Cześć! Miło, że odwiedziłeś moją stronę')">
<H1><CENTER>
Moja strona WWW
</H1></CENTER>
</BODY>
</HTML>
```

Warte uwagi jest, że okienko z powitaniem (rysunek 5.1) pojawi się dopiero po pełnym załadowaniu strony. Jeśli zatem w treści umieściliśmy wywołania jakichś innych funkcji, zostaną one wykonane w pierwszej kolejności! Ogólnie składnia definiująca procedurę obsługi zdarzenia wygląda następująco:

```
<znacznik_HTML parametry_znacznika nazwa_zdarzenia = "instrukcje_Java_Script"
```

lub

```
<znacznik_HTML parametry_znacznika nazwa_zdarzenia = "nazwa_funkcji_w_Java_Script"
```

Ponieważ procedura czy funkcja obsługująca dane zdarzenia może być dosyć skomplikowana, wygodniejsze i bardziej czytelne jest stworzenie oddzielnej funkcji, a następnie przypisanie jej do zdarzenia. Zwykle tak właśnie będziemy postępować w kolejnych ćwiczeniach. Wyjątkiem są oczywiście sytuacje, kiedy procedura obsługi zdarzenia składa się tylko z jednej prostej instrukcji i śmiało można ją umieścić przy znaczniku.

Zdarzeniem analogicznym do `onLoad` jest `onUnload`, z tym że jest ono wykonywane przy opuszczaniu strony przez użytkownika. Można je więc wykorzystać do pożegnania.

Ćwiczenie 5.2.

Napisz skrypt wyświetlający okno pożegnalne przy opuszczaniu strony WWW.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<BODY onUnload = "alert ('Do widzenia!\nMamy nadzieję, że niedługo znów nas odwiedzisz!')">
<H1><CENTER>
Moja strona WWW
</H1></CENTER>
</BODY>
</HTML>
```

Ćwiczenie 5.3.

Napisz skrypt, który przy ładowaniu strony zapyta użytkownika o imię, powita go, a przy opuszczeniu witryny pożegna, korzystając z podanego imienia.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = JavaScript>
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function powitanie (imie){
    alert ("Cześć! Witamy na naszej stronie " + ((imie=="brak")?"":imie) + "!");
}
function podaj_imie (){
    imie = prompt ("Podaj nam swoje imię", "");
    if (imie == null || imie == ""){
        imie = "brak";
        document.write ("Miło, że jesteś z nami!<BR>");
        document.write ("Mamy nadzieję, że znajdziesz tu coś ciekawego dla siebie.");
    }
    else{
        document.write ("Miło, że jesteś z nami " + imie + "!<BR>");
        document.write ("Mamy nadzieję, że znajdziesz tu coś ciekawego dla siebie.");
    }
    return imie;
}

function pozegnanie (imie){
    if (imie != "brak"){
        alert ("Do widzenia " + imie + "\nMamy nadzieję, że niedługo znów nas odwiedzisz");
    }
    else{
        alert ("Do widzenia!\nMamy nadzieję że niedługo znów nas odwiedzisz");
    }
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY onLoad = "powitanie(imie)"; onUnload = "pozegnanie (imie)">
<H2>
<SCRIPT LANGUAGE = JavaScript>
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
var imie = podaj_imie();
// Koniec kodu JavaScript -->
</SCRIPT>
</H2>
<BODY>
</HTML>
```

Całość działa w sposób następujący. Najpierw zostanie załadowana treść strony, w tym przypadku jest to sekwencja:

```
var imie = podaj_imie();
```

Zostanie więc wykonana funkcja JavaScript `podaj_imie()`. Funkcja ta, wykorzystując znane już nam konstrukcje, pyta się użytkownika o jego imię, a następnie sprawdza, czy podany został jakiś łańcuch znaków. Jeśli tak, to na ekranie pojawia się część strony z uwzględnieniem podanego imienia i funkcja zwraca wprowadzony przez użytkownika łańcuch znaków. Ciąg ten staje się wartością zmiennej o nazwie `imie`. Następnie wykonywana jest procedura `onLoad`, czyli de facto funkcja `powitanie()`. Wyświetla ona dodatkowe okienko z powitaniem, wykorzystując jako parametr podany łańcuch znaków.

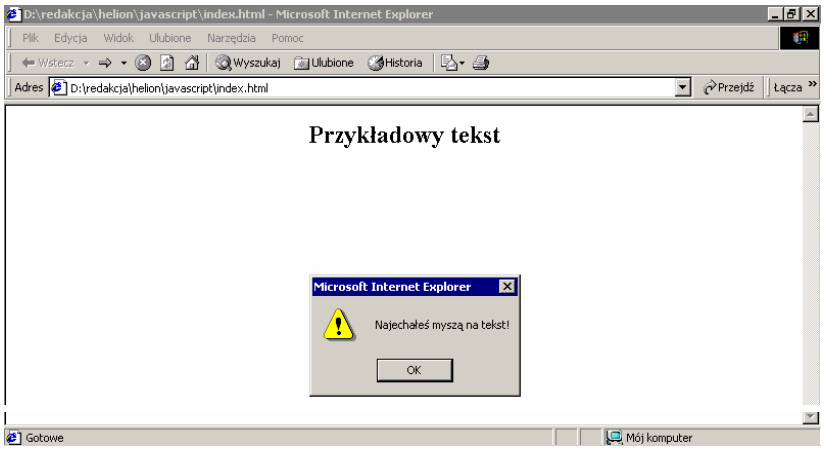
Przy opuszczaniu strony wywoływana jest procedura `onUnload` (czyli przypisana jej funkcja `pozegnanie()`). Jako parametr otrzymuje ona również łańcuch znaków ze zmiennej `imie` i wykorzystuje go w wyświetlanym oknie pożegnalnym. W przypadku, gdy nie podamy imienia, wartością zmiennej `imie` staje się łańcuch znaków „brak”. Wykorzystujemy ten fakt w funkcjach `powitanie()` i `pozegnanie()` do stwierdzenia, którą wersję strony i okna dialogowego mamy wyświetlić.

Zdarzenia związane z myszą

Zdarzenia związane z obsługą myszy to `onMouseOver`, `onMouseOut` i `onClick`. Zdarzenie `onMouseOver` zachodzi, kiedy kursor myszy znajdzie się nad obiektem, którego ma dotyczyć. Analogicznie `onMouseOut` zachodzi, kiedy kursor opuści obiekt, a `onClick`, kiedy obiekt zostanie kliknięty.

Ćwiczenie 5.4.

Napisz skrypt, który gdy naprowadzimy myszkę na tekst znajdujący się w dokumencie, wyświetli okno dialogowe z dowolnym napisem (rysunek 5.2).



Rysunek 5.2. *Efekt działania skryptu z ćwiczenia 5.4*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<H2><CENTER>
<P onMouseOver = "alert('Najechałeś myszą na tekst!')">Przykładowy tekst</P>
</H2></CENTER>
</BODY>
</HTML>
```

Niestety przykład ten nie zadziała w przypadku przeglądarki Netscape Navigator, gdyż nie obsługuje ona zdarzenia `onMouseOver` przypisanego do paragrafów tekstowych (znacznik `<P>`).

Ćwiczenie 5.5.

Napisz skrypt, który po kliknięciu na odnośnik zawarty w dokumencie zamknie okno przeglądarki.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<BODY>
<A HREF="#"
   onClick="window.close()";
>
Zamknij okno</A>
</BODY>
</HTML>
```

Ćwiczenie 5.6.

Napisz skrypt, który po kliknięciu na przycisk zapyta użytkownika, czy na pewno chce opuścić bieżąca stronę. Jeśli tak, należy wywołać nową stronę.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE="JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function obsluga_zdarzenia(){
  if (confirm ('Czy na pewno chcesz przejść na stronę http://helion.pl?'))
    window.location.href = "http://helion.pl";
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H2><CENTER>
<FORM>
<INPUT TYPE = "button"
       VALUE = "helion.pl"
       onClick = "obsluga_zdarzenia()";
>
</FORM>
</H2></CENTER>
</BODY>
</HTML>
```

Ćwiczenie 5.7.

Napisz skrypt, który będzie zmieniał napis na pasku stanu, kiedy użytkownik najedzie kursorem lub kliknie na odnośnik znajdujący się w dokumencie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "javascript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function ObslugaPaskaStanu (){
  window.status='Strona Wydawnictwa Helion';
}
function ObslugaKlikniecia (){
  window.status="Otwieram nową stronę";
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H2><CENTER>
<A HREF = "http://helion.pl"
   onClick = "ObslugaKlikniecia(); return true;"
   onMouseOver = "ObslugaPaskaStanu();return true;"
>
Strona wydawnictwa Helion
</A>
</H2></CENTER>
<SCRIPT LANGUAGE = "javascript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
window.defaultStatus = "Przykładowa strona z obsługą skryptów";
// Koniec kodu JavaScript -->
</SCRIPT>
</BODY>
</HTML>
```

Domyślnie na pasku stanu wyświetlany jest tekst „Przykładowa strona z obsługą skryptów”. Po naprowadzeniu kursora na odnośnik wywoływana jest procedura obsługi zdarzenia `onMouseOver`. W naszym przypadku jest to funkcja `ObsługaPaskaStanu()`, która zmienia tekst na „Strona Wydawnictwa Helion”. W przypadku kliknięcia wywoływana jest procedura zdarzenia `onClick`, czyli `ObsługaKliknięcia()` i na krótką chwilę pojawia się napis „Otwieram nową stronę”.

Formularze

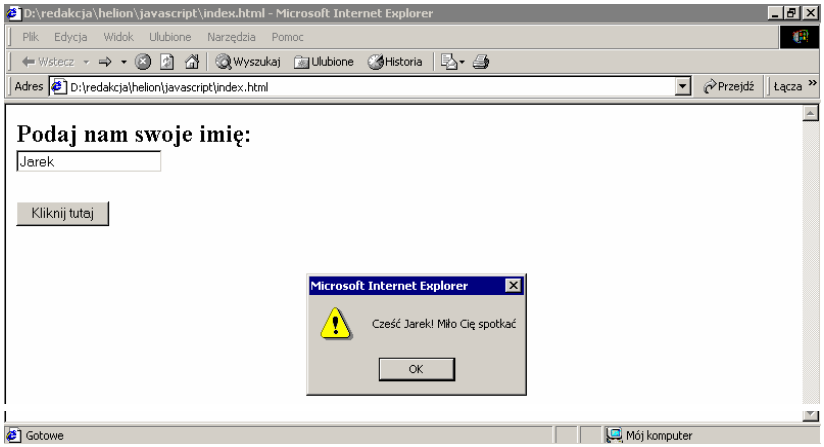
Formularze są reprezentowane przez obiekt o nazwie `form`. Tworzymy go, korzystając z typowej konstrukcji języka HTML, czyli znacznika `FORM` z parametrami:

