helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

JavaScript. Ćwiczenia praktyczne. Wydanie II

Autor: Marcin Lis
ISBN: 83-246-0795-1
Format: A5, stron: 160
Przykłady na ftp: 52 kB

Ożyw swoje witryny WWW

- Poznaj elementy języka JavaScript
- Napisz procedury obsługi zdarzeń
- Stwórz mechanizmy obsługi formularzy HTML

HTML, mimo ciągłego rozwoju, pozostaje wyłącznie językiem służącym do formatowania dokumentów. Przetwarzanie danych wprowadzanych przez użytkowników witryny WWW realizuje się za pomocą innych mechanizmów. Jedną z technologii służących do wykonywania takich operacji jest JavaScript – język skryptowy interpretowany po stronie przeglądarki, opracowany przez firmę Netscape. JavaScript umożliwia tworzenie i umieszczanie bezpośrednio w kodzie HTML krótkich programów, za pomocą których można wykonywać różne zadania, takie jak rozpoznawanie i obsługiwanie kliknięć myszą, weryfikacja danych wprowadzanych do formularzy czy też nawigowanie pomiędzy stronami. Ma prostą składnię i jest stosunkowo łatwy do opanowania.

Czytając książkę „JavaScript. Ćwiczenia praktyczne. Wydanie II" i wykonując zawarte w niej przykłady, poznasz podstawy tego języka. Dowiesz się, z jakich elementów składa się JavaScript i w jaki sposób umieszczać jego kod w dokumentach HTML. Nauczysz się tworzyć funkcje i korzystać z obiektów. Przeczytasz o tym, w jaki sposób JavaScript może współpracować z przeglądarką internetową, przetwarzać zdarzenia generowane przez użytkownika i interpretować dane pochodzące z formularzy umieszczonych na stronie WWW.

- Umieszczanie skryptów w dokumencie
- Wyświetlanie okien dialogowych
- Typy danych, zmienne i operatory
- Pętle i konstrukcje warunkowe
- Współpraca z przeglądarką
- Obsługa zdarzeń
- Weryfikacja danych z formularzy

Naucz się korzystać z języka JavaScript, który jest podstawą wielu nowoczesnych technologii sieciowych

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

Spis treści

Wstęp 5

Rozdział 1. Podstawy 9

Umieszczanie skryptów w dokumencie 9
Instrukcja document.write 11
Formatowanie tekstu 13
Wyświetlenie okna dialogowego 16
Jeśli przeglądarka nie obsługuje skryptów 17
Komentarze 18

Rozdział 2. Elementy języka 21

Typy danych 21
Zmienne 23
Operatory 26
Instrukcje warunkowe 36
Pętle 43
Funkcje 50
Zasięg zmiennych 55

Rozdział 3. Obiekty i funkcje globalne 59

Funkcje globalne 59
Obiekty dostępne standardowo 65

Rozdział 4. Współpraca z przeglądarką 87

Obiekty 87
Obiekt window 88
Obiekt document 105
Obiekt history 112

Obiekt location 113
Obiekt navigator 117

Rozdział 5. Zdarzenia 123

Zdarzenia 123
Zdarzenia onload i onunload 126
Zdarzenia związane z myszą 129

Rozdział 6. Obsługa formularzy 139

Obiekty formularza 139
Element button (przycisk) 142
Element checkbox (pole wyboru) 144
Element radio (pole wyboru) 146
Element text (pole tekstowe) 148
Element textarea (rozszerzone pole tekstowe) 150
Element list (lista wyboru) 153
Walidacja formularzy 155

Elementy języka

Typy danych

Występujące w JavaScript typy danych można podzielić następująco:

- typ liczbowy,
- typ łańcuchowy,
- typ logiczny,
- typ **null**,
- typ obiektowy.

Typ liczbowy

Typ liczbowy służy do reprezentacji liczb, przy czym nie ma występującego w klasycznych językach programowania rozróżnienia na typy całkowitoliczbowe i rzeczywiste (zmiennopozycyjne). Liczby zapisywane są za pomocą literałów (inaczej stałych napisowych, z ang. *string constant*, *literal constant*) liczbowych. Obowiązują przy tym następujące zasady:

- Jeżeli ciąg cyfr nie jest poprzedzony żadnym znakiem lub jest poprzedzony znakiem +, reprezentuje on wartość dodatnią, jeżeli natomiast jest poprzedzony znakiem –, reprezentuje wartość ujemną.

- Jeżeli literał rozpoczyna się od cyfry zero, jest traktowany jako wartość ósemkowa.
- Jeżeli literał rozpoczyna się od ciągu znaków `0x`, jest traktowany jako wartość szesnastkowa