```
<FORM NAME = ”nazwa” TARGET = ”okno” ACTION = ”url” METHOD = ”metoda” ENCTYPE = ”typ kodowania”>
 </FORM>
```

Nazwa jest po prostu nazwą formularza. Parametr `TARGET` podaje nazwę okna, w którym ma się pojawić odpowiedź. Może on zawierać nazwę okna lub ramki. `ACTION` podaje lokalizację serwera, do którego mają zostać wysłane dane zebrane z formularza. Zwykle jest to adres skryptu CGI. `METHOD` podaje sposób wysłania informacji do serwera, natomiast `ENCTYPE` — sposób kodowania `MIME`. Więcej informacji na temat formularzy i sposobów przekazywania za ich pomocą informacji można znaleźć w dokumentach RFC 1866, RFC 1867. W tej chwili nie będziemy się zajmowali dokładnym opisem ich tworzenia, przejdziemy natomiast od razu do wykorzystania JavaScriptów w ich obsłudze.

Ćwiczenie 5.8.

Napisz skrypt wyświetlający pole, które umożliwia wprowadzenie imienia użytkownika oraz przycisk *Kliknij tutaj*. Po kliknięciu na przycisk skrypt ma sprawdzić, czy został podany jakiś tekst i wyświetli powitanie (rysunek 5.3) lub ponowną prośbę o podanie imienia.



Rysunek 5.3. *Wykorzystanie formularzy i obsługi zdarzeń do wyświetlenia powitania*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function przetwarzaj_dane (formularz){
  if (formularz.imie.value == "")
    alert ("A może jednak podałbyś swoje imię?");
  else
    alert ("Cześć " + formularz.imie.value + "! Miło Cię spotkać");
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H2>
<FORM NAME = "formularz1">
Podaj nam swoje imię:
<BR>
<INPUT TYPE = "text" NAME = "imie" >
<BR><BR>
<INPUT TYPE = "button" VALUE = "Kliknij tutaj" onClick = "przetwarzaj_dane(this.form)">
</H2>
</BODY>
</HTML>
```

W sekcji `<BODY>` dokumentu HTML w klasyczny tworzymy sposób pole tekstowe, nadając mu nazwę `imie` oraz pole typu `przycisk`. Przyciskowi nie nadajemy nazwy, nie będzie ona tutaj do niczego potrzebna. Określamy jednak procedurę, która ma zostać wykonana po kliknięciu na nim. Podajemy zatem nazwę zdarzenia `onClick` i przypisujemy mu procedurę `przetwarzaj_dane()`.

Do tej procedury (funkcji) jako parametr przekazujemy `this.form`, czyli obiekt reprezentujący bieżący formularz o nazwie `formularz1`. Odwołujemy się do niego w funkcji `przetwarzaj_dane()` za pomocą nazwy zmiennej — `formu-`

larz. Ponieważ polu do wprowadzania tekstu nadaliśmy nazwę imie, możemy się teraz do niego odwoływać, pisząc formularz.imie.

Dokładnie chodzi nam o sprawdzenie wartości tego pola, piszemy zatem formularz.imie.value. Dalej następuje klasyczna konstrukcja if...else, której nie trzeba chyba tłumaczyć. Gdybyśmy chcieli, aby pole tekstowe nie było na początku puste, ale zawierało jakąś wartość domyślną, powinniśmy dodać do definicji tego pola parametr value np.

```
<INPUT TYPE = ”text” NAME = ”imie” VALUE = ”Marek”>
```

Konkretny formularz jest tak naprawdę elementem zbioru wszystkich formularzy w dokumencie. A więc żeby się do niego dostać, można również napisać: document.forms. formularz1 lub też skorzystać z będącej częścią dokumentu tablicy zawierającej wszystkie formularze. Odwołujemy się do niej, pisząc document.forms[indeks], gdzie indeks jest liczbą określającą kolejny formularz. Liczbę wszystkich formularzy w dokumencie możemy określić, korzystając z argumentu length obiektu forms.

Ćwiczenie 5.9.

Napisz skrypt sprawdzający ilość formularzy w dokumencie i wyświetlający tę wartość w polu tekstowym.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function przetwarzaj_dane (){
  document.forms[0].liczba.value = document.forms.length
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H2>
<FORM NAME = "formularz1">
Ilość formularzy w dokumencie:
<input type = "text" name = "liczba" >
</FORM>
<FORM NAME = "formularz2">
<input type = "button" value = "Kliknij tutaj" onClick = "przetwarzaj_dane()">
</FORM>
</BODY>
</HTML>
```

Jak widać, aby „dostać się” do formularza, nie trzeba przekazywać go jako parametru funkcji przetwarzaj_dane(), jak w poprzednim ćwiczeniu. Można skorzystać z opisanych wyżej konstrukcji języka. Trzeba jedynie pamiętać, że elementy tablicy forms numerowane są od zera, a nie od jedynki. Zatem formularz1 to forms[0], formularz2 to forms[1], itd.

Obiekt formularza udostępnia właściwości zebrane w tabeli 5.1 oraz jedną metodę — submit.

Tabela 5.1. *Właściwości udostępniane przez obiekt form*

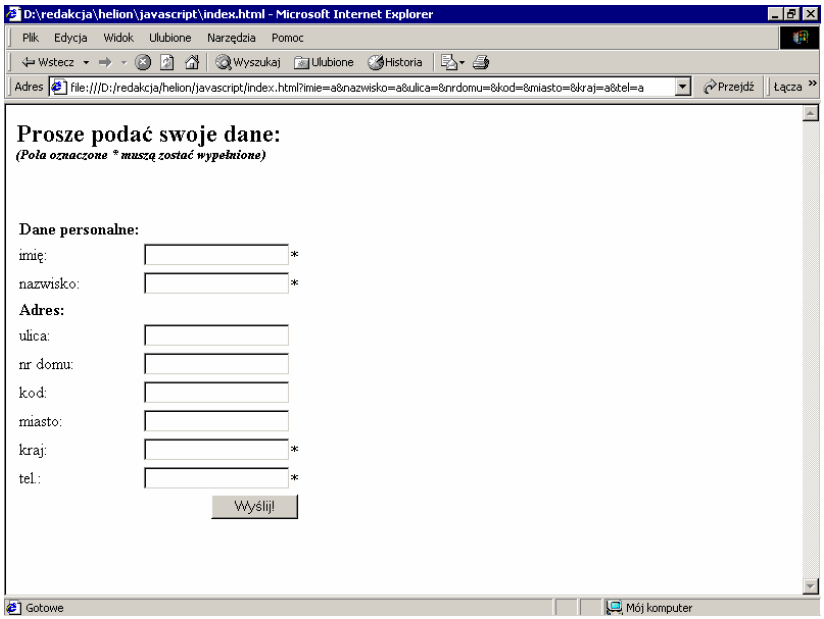
Nazwa metody	Znaczenie
action	wartość parametru ACTION w definicji formularza
Elements	tablica elementów, z których składa się dany formularz
Encoding	typ kodowania, czyli wartość parametru ENCTYPE etykiety <FORM>
Length	liczba elementów zawartych w formularzu
Method	wartość parametru METHOD etykiety <FORM>
Target	wartość parametru TARGET etykiety <FORM>

Dzięki tym właściwościom możemy dynamicznie zmieniać sposób zachowania formularza zawarty w jego definicji. Oprócz właściwości mamy dostępną jedną metodę, mianowicie submit, która powoduje wysłanie danych z formularza do serwera. Możemy wykorzystać ją np. do sprawdzenia, czy użytkownik podał wszystkie wymagane przez nas dane.

Ćwiczenie 5.10.

Utwórz formularz jak na rysunku 5.4. Po naciśnięciu przycisku *Wyślij*, skrypt ma sprawdzić, czy użytkownik podał wszystkie wymagane dane.

Rysunek 5.4.
Wygląd formularza z ćwiczenia 5.10




```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = JavaScript>
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function przetwarzaj_dane (){
  var brakuje_danych = false;
  var formularz = document.forms[0];
  var napis = "";
  if (formularz.imie.value == ""){
    napis += "imie\n"
    brakuje_danych = true;
  }
  if (formularz.nazwisko.value == ""){
    napis += "nazwisko\n"
    brakuje_danych = true;
  }
  if (formularz.kraj.value == ""){
    napis += "kraj\n"
    brakuje_danych = true;
  }
  if (formularz.tel.value == ""){
    napis += "telefon\n"
    brakuje_danych = true;
  }
  if (!brakuje_danych)
    formularz.submit();
  else
    alert ("Nie wypełniłeś następujących pól:\n" + napis);
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H2>
<FORM NAME = "formularz1">
Proszę podać swoje dane:
<FONT SIZE = "-1"><I><BR>
(Pola oznaczone * muszą zostać wypełnione)
</FONT></I><BR><BR>
<TABLE>
<TR><TD><B>
Dane personalne:</B>
</TD><TD></TD></TR>
<TR><TD>
imię:</TD><TD>
<input type = "text" name = "imie">*</TD></TR>
<TR><TD>
nazwisko:</TD><TD>
<input type = "text" name = "nazwisko">*</TD></TR>
<TR><TD><B>
Adres:</B></TD><TD>
</TD>
<TR><TD>
ulica:</TD><TD>
<input type = "text" name = "ulica"></TD></TR>
```

```
<TR><TD>
nr domu:</TD><TD>
<input type = "text" name = "nrdomu"></TD></>TR
<TR><TD>
kod:</TD><TD>
<input type = "text" name = "kod"></TD></TR>
<TR><TD>
miasto:</TD><TD>
<input type = "text" name = "miasto"></TD></TR>
<TR><TD>
kraj:</TD><TD>
<input type = "text" name = "kraj">*</TD></TR>
<TR><TD>
tel.:</TD><TD>
<input type = "text" name = "tel">*</TD></TR>
<TR><TD>
</TD><TD ALIGN = "right">
</H2>
<input type = "button" name = "wyslij" value = "   Wyślij!   " onClick = "przetwarzaj_dane()">
</TD></TR>
</TABLE>
</BODY>
</HTML>
```

Powyższy skrypt generuje formularz przedstawiony na rysunku 5.4. Jest on formatowany za pomocą tabel HTML. Każde pole tekstowe formularza ma swoją nazwę, podawaną parametrem `name`, tak że możemy je bez problemu zidentyfikować. Na samym dole nie umieszczamy przycisku typu `submit`, jak w klasycznym formularzu, tylko zwyczajny przycisk typu `button`. Dodajemy natomiast do niego procedurę obsługi zdarzenia `onClick` o nazwie `przetwarzaj_dane()`. Procedura ta sprawdza po kolei wartości interesujących nas pól i jeżeli dane pole nie zostało wypełnione, dodaje do zmiennej `napis` nazwę tego pola oraz nadaje zmiennej `brak_danych` wartość `true` (prawda). Na samym końcu sprawdzamy wartość tej zmiennej. Jeżeli jest ona równa `true`, czyli brakuje jakichś danych, za pomocą metody `alert()` informujemy o tym użytkownika. Konkretnie wyświetlamy nazwy niewypełnionych pól oraz kończymy wykonywanie procedury. Jeżeli wartość zmiennej `brak_danych` nie została zmodyfikowana, wywołujemy metodę `submit()`. Powoduje ona przesłanie danych do serwera. Przypisanie `formularz = document.forms[0]` ma na celu jedynie uproszczenie zapisu.

Skrypt ten można zmodyfikować również w taki sposób, aby w przypadku nie wypełnienia pól, które nie są wymagane, była im przypisywana wartość np. „`brak danych`”. Może być to przydatne w późniejszej analizie danych. Nie jest to skomplikowane. Wystarczy sprawdzać wartości poszczególnych pól i w przypadku gdy zawierają pusty łańcuch znaków, przypisać im wartość „`brak danych`”, czyli

```
if (formularz.nazwa_pola.value == ””) formularz.nazwa_pola.value = ”<brak danych>”;
```

Zamiast korzystać z metody `submit()`, możemy również, w celu uzyskania tego samego efektu, wykorzystać zdarzenie `onSubmit`.

Ćwiczenie 5.11.

Utwórz formularz jak na rysunku 5.4. Po naciśnięciu przycisku *Wyślij* skrypt ma sprawdzić, czy użytkownik podał wszystkie wymagane dane. Jeśli pola nieoznaczone gwiazdkami nie zostaną wypełnione, nadaj im wartość brak danych. Wykorzystaj zdarzenie `onSubmit`.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = JavaScript>
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function przetwarzaj_dane (){
  var brakuje_danych = false;
  var formularz = document.forms[0];
  var napis = "";
  if (formularz.imie.value == ""){
    napis += "imie\n"
    brakuje_danych = true;
  }
  if (formularz.nazwisko.value == ""){
    napis += "nazwisko\n"
    brakuje_danych = true;
  }
  if (formularz.kraj.value == ""){
    napis += "kraj\n"
    brakuje_danych = true;
  }
  if (formularz.tel.value == ""){
    napis += "telefon\n"
    brakuje_danych = true;
```

```
  }
  if (formularz.ulica.value == "") formularz.ulica.value = "<brak danych>";
  if (formularz.nrdomu.value == "") formularz.nrdomu.value = "<brak danych>";
  if (formularz.kod.value == "") formularz.kod.value = "<brak danych>";
  if (formularz.miasto.value == "") formularz.miasto.value = "<brak danych>";
  if (!brakuje_danych)
    return true;
  else{
    alert ("Nie wypełniłeś następujących pól:\n" + napis);
    return false;
  }
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H2>
<FORM NAME = "formularz1" onSubmit = "return przetwarzaj_dane()">
Prosze podać swoje dane:
<FONT SIZE = "-1"><I><BR>
(Pola oznaczone * muszą zostać wypełnione)
</FONT></I><BR><BR>
<TABLE>
<TR><TD><B>
Dane personalne:</B>
</TD><TD></TD>
<TR><TD>
imię:</TD><TD>
<input type = "text" name = "imie">*</TD>
<TR><TD>
nazwisko:</TD><TD>
<input type = "text" name = "nazwisko">*</TD>
<TR><TD><B>
Adres:</B></TD><TD>
</TD>
<TR><TD>
ulica:</TD><TD>
<input type = "text" name = "ulica"></TD>
<TR><TD>
nr domu:</TD><TD>
<input type = "text" name = "nrdomu"></TD></TR>
<TR><TD>
kod:</TD><TD>
<input type = "text" name = "kod"></TD></TR>
<TR><TD>
miasto:</TD><TD>
<input type = "text" name = "miasto"></TD></TR>
<TR><TD>
kraj:</TD><TD>
<input type = "text" name = "kraj">*</TD></TR>
<TR><TD>
tel.:</TD><TD>
<input type = "text" name = "tel">*</TD></TR>
<TR><TD>
</TD><TD ALIGN = "right">
</H2>
<input type = "submit" name = "wyslij" value = "   Wyślij!   "> </TD>
</TABLE>
</BODY>
</HTML>
```

Mimo, iż kod wygląda podobnie jak w ćwiczeniu 5.10, wykorzystywana jest tu jednak inna technika. Obecnie przycisk wysyłający jest już typu `submit`. Jego naciśnięcie powoduje wywołanie zdarzenia `onSubmit`, któremu przypisana jest odpowiednia procedura obsługi. W przypadku wypełnienia wszystkich żądanych pól zwraca ona wartość `true` i następuje wysłanie zawartości formularza. W przeciwnym przypadku, `onSubmit = false`, wysłanie nie nastąpi. Użytkownik zostanie jednak poinformowany, które pola nie zostały przez niego wypełnione.

Elementy formularzy

Elementami formularza mogą być następujące obiekty:

- `button` — czyli klasyczny przycisk;
- `checkbox` — pola wyboru;
- `hidden` — element ukryty;

- password — pole do wpisywania haseł;
- `radio` — pole wyboru;
- `reset` — przycisk *reset*;
- `select` — lista wyboru;
- `submit` — przycisk *submit*;
- `text` — pole tekstowe;
- `textarea` — rozszerzone pole tekstowe.

Element button

Jest to przycisk umieszczany w dokumencie HTML. Definiujemy go w sposób następujący:

```
<INPUT
 TYPE = ”button”
 NAME = ”nazwa przycisku”
 VALUE = ”wartość na przycisku”
 [onClick = ”obsługa zdarzenia”]
>
```

`Nazwa przycisku` to identyfikator, dzięki któremu możemy się później w prosty sposób do niego odwoływać. Parametr `wartość na przycisku` to tekst, który będzie na nim widoczny. Opcjonalny parametr `obsługa zdarzenia` pozwala nam podać nazwę funkcji, która zostanie wywołana po kliknięciu na tak zdefiniowany przycisk.

Ćwiczenie 5.12.

Wyświetlić na stronie przycisk. Kliknięcie na przycisk powinno spowodować wyświetlenie okna dialogowego.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function przetwarzaj_dane (){
    alert ("Dzięki, bardzo to lubię! :)");
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR><BR>
<FORM NAME = "formularz1">
<INPUT TYPE = "button"
NAME = "przycisk1"
VALUE = "   Kliknij mnie   "
onClick = "przetwarzaj_dane()">
</FORM>
</BODY>
</HTML>
```

Element checkbox

`Checkbox` jest to klasyczne pole wyboru definiowane w sposób następujący:

```
<INPUT
 TYPE = ”checkbox”
 NAME = ”nazwa pola”
 VALUE = ”wartość”
 [CHECKED]
 [onClick = ”obsługa zdarzenia]
>
```

`Nazwa pola` identyfikuje oczywiście obiekt, `wartość` jest wartością zwracaną do serwera podczas przesyłania formularza. Domyślna wartość to `”on”`. Podanie parametru `CHECKED` oznacza, że obiekt ma być domyślnie zaznaczony.

Ćwiczenie 5.13.

Wyświetl na ekranie pole tekstowe umożliwiające wprowadzenie tekstu przez użytkownika. Skrypt ma umożliwić zamianę wszystkich wprowadzonych liter na małe lub wielkie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function imie_przetwarzaj_duze (formularz1){
    if (formularz1.imie_duze.checked){
        formularz1.imie.value = formularz1.imie.value.toUpperCase();
        if (formularz1.imie_male.checked){
           formularz1.imie_male.click();
        }
    }
}
function imie_przetwarzaj_male (formularz1){
    if (formularz1.imie_male.checked){
        formularz1.imie.value = formularz1.imie.value.toLowerCase();
        if (formularz1.imie_duze.checked){
           formularz1.imie_duze.click();
        }
    }
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR>
<BR>
<FORM NAME = "formularz1">
<B> imię: </B>
<INPUT TYPE = "text"
       NAME = "imie"
       SIZE = "15">
<INPUT TYPE = "checkbox"
       NAME = "imie_duze"
       onClick = "imie_przetwarzaj_duze(formularz1)"> duże litery
<INPUT TYPE = "checkbox"
       NAME = "imie_male"
       onClick = "imie_przetwarzaj_male(formularz1)"> małe litery
</FORM>
</BODY>
</HTML>
```

Działanie skryptu jest następujące: tworzymy pole tekstowe o długości 15 znaków. Po wpisaniu do niego dowolnego tekstu możemy, zaznaczając odpowiednie pola wyboru, zamienić go w całości na wielkie, bądź małe litery. Zamiany takiej dokonują funkcje `toUpperCase()` i `toLowerCase()`, zatem wiersz:

```
formularz1.imie.value = formularz1.imie.value.toUpperCase();
```

zamieni wszystkie litery z pola tekstowego „imię" na wielkie litery. Musimy tylko pamiętać o tym, aby uniemożliwić jednoczesne zaznaczenie obu pól wyboru. Zatem po kliknięciu na pole zamiany na duże litery sprawdzamy w funkcji `imie_przetwarzaj_duze()`, czy nie jest przypadkiem zaznaczone pole konwersji na litery małe. Jeśli tak, to usuwamy zaznaczenie tego pola. Ponieważ jednak nie mamy możliwości bezpośredniego ustawienia tej wartości, symulujemy kliknięcie na to pole za pomocą funkcji `click()`. Funkcja `imie_przetwarzaj_male()` dokonuje oczywiście czynności odwrotnej.

Element hidden

Jest to obiekt, którego nie widać w dokumencie HTML, może być on jednak użyty do przechowywania wprowadzonych przez użytkownika wartości. Definicja wygląda w sposób następujący:

```
<INPUT
 TYPE="hidden"
 NAME="nazwa obiektu"
 [VALUE="wartość"]
>
```

`wartość` oznacza tu początkową wartość przypisaną obiektowi.

Ćwiczenie 5.14.

Stwórz formularz umożliwiający wprowadzanie danych tekstowych przez użytkownika. Stosując element `hidden`, zapamiętaj aktualną oraz poprzednio wprowadzoną wartość.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = JavaScript>
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function wprowadz (){
    document.formularz1.ukryta_wartosc.value = document.formularz1.imie.value;
    document.formularz1.imie.value = "";
}
function wyswietl (){
    var poprzednia = document.formularz1.ukryta_wartosc.value;
    var aktualna = document.formularz1.imie.value;
    alert ("poprzednio wprowadzona wartość: " + poprzednia + "\naktualna wartość: " + aktualna);
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR>
<BR>
<FORM NAME = "formularz1">
<INPUT TYPE = "hidden"
       NAME = "ukryta_wartosc"
       VALUE = "nie wprowadzono">
<B> imię: </B>
<INPUT TYPE = "text"
       NAME = "imie"
       SIZE = "15">
<INPUT TYPE = "button"
       NAME = "przycisk1"
       VALUE = "wprowadź"
       onClick = "wprowadz()">
<BR><BR><BR>
<INPUT TYPE = "button"
       NAME = "przycisk2"
       VALUE = "    wyswietl wartości    "
       onClick = "wyswietl()">
</FORM>
</BODY>
</HTML>
```

Element radio

Jest to również pole wyboru, podobnie jak `checkbox`, z ta jednak różnicą, że można je grupować i traktować jako jeden obiekt. W takim przypadku jednocześnie może być zaznaczone tylko jedno pole. Element typu `radio` tworzy się w sposób następujący:

```
<INPUT
 TYPE = ”radio”
 NAME = ”nazwa”
 VALUE = ”wartość”
 [CHECKED]
 [onClick = ”obsługa zdarzenia”]
>
```

Argumenty mają takie samo znaczenie jak w przypadku pól wyboru typu `checkbox`. Ponieważ mamy możliwość grupowania tych elementów w jeden obiekt, otrzymujemy do dyspozycji dodatkowe właściwości:

- `length` — określającą ilość elementów w grupie;
- `index` — określającą numer aktualnie wybranego elementu.

Ćwiczenie 5.15.

Wykonaj zadanie z ćwiczenia 5.13, używając zamiast elementu `checkbox` elementu `radio`.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = JavaScript>
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function konwertuj_na_duze (imie){
    imie.value = imie.value.toUpperCase();
}
function konwertuj_na_male (imie){
    imie.value = imie.value.toLowerCase();
}
```

```
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR>
<BR>
<FORM NAME = "formularz1">
<B> imię: </B>
<INPUT TYPE = "text"
       NAME = "imie"
       SIZE = "15">
<BR>
<BR>
<INPUT TYPE = "radio"
       NAME = "konwersja"
       VALUE = "duze"
       onClick = "konwertuj_na_duze(imie)"> duże litery
<INPUT TYPE = "radio"
       NAME = "konwersja"
       VALUE = "male"
       onClick = "konwertuj_na_male(imie)"> małe litery
<INPUT TYPE = "radio"
       NAME = "konwersja"
       CHECKED> bez konwersji
</FORM>
</BODY>
</HTML>
```

Widać wyraźnie, że kod jest teraz mniej skomplikowany. Głównie dlatego, że nie istnieje już potrzeba obsługi stanów pól wyboru. System sam dba o odpowiednie zaznaczenia.

Element reset

Jest to znany ze stron WWW przycisk, którego wciśnięcie powoduje wyzerowanie formularza. Ściślej przypisanie wszystkim polom ich wartości domyślnych. Definiowany jest w sposób następujący:

```
<INPUT
 TYPE = "reset"
 NAME = "nazwa"
 VALUE = "tekst"
 [onClick = "obsługa zdarzenia"]
>
```

Parametr `tekst` określa, jaki napis będzie widniał na przycisku. Jak widać, można też dołączyć własną procedurę obsługi zdarzenia, tak by np. pewnym polom przypisać wartości inne niż domyślne.

Ćwiczenie 5.16.

Zmodyfikuj przykład z ćwiczenia 5.11, dodając przycisk umożliwiający usunięcia danych z formularza.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = JavaScript>
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptów
function przetwarzaj_dane (){
  var brakuje_danych = false;
  var formularz = document.forms[0];
  var napis = "";
  if (formularz.imie.value == ""){
    napis += "imie\n"
    brakuje_danych = true;
  }
  if (formularz.nazwisko.value == ""){
    napis += "nazwisko\n"
    brakuje_danych = true;
  }
  if (formularz.kraj.value == ""){
    napis += "kraj\n"
    brakuje_danych = true;
  }
  if (formularz.tel.value == ""){
    napis += "telefon\n"
    brakuje_danych = true;
  }
  if (formularz.ulica.value == "") formularz.ulica.value = "<brak danych>";
```

```
  if (formularz.nrdomu.value == "") formularz.nrdomu.value = "<brak danych>";
  if (formularz.kod.value == "") formularz.kod.value = "<brak danych>";
  if (formularz.miasto.value == "") formularz.miasto.value = "<brak danych>";
  if (!brakuje_danych)
    return true;
  else{
    alert ("Nie wypełniłeś następujących pól:\n" + napis);
    return false;
  }
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<H2>
<FORM NAME = "formularz1" onSubmit = "return przetwarzaj_dane()">
Prosze podać swoje dane:
<FONT SIZE = "-1"><I><BR>
(Pola oznaczone * muszą zostać wypełnione)
</FONT></I><BR><BR>
<TABLE>
<TR><TD><B>
Dane personalne:</B>
</TD><TD></TD>
<TR><TD>
imię:</TD><TD>
<input type = "text" name = "imie">*</TD>
<TR><TD>
nazwisko:</TD><TD>
<input type = "text" name = "nazwisko">*</TD>
<TR><TD><B>
Adres:</B></TD><TD>
</TD>
<TR><TD>
ulica:</TD><TD>
<input type = "text" name = "ulica"></TD>
<TR><TD>
nr domu:</TD><TD>
<input type = "text" name = "nrdomu"></TD></TR>
<TR><TD>
kod:</TD><TD>
<input type = "text" name = "kod"></TD></TR>
<TR><TD>
miasto:</TD><TD>
<input type = "text" name = "miasto"></TD></TR>
<TR><TD>
kraj:</TD><TD>
<input type = "text" name = "kraj">*</TD></TR>
<TR><TD>
tel.:</TD><TD>
<input type = "text" name = "tel">*</TD></TR>
<TR><TD ALIGN = "left">
<input type = "reset" name = "wyczysc" value = "  Wyczyść!   ">
</TD><TD ALIGN = "right">
</H2>
<input type = "submit" name = "wyslij" value = "   Wyślij!   "> </TD>
</TABLE>
</BODY>
</HTML>
```

Element select

Element `select` tworzy listę wyboru w formularzu. Definiowany jest w sposób następujący:

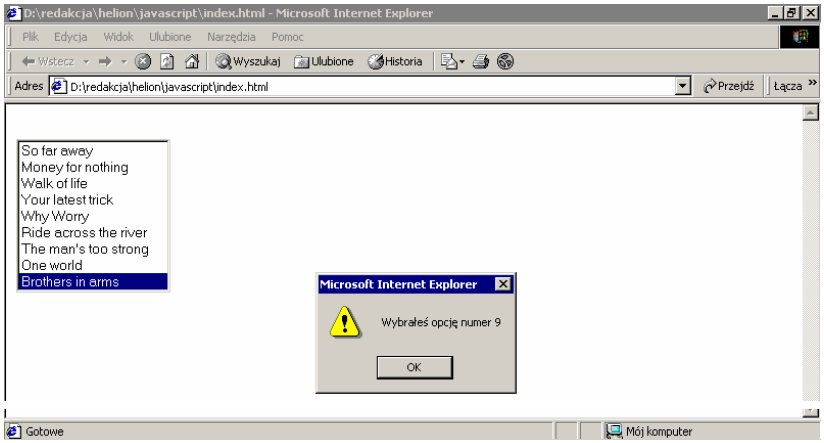
```
<SELECT
 NAME = "nazwa"
 [SIZE = "wielkość"]
 [MULTIPLE]
 [onBlur = "procedura obsługi"]
 [onChange = "procedura obsługi "]
 [onFocus = " procedura obsługi "]>
 <OPTION VALUE = "wartość" [SELECTED]> tekst [ ... <OPTION> tekst]
</SELECT>
```

Parametr `nazwa` specyfikuje nazwę obiektu, `wielkość` jest liczbą pozycji na liście, które mają być wyświetlane. `SIZE = „wielkość”` określa liczbę widocznych elementów listy. `OnBlur`, `onChange` i `onFocus` specyfikują procedury obsługi zdarzeń odpowiednio, gdy obiekt traci `focus`, gdy zostanie wybrana nowa pozycja z listy oraz gdy obiekt otrzymuje `fo-`

cus. Za pomocą znaczników <OPTION> tworzymy natomiast wartości na liście wyboru. Parametr VALUE znacznika OPTION podaje wartość, jaka zostanie zwrócona do serwera po wybraniu danej opcji i wysłaniu formularza. Dodatkowy parametr SELECTED oznacza, że dana pozycja na liście opcji ma być domyślnie zaznaczona. Podanie parametru MULTIPLE powoduje stworzenie przewijanej listy wielokrotnego wyboru.

Ćwiczenie 5.17.

Utwórz i wypełnij przykładowymi danymi listę wyboru. Po kliknięciu na dany element należy wyświetlić jego numer (rysunek 5.5).



Rysunek 5.5. *Obsługa listy wyboru w JavaScript*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = JavaScript>
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function obsluga_zdarzenia (obj){
  alert ("wybrales opcje numer " + (obj.selectedIndex + 1));
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR>
<FORM NAME = "lista">
<SELECT NAME = "songs"
        SIZE = "9"
        MULTIPLE
        onChange = "obsluga_zdarzenia(this)">
  <option> So far away
  <option> Money for nothing
  <option> Walk of life
  <option> Your latest trick
  <option> Why Worry
  <option> Ride across the river
  <option> The man's too strong
  <option> One world
  <option> Brothers in arms
</select>
</FORM>
</BODY>
</HTML>
```

Element text

Element ten służy do wprowadzania przez użytkownika krótkiego ciągu znaków. Tworzy się go w następujący sposób:

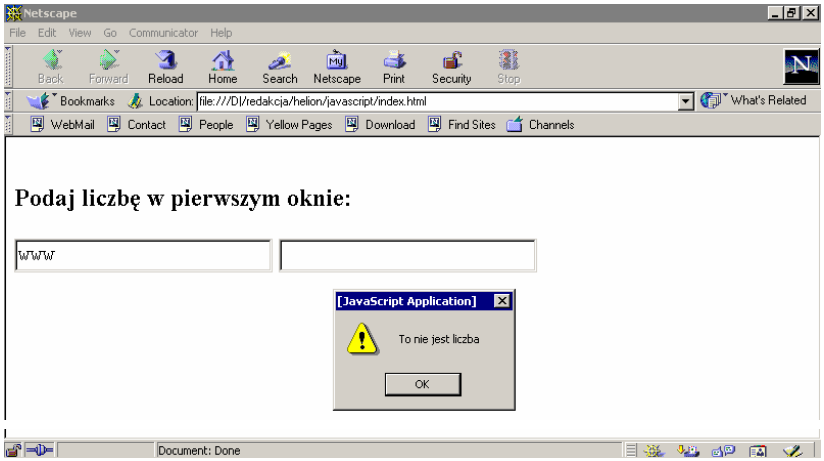
```
<INPUT
 TYPE = ”tekst”
 NAME = ”nazwa obiektu”
 VALUE = ”tekst”
 SIZE = liczba
 [onBlur = ”procedura obsługi”]
 [onChange = ”procedura obsługi”]
 [onFocus = ”procedura obsługi”]
 [onSelect = ”procedura obsługi”]
>
```

Parametr `NAME` specyfikuje nazwę obiektu. `VALUE` jest to wartość początkowa, czyli tekst, który ukaże się domyślnie w oknie tekstowym. `SIZE` to liczba znaków tekstu, jakie można wpisać do okna bez konieczności jego przewijania, czyli po prostu wielkość okna tekstowego. Procedury `onBlur`, `onFocus` i `onChange` działają tak samo, jak w przypadku elementu `select`. `OnSelect` jest procedurą obsługi zdarzenia polegającego na zaznaczeniu fragmentu tekstu. Do dyspozycji mamy też metody `focus`, `blur` i `select`.

Ćwiczenie 5.18.

Wyświetl na ekranie dwa pola testowe oraz prośbę o wpisanie w pierwszym z nich liczby. Jeżeli użytkownik nie poda liczby tylko tekst, po przejściu do drugiego okna ma się wyświetlić ostrzeżenie (rysunek 5.6), a następnie skrypt spowoduje zaznaczenie wprowadzonego tekstu. Należy wreszcie przenieść `focus` do pierwszego okna tekstowego, umożliwiając poprawne wprowadzenie liczby.

Rysunek 5.6.
Efekt działania skryptu z ćwiczenia 5.18




```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = JavaScript>
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function obsluga_zdarzenia (obj){
if (isNaN (obj.value)){
    alert ("To nie jest liczba");
    obj.focus();
    obj.select();
  }
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR>
<BR>
<H2>
Podaj liczbę w pierwszym oknie:
<FORM NAME = "przyklad 69a">
  <INPUT TYPE = "text"
NAME = "okno_tekstowe"
VALUE = ""
onChange = "obsluga_zdarzenia(this)"
  >
  <INPUT TYPE = "text">
</FORM>
</H2>
</BODY>
</HTML>
```

Funkcja `isNaN()` wykorzystana w procedurze obsługi zdarzenia `onChange` sprawdza, czy podany parametr nie jest wartością liczbową (z ang. *NaN — Not A Number*). Jeśli nie jest, zwraca wartość `TRUE`, jeśli jest — `FALSE`. Niestety w przypadku przeglądarki Internet Explorer skrypt może nie zadziałać do końca zgodnie z założeniami, gdyż nie zawsze obsługuje ona poprawnie metody `focus()` i `select()`.

Element textarea

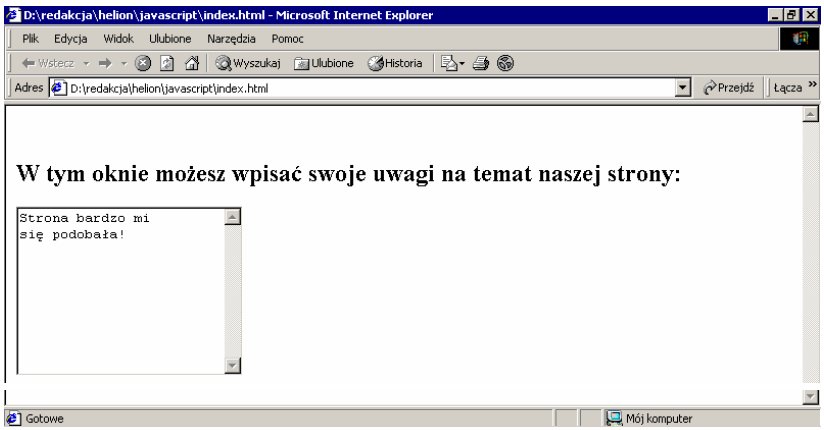
Element tego typu służy do wprowadzania dłuższego tekstu. Definiuje się go w sposób następujący:

```
<TEXTAREA
 NAME = ”nazwa obiektu”
 ROWS = ”liczba rzędów”
 COLS = ”liczba kolumn”
 [onBlur = ”obsługa zdarzenia”]
 [onChange = ”obsługa zdarzenia”]
 [onFocus = ”obsługa zdarzenia”]
 [onSelect = ”obsługa zdarzenia”]>
 tekst
</TEXTAREA>
```

Parametr `NAME` oraz `onBlur`, `onChange`, `onFocus` i `onSelect` mają takie samo znaczenie jak w przypadku poprzednio omawianych elementów. `ROWS` to liczba rzędów, czyli wielkość w pionie (wysokość), `COLS` to liczba kolumn, czyli wielkość w poziomie (długość). `tekst` jest tekstem, który pojawi się jako domyślny.

Ćwiczenie 5.19.

Wyświetl na ekranie element `textarea` służący do wpisania przez użytkownika uwag na temat oglądanej strony (rysunek 5.7).



Rysunek 5.7. *Element textarea umożliwiający wpisanie komentarza na temat strony*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<BODY>
<BR>
<BR>
<H2>
W tym oknie możesz wpisać swoje uwagi na temat naszej strony:
<FORM NAME = "przyklad 70">
  <TEXTAREA
NAME = "okno_tekstowe"
COLS = "25"
ROWS = "10"
  >
Strona bardzo mi
się podobała!
  </TEXTAREA>
</FORM>
</BODY>
</HTML>
```

Wykorzystanie formularzy i zdarzeń

Ćwiczenie 5.20.

Stwórz na stronie WWW kalkulator umożliwiający wykonywanie podstawowych działań arytmetycznych.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptów
var wyrazenie = "";
var nowe = true;
function wprowadz (wartosc){
  var wyswietlacz = document.kalkulator.wyswietlacz;
```

```
  if (nowe){
    nowe = !nowe;
    wyswietlacz.value = "";
  }
  wyswietlacz.value += wartosc;
  wyrazenie += wartosc;
}
function dzialanie (wartosc){
  var wyswietlacz = document.kalkulator.wyswietlacz;
  wyswietlacz.value = "";
  wyrazenie += wartosc;
}
function oblicz (){
  var wyswietlacz = document.kalkulator.wyswietlacz;
  if (wyrazenie != "");{
     wyswietlacz.value = eval (wyrazenie);
     wyrazenie = wyswietlacz.value;
     nowe = true;
  }
}
function zeruj(){
  wyrazenie = "";
  document.kalkulator.wyswietlacz.value = "";
}
function kwadrat(){
  if (wyrazenie != ""){
    document.kalkulator.wyswietlacz.value = wyrazenie * wyrazenie;
    wyrazenie = document.kalkulator.wyswietlacz.value;
    nowe = true;
  }
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR>
<BR>
<FORM NAME = "kalkulator">
<INPUT TYPE = "text"
       NAME = "wyswietlacz"
       SIZE = "13">
<BR>
<BR>
<TABLE BORDER = "0">
<TR><TD>
<INPUT TYPE = "button"
       NAME = "7"
       VALUE = " 7 "
       onClick = "wprowadz(7)">
<INPUT TYPE = "button"
       NAME = "8"
       VALUE = " 8 "
       onClick = "wprowadz(8)">
<INPUT TYPE = "button"
       NAME = "9"
       VALUE = " 9 "
       onClick = "wprowadz(9)">
<INPUT TYPE = "button"
       NAME = "/"
       VALUE = " /  "
       onClick = "dzialanie('/')">
</TD></TR><TR><TD>
<INPUT TYPE = "button"
       NAME = "4"
       VALUE = " 4 "
       onClick = "wprowadz(4)">
<INPUT TYPE = "button"
       NAME = "5"
       VALUE = " 5 "
       onClick = "wprowadz(5)">
<INPUT TYPE = "button"
       NAME = "6"
       VALUE = " 6 "
       onClick = "wprowadz(6)">
<INPUT TYPE = "button"
       NAME = "*"
       VALUE = " * "
       onClick = "dzialanie('*')">
</TD></TR><TR><TD>
<INPUT TYPE = "button"
       NAME = "1"
       VALUE = " 1 "
       onClick = "wprowadz(1)">
<INPUT TYPE = "button"
```

```
        NAME = "2"
        VALUE = " 2 "
        onClick = "wprowadz(2)">
<INPUT TYPE = "button"
        NAME = "3"
        VALUE = " 3 "
        onClick = "wprowadz(3)">
<INPUT TYPE = "button"
        NAME = "-"
        VALUE = " - "
        onClick = "dzialanie('-')">
</TD></TR><TR><TD>
<INPUT TYPE = "button"
        NAME = "0"
        VALUE = " 0 "
        onClick = "wprowadz(0)">
<INPUT TYPE = "button"
        NAME = "!"
        VALUE = "^2"
        onClick = "kwadrat('^2')">
<INPUT TYPE = "button"
        NAME = "c"
        VALUE = " c "
        onClick = "zeruj()">
<INPUT TYPE = "button"
        NAME = "+"
        VALUE = " + "
        onClick = "dzialanie('+')">
</TD></TR><TR><TD>
<INPUT TYPE = "button"
        NAME = "="
        VALUE = "        =        "
        onClick = "oblicz()">
</TR></TD>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Wygląd kalkulatora ilustruje rysunek 5.8. Nie jest on może bardzo urodziwy, niemniej proste obliczenia można na nim wykonać.

Rysunek 5.8. *Kalkulator stworzony przy pomocy elementów formularzy*

Wszystkie przyciski są wykonane z obiektów `button`, które zgrupowane są po cztery w kolejnych rzędach tabeli użytej do sformatowania. Niestety, obiekt `button` nie ma parametru, w którym można by podać jego wielkość, tak więc rozmiar przycisków jest zależny od tekstów na nich się znajdujących. Co prawda, cyfry mają stały rozmiar, jednak pozostałe użyte znaki nie. Dlatego też całość jest nieco nierówna. Można spróbować to poprawić, umieszczając każdy przycisk w oddzielnej komórce tabeli, jednak nie zawsze daje to pożądany efekt.

Każdy przycisk z cyfrą ma przypisaną procedurę zdarzenia `wprowadz()`. Procedura ta najpierw sprawdza, czy rozpoczynamy nowe działanie, czy też wprowadzamy dalszy ciąg liczby wielocyfrowej. W pierwszym przypadku zerowana jest zmienna `wyrazenie`, która przechowuje nasze działanie oraz wyświetlacz, którym jest poznany dzisiaj obiekt typu `text`.

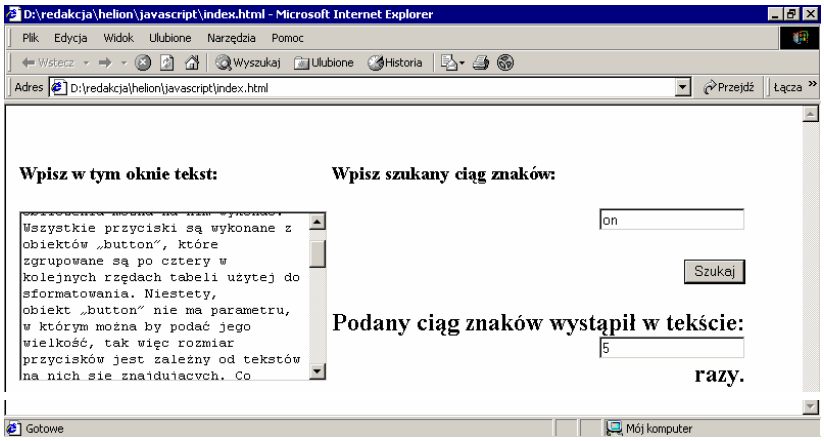
W przypadku drugim (nie rozpoczynamy nowego działania) do zmiennej `wyrażenie` oraz na wyświetlaczu dopisywana jest kolejna cyfra. Jeśli naciśnięty zostanie przycisk jednego z działań: dodawanie, odejmowanie, mnożenie lub dzielenie, wykonywana jest funkcja `działanie()`. Działa w sposób podobny do funkcji `wprowadz()`, z tą różnicą, że nie sprawdza, czy rozpoczynamy nowe działanie. Nie ma takiej potrzeby, jako że na pewno jesteśmy w jego trakcie.

Przycisk = ma przypisaną funkcję `oblicz()`. Sprawdza ona, czy zmienna `wyrażenie` zawiera jakiś łańcuch znaków. Jeśli nie, żadna czynność nie jest wykonywana. W przeciwnym przypadku wywoływana jest funkcja `eval()` ze zmienną `wyrazenie` jako parametrem. Funkcja ta oblicza wyrażenie arytmetyczne, zwraca więc wynik wprowadzonego do zmiennej `wyrazenie` działania. Funkcje `zeruj()` i `kwadrat()` nie wymagają chyba bliższego wyjaśniania.

Ćwiczenie 5.21.

Utwórz jedno okno tekstowe umożliwiające wpisanie dowolnego dłuższego tekstu oraz mniejsze służące do wpisania poszukiwanego ciągu znaków. Zdefiniuj przycisk *Szukaj*, którego kliknięcie spowoduje obliczenie ilość wystąpień szukanego ciągu znaków we wprowadzonym tekście (rysunek 5.9).

Rysunek 5.9. *Obliczanie ilości wystąpień podanego ciągu znaków we wprowadzonym tekście*




```
<HTML>
<HEAD>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function przetwarzaj (f){
  var tekst = f.okno_tekstowe.value;
  var ciag = f.ciag_znakow.value;
  var indeks = 0;
  var indeks_wystapienia = 0;
  var liczba_wystapien = 0;
  while (indeks_wystapienia != -1){
    indeks_wystapienia = tekst.indexOf (ciag, indeks);
    if{
        indeks += indeks_wystapienia;
        liczba_wystapien++;
    }
  }
  f.ile.value = liczba_wystapien;
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR>
<BR>
<TABLE BORDER = "0">
<TR><TD>
<H3>Wpisz w tym oknie tekst:</H3>
</TD><TD>
<H3>Wpisz szukany ciąg znaków:</H3>
</TD></TR>
<TR><TD ROWSPAN = "2">
<H2>
<FORM NAME = "formularz">
  <TEXTAREA
   NAME = "okno_tekstowe"
   COLS = "35"
   ROWS = "10"
  >
</TEXTAREA>
</H2></TD><TD VALIGN = "top" ALIGN = "right"><H2>
<INPUT TYPE = "text" NAME = "ciag_znakow"><BR>
<BR><INPUT
     TYPE = "button"
     VALUE = "Szukaj"
     onClick = "przetwarzaj(formularz)"
    >
<H2></TD></TR>
<TR><TD ALIGN = "right"><H2>
Podany ciąg znaków wystąpił w tekście:<BR>
<INPUT TYPE = "text" NAME = "ile"><BR>
razy.
</H2></TD></TR>
</FORM>
</TABLE>
</BODY>
</HTML>
```

Stwórz skrypt, który umożliwi użytkownikowi wprowadzenie do formularza imienia i nazwiska. Zamień pierwsze litery obu wyrazów na wielkie, a wszystkie pozostałe na małe, niezależnie od tego, w jaki sposób zostały podane przez użytkownika.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function przetwarzaj (f){
  var nazwisko = f.nazwisko.value;
  var imie = f.imie.value;
  f.nazwisko.value = nazwisko.substring (0, 1).toUpperCase() + nazwisko.substring
(1, nazwisko.length).toLowerCase();
  f.imie.value = imie.substring (0, 1).toUpperCase() + imie.substring
(1, imie.length).toLowerCase();
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR><BR>
<FORM NAME = "formularz1">
<TABLE BORDER = "0">
<TR><TD><H2>
 imię:
</H2></TD><TD><H2>
<INPUT TYPE = "text"
       NAME = "imie"
       SIZE = "15">
</H2></TD></TR>
<TR><TD><H2>
nazwisko:
</H2></TD><TD><H2>
<INPUT TYPE = "text"
       NAME = "nazwisko"
       SIZE = "15">
</H2></TD></TR>
<TR><TD COLSPAN = "2" ALIGN = "right"><H2>
<INPUT TYPE = "button"
       NAME = "zamien"
       VALUE = "zamień"
       onClick = "przetwarzaj (formularz1)"
>
</H2></TD></TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Rozdział 6.

Okna, ramki i ciasteczka

Okna

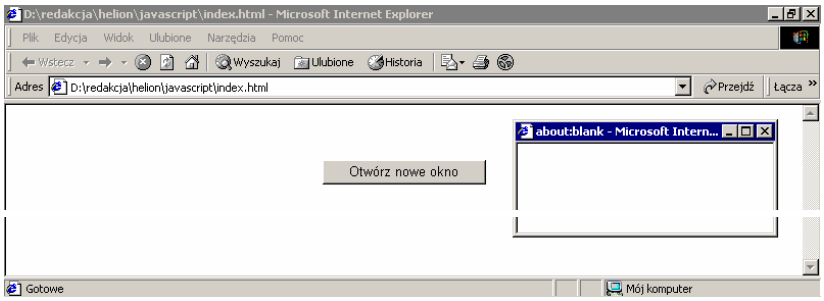
Manipulację oknami umożliwia nam obiekt `window`. Jego właściwości i metody przedstawione zostały w rozdziale czwartym. Nie zostały jednak wykorzystane wtedy wszystkie jego możliwości. Obiekt ten udostępnia m.in. metodę `open` pozwalającą na otworzenie nowego okna przeglądarki. Metodę tę należy wywoływać w sposób następujący:

```
zmienna = window.open ("URL", "nazwa okna", ["właściwości okna"])
```

`zmienna` — jest to zmienna identyfikująca nowo powstałe okno, `URL` — url dokumentu, który ma zostać załadowany, `nazwa okna` — parametr, który będzie używany do identyfikacji okna przy odwoływaniu się do niego za pomocą parametru `TARGET` znaczników HTML `<FRAME>` i `<A>`, `właściwości okna` — opcjonalna lista argumentów określających wygląd okna.

Ćwiczenie 6.1.

Napisz skrypt otwierający nowe okno z pustą zawartością po wciśnięciu przycisku na bieżącej stronie (rysunek 6.1).

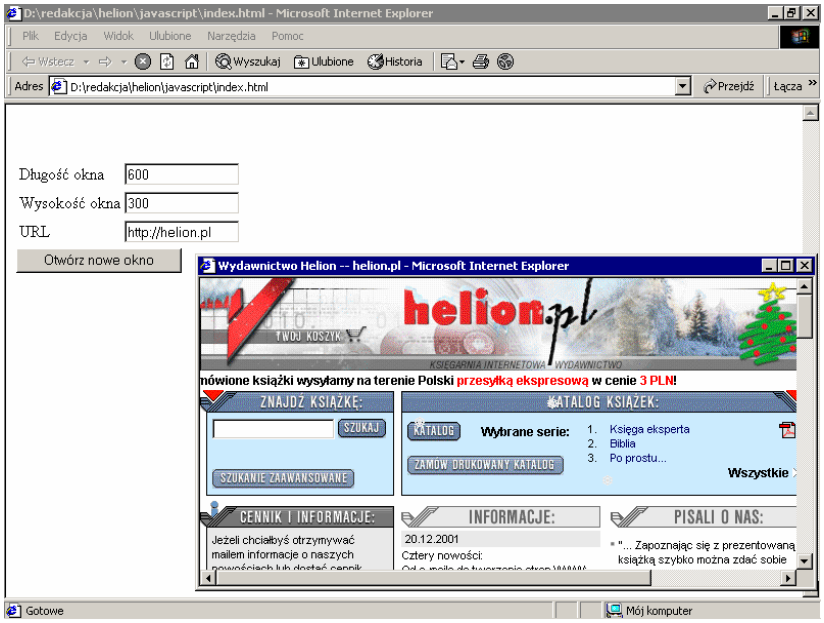


Rysunek 6.1. *Skrypt otwierający nowe okno przeglądarki po wciśnięciu przycisku*

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</HEAD>
<BODY>
<BR><BR>
<H2><CENTER>
<FORM NAME = "formularz1">
<INPUT TYPE = "button"
       NAME = "nowe_okno"
       VALUE = "Otwórz nowe okno"
       onClick = "window2=open('about:blank','secondWindow','scrollbars=no,width=250,height=400')"
>
</FORM>
</H2></CENTER>
</BODY>
</HTML>
```

Napisz skrypt otwierający nowe okno o zadanych przez użytkownika rozmiarach i zawartości (rysunek 6.2).

Rysunek 6.2. *Otwarcie okna o zadanych przez użytkownika parametrach*




```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE = "JavaScript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptów
function otworzOkno(){
  var dlugosc = document.forms.formularz1.dlugosc.value;
  var wysokosc = document.forms.formularz1.wysokosc.value;
  var url = document.forms.formularz1.url.value;
  parametry = 'scrollbars=yes,width=' + dlugosc + ',height=' + wysokosc;
  window2=open(url,'secondWindow',parametry);
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<BR><BR>
<H2>
<FORM NAME = "formularz1">
<TABLE><TR><TD>
Długość okna</TD><TD>
<INPUT TYPE = "text"
       NAME = "dlugosc"
       SIZE = "15">
</TD></TR><TR><TD>
Wysokość okna</TD><TD>
<INPUT TYPE = "text"
       NAME = "wysokosc"
       SIZE = "15">
</TD></TR><TR><TD>
URL</TD><TD>
<INPUT TYPE = "text"
       NAME = "url"
       SIZE = "15">
</TD></TR></TABLE>
<INPUT TYPE = "button"
       NAME = "nowe_okno"
       VALUE = "Otwórz nowe okno"
       onClick = "otworzOkno();"
>
</FORM>
</H2>
</BODY>
</HTML>
```

Obiekt `window` zawiera również dwie metody związane z licznikiem czasu, mianowicie `setTimeout()` i `clearTimeout()`. Pierwszą z nich używamy w sposób następujący:

```
identyfikator = setTimeout (wyrażenie, n)
```

`Identyfikator` to zmienna używana, gdy chcemy przerwać liczenie czasu — wyrażenie zostanie opracowane po upływie czasu `n`, przy czym `n` podajemy w milisekundach. Opracowanie wyrażenia może oznaczać np. wywołanie funkcji. Prześledzimy to na konkretnym przykładzie.

Druga metoda, `clearTimeout()`, przerywająca liczenie czasu zapoczątkowane metodą `setTimeout()`, wywoływana jest następująco:

```
clearTimeout (identyfikator)
```

Identyfikator jest w tym przypadku zmienną zwróconą wcześniej przez `setTimeout()`.

Ćwiczenie 6.3.

Napisz skrypt, który po wciśnięciu przycisku zacznie odmierzać czas. Po upływie 3 sekund należy wyświetlić okno dialogowe z informacją o tym fakcie.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE="JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function wyswietl() {
   alert("Upłynęły trzy sekundy o wciśnięcia przycisku START")
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<H2>
Wciśnij przycisk start, aby zapoczątkować liczenie czasu. Po upływie trzech sekund zostanie
wyświetlone okno dialogowe..
<P>
<INPUT TYPE="button" VALUE="S T A R T"
   NAME="start"
   onClick="identyfikator=setTimeout('wyswietl()',3000)">
</FORM>
</BODY>
</HTML>
```

Skrypt działa w sposób następujący. Jeśli wciśniemy przycisk *Start*, rozpocznie się odliczanie 3 sekund (3 000 milisekund). Po jego upływie zostanie wykonana funkcja `wyświetl()` pokazująca na ekranie okno dialogowe. Takie zachowanie zapewnia przypisanie procedurze `onClick()` związanej z przyciskiem *Start* wyrażenia `setTimeout('wyswietl ()',3000)`.

Ćwiczenie 6.4.

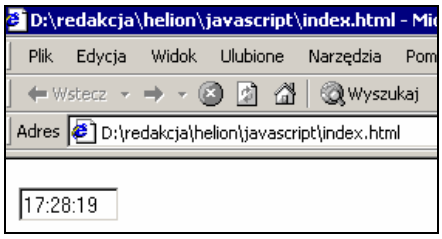
Zmodyfikować przykład z ćwiczenie 6.3 w taki sposób, aby użytkownik miał możliwość zatrzymania licznika czasu.

```
<HTML>
<HEAD>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
<SCRIPT LANGUAGE="JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function wyswietl() {
   alert("Upłynęły trzy sekundy o wciśnięcia przycisku START")
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<H2>
Wciśnij przycisk start, aby zapoczątkować liczenie czasu. Po upływie trzech sekund zostanie
wyświetlone okno dialogowe. Jeśli chcesz przerwać liczenie wciśnij przycisk stop.
<P>
<INPUT TYPE="button" VALUE="S T A R T"
   NAME="start"
   onClick="identyfikator=setTimeout('wyswietl()',3000)">
<INPUT TYPE="button" VALUE="S T O P"
   NAME="stop"
   onClick="clearTimeout(identyfikator)">
</FORM>
</BODY>
</HTML>
```

Ćwiczenie 6.5.

Stwórz zegar wyświetlający na stronie WWW aktualny czas (rysunek 6.3).

Rysunek 6.3.
Zegar na stronie WWW




```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptów
var timerID = null
function wyswietlCzas(){
    var data = new Date();
    var godziny = data.getHours();
    var minuty = data.getMinutes();
    var sekundy = data.getSeconds();
    var czas = godziny;
    czas += ((minuty < 10) ? ":0" : ":") + minuty;
    czas += ((sekundy < 10) ? ":0" : ":") + sekundy;
    document.zegar.wyswietlacz.value = czas;
    timerID = setTimeout("wyswietlCzas()",1000);
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY onLoad="wyswietlCzas()">
<H2>
<FORM NAME="zegar">
    <INPUT TYPE="text"
     NAME="wyswietlacz"
     SIZE="7"
     VALUE =""">
</FORM>
</BODY>
</HTML>
```

W skrypcie tym wykorzystany został obiekt `Date` wraz z jego metodami, przedstawionymi we wcześniejszych rozdziałach. Po załadowaniu strony wykonywana jest po raz pierwszy funkcja `wyswietlCzas()`, która startuje zegar. Następnie dzięki wierszowi `timerID = setTimeout(„wyswietlCzas()”,1000);` metoda ta jest co sekundę wykonywana ponownie. Konstrukcja warunkowa związana z liczbą minut i sekund jest konieczna, aby na wyświetlaczu była zawsze taka sama ilość pozycji. To znaczy, jeśli liczba minut lub sekund jest mniejsza niż 10, trzeba na początku dodać zero, inaczej zamiast np. 10:09:53 otrzymalibyśmy 10:9:53, co oczywiście byłoby efektem niepożądanym.

Ramki

Ramki pozwalają na podzielenie strony w HTML-u na niezależne części. Często spotykanym zastosowaniem jest stworzenie spisu treści strony WWW, kiedy to tytuły dokumentów pojawiają się w jednym oknie, a ich treść w drugim. Do stworzenia ramek służy polecenie `<FRAMESET>`, które przyjmuje następujące parametry:

```
<FRAMESET
  ROWS = ”liczba wierszy”
  COLS = ”liczba kolumn”
  [onLoad = ”procedura obsługi zdarzenia”]
  [onUnload = ”procedura obsługi zdarzenia”]>
  [<FRAME SRC = ”URL” NAME = ”nazwa ramki”>]
</FRAMESET>
```

Ogólna struktura dokumentu z ramkami jest następująca:

```
<HTML>
<HEAD>
<TITLE>
Tytuł strony
```

```
</TITLE>
Treść nagłówka
</HEAD>
<FRAMESET>
Treść dokumentu dla przeglądarek obsługujących ramki
</FRAMESET>
<NOFRAMES>
<BODY>
Treść dokumentu dla przeglądarek nieobsługujących ramek
</BODY>
</NOFRAMES>
</HTML>
```

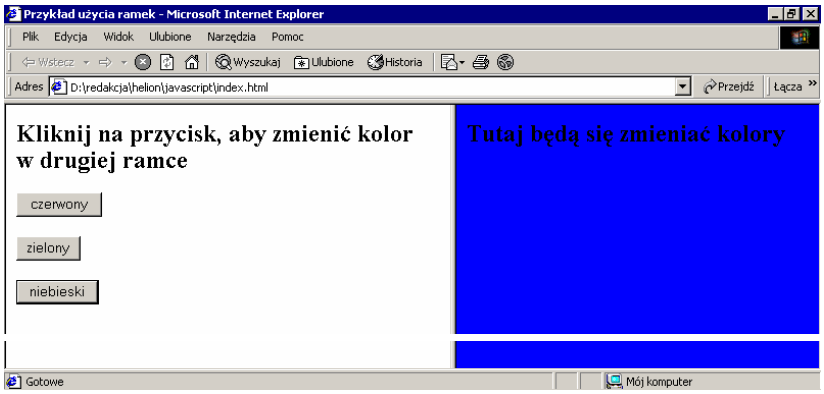
Znacznik `<FRAME>` służy do definiowania zachowania i zawartości każdej z ramek. Możliwe jest użycie następujących parametrów:

- `SRC = „nazwa.html”` — definiuje dokument HTML, który znajdzie się w ramce;
- `SCROLLING = „wartość”` — podaje, czy ramka ma mieć możliwość przesuwania zawartości, o ile nie mieści się ona w oknie. Możliwe parametry to: `YES` — ramka będzie miała suwak do przewijania zawartości, `NO` — nie będzie suwaka, `AUTO` — suwaki pojawią się, jeżeli dokument nie będzie mieścił się w ramce;
- `NORESIZE` — podanie tego parametru uniemożliwi zmianę wielkości ramki. Domyślnie użytkownik może dowolnie manipulować wielkością ramki przesuwając jej brzegi przy pomocy myszy,
- `MARGINHEIGHT = n` — określa odległość zawartości ramki od górnego i dolnego brzegu (wartość n podawana w pikselach);
- `MARGINWIDTH = n` — określa odległość zawartości ramki od lewego i prawego brzegu (wartość n podawana w pikselach);
- `NAME` — to nazwa, która identyfikuje dana ramkę, dzięki czemu możemy się do niej w prosty sposób odwoływać.

Te wiadomości powinny w zupełności wystarczyć do wykonania kolejnego ćwiczenia.

Ćwiczenie 6.6.

Utwórz dokument z ramkami. W ramce pierwszej umieść przyciski umożliwiające zmianę koloru tła w ramce drugiej.



Rysunek 6.4. *Skrypt ilustrujący manipulację ramkami w dokumencie*

Plik *index.html*

```
<HTML>
<HEAD>
<TITLE>Przykład użycia ramek</TITLE>
<HEAD>
</HEAD>
<FRAMESET>
<FRAMESET COLS="*, 45%">
<FRAME SRC= ramka1.html NAME = "ramka1">
<FRAME SRC= ramka2.html NAME = "ramka2">
</FRAMESET>
<NOFRAMES>
<BODY>
Twoja przeglądarka nie obsługuje ramek!
</BODY>
</NOFRAMES>
</HTML>
```

Plik *ramka1.html*

```
<HTML>
<TITLE>Ramka1</TITLE>
<HEAD>
<SCRIPT LANGUAGE = "JavaScript">
function zmienKolor (kolor){
parent.ramka2.document.bgColor = kolor;
}
</SCRIPT>
</HEAD>
<BODY>
<H2>Kliknij na przycisk, aby zmienić kolor w drugiej ramce</H2>
<FORM>
<INPUT
    TYPE = "button"
    NAME = "przycisk1"
    VALUE = "czerwony"
    onClick = "zmienKolor ('red')";
>
<BR><BR>
<INPUT
    TYPE = "button"
    NAME = "przycisk2"
    VALUE = "zielony"
    onClick = "zmienKolor ('green')";
>
<BR><BR>
<INPUT
    TYPE = "button"
    NAME = "przycisk3"
    VALUE = "niebieski"
    onClick = "zmienKolor ('blue')";
>
</BODY>
</HTML>
```

Plik *Ramka2.html*

```
<HTML>
<TITLE>Ramka1</TITLE>
<HEAD></HEAD>
<BODY>
<H2>Tutaj będą się zmieniać kolory</H2>
</BODY>
</HTML>
```

Wszystkie konstrukcje programowe użyte w powyższym ćwiczeniu były już omawiane, więc nie trzeba ich chyba dodatkowo wyjaśniać. Uwagę zwraca tylko sposób odwołania się do właściwości `gbColor` w drugiej ramce. Otóż gdyby napisać `window.ramka2.document.bgColor`, powstałby oczywiście błąd, jako że `window` odnosi się do bieżącego okna. Oknem bieżącym jest natomiast `ramka1`, która „nic nie wie" o tym, że istnieje jeszcze `ramka2`. Należy więc odwołać się do rodzica (`parent`), który został stworzony przez `index.html` i „wie" zarówno o `ramce1`, jak i `ramce2` (`parent.ramka2.document.bgColor = kolor`).

Ciasteczka, czyli cookies

Cookies są to małe porcje informacji, które mogą być przesyłane między serwerem a przeglądarką. Zwykle odbywa się to w następujący sposób. Przy pierwszym połączeniu z daną stroną serwer wysyła ciasteczko z jakąś informacją do przeglądarki. Ta zapisuje je na dysku. Po ponownym połączeniu z tą stroną przeglądarka odsyła zapamiętaną informację do serwera. Jest to sposób na stwierdzenie, czy użytkownik był już na naszej stronie, bądź też, jakie czynności na niej wykonywał. *Cookie* przesyłane jest w nagłówku HTTP w postaci pola:

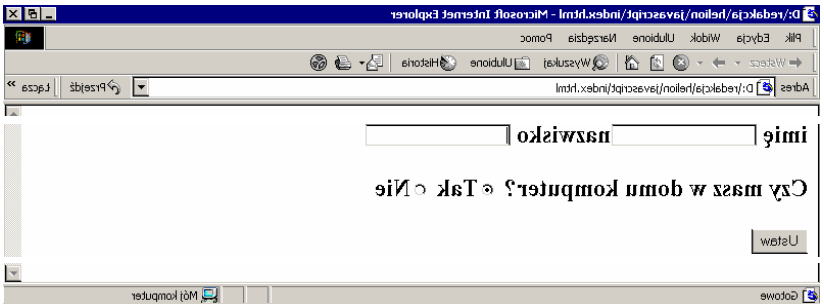
```
Set-Cookie: nazwa_parametru = wartość; expires = data; path = ścieżka; domain =
domena; secure
```

Interesują nas w tej chwili dwa pola: `nazwa_parametru = wartość` oraz `expires = data`. Pierwsze z nich pozwala zapamiętać jakąś informację, np. podane przez użytkownika jego imię w postaci `imie = Jarek`. Drugie określa ważność ciasteczka, tzn. datę, po której zostanie usunięte z systemu. Data ta musi zostać podana w formacie GMT, np. „Thu, 01 Jul 2002 12:00:00 GMT". Można do jej ustalenia wykorzystać obiekt `Date` i udostępniane przez niego konwersje.

Ćwiczenie 6.7.

Przygotuj formularz, w którym użytkownik będzie mógł podać swoje dane, imię, nazwisko oraz informację o tym, czy posiada komputer (rysunek 6.5). Po kliknięciu przycisku *Ustaw* należy zapisać dane w trzech ciasteczkach. Przy ponownym wczytaniu skryptu formularz powinien się wypełnić uprzednio zapamiętanymi danymi.

Rysunek 6.5. *Skrypt wysyłający do przeglądarki użytkownika cookies*



```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- Ukrycie przed przeglądarkami nieobsługującymi JavaScript
function setCookie(){
  var expirationDate = "Thu, 01 Jul 2002 12:00:00 GMT";
  var imie = document.form1.imie.value;
  var nazwisko = document.form1.nazwisko.value;
  var komputer = "undefined"
  if (document.form1.komputer[0].checked)
    komputer = "tak";
  if (document.form1.komputer[1].checked)
    komputer = "nie";
  document.cookie = "imie =" + imie + "; expires = " + expirationDate;
  document.cookie = "nazwisko =" + nazwisko + "; expires = " + expirationDate;
  document.cookie = "komputer =" + komputer + "; expires = " + expirationDate;
}
function checkCookie(searchString){
  var c = document.cookie;
  point = c.indexOf (searchString);
  if (point != -1){
    endAt = c.indexOf (";", point + searchString.length);
    if (endAt == -1) endAt = c.length;
    var temp = c.substring (point + searchString.length, endAt);
    return temp;
  }
  return false;
}
function getCookie(){
  tempString = checkCookie ("imie=");
  if (tempString)
    document.form1.imie.value = tempString;
  tempString = checkCookie ("nazwisko=");
  if (tempString)
    document.form1.nazwisko.value = tempString;
  tempString = checkCookie ("komputer=");
  if (tempString){
    if (tempString == "tak")
      document.form1.komputer[0].checked = true;
    if (tempString == "nie")
      document.form1.komputer[1].checked = true;
  }
}
// Koniec kodu JavaScript -->
</SCRIPT>
</HEAD>
<BODY onLoad = "getCookie()">
<H2><BR><BR>
<FORM NAME = "form1">
imię
<INPUT TYPE = "text"
       NAME = "imie"
>nazwisko
<INPUT TYPE = "text"
       NAME = "nazwisko"
>
<BR><BR>
Czy masz w domu komputer?
<INPUT TYPE = "radio"
       NAME = "komputer"
       VALUE = "tak"
>Tak
<INPUT TYPE = "radio"
       NAME = "komputer"
       VALUE = "nie"
>Nie
```

```
<BR><BR>
<INPUT TYPE = "button"
       VALUE = "Ustaw"
       onClick = "setCookie()"
>
<FORM>
</BODY>
</HTML>
```

Formularz jest tworzony w sposób standardowy dla języka HTML. Dla przycisku ustawiana jest obsługa zdarzenia — kliknięcia na funkcję `setCookie()`. W funkcji tej ustawiane są zmienne określające datę wygaśnięcia ważności ciasteczka oraz imię i nazwisko użytkownika. Dwie ostatnie dane pobierane są z formularza. Jeśli użytkownik nie wypełni formularza, są im przypisywane puste ciągi znaków. Zmienna `komputer` przyjmie wartość `tak` lub `nie`, w zależności od tego, które pole wyboru zostało zaznaczone. Jeśli żadne nie zostało zaznaczone, wartością tej zmiennej będzie „`undefined`". Dalej ustawiane są trzy ciasteczka, co robimy przypisując własności `cookie` obiektu `document` określone wcześniej ciągi znaków.

Nieco bardziej skomplikowany jest sposób odczytu ciasteczek. Dokonujemy tego przy użyciu funkcji `getCookie()`, która jest wywoływana podczas ładowania strony oraz funkcji pomocniczej `checkCookie()`. Ponieważ `getCookie()` stosuje wielokrotnie już używane przez nas konstrukcje, nie trzeba jej chyba bliżej wyjaśniać. Problem może natomiast sprawić `checkCookie()`. Jakie ona ma zadanie? Otóż we własności `document.cookie` znajdziemy wszystkie ciasteczka odnoszące się do danej strony w postaci ciągu znaków np.:

```
zmienna1=brak;imie=Marek;zmienna3=154;nazwisko=Kowalski;komputer=tak
```

Z tego ciągu znaków musimy po pierwsze wyciągnąć interesujące nas zmienne, a po drugie dostać się do ich wartości. Nasza funkcja jako parametr otrzymuje szukany ciąg znaków. Sprawdza, czy występuje on we własności `document.cookie` oraz zapamiętuje miejsce jego wystąpienia. Następnie szuka końca wartości tej opcji, czyli znaku „;". W przypadku gdyby szukana wartość była ostatnią w ciągu, tzn. na jej końcu nie występowałby znak `;` zastosowana funkcja szukająca `indexOf()` zwróci wartość `-1`. Zatem taką sytuację też możemy rozpoznać. Kiedy mamy już indeks wystąpienia szukanej zmiennej, długość jej nazwy oraz całkowitą długość opisującego ją łańcucha znaków, możemy już w prosty sposób (przy użyciu funkcji `substring()`) dostać się do wartości tej zmiennej. Na koniec zwracamy znalezioną wartość, którą funkcja `getCookie()` przypisze właściwemu elementowi formularza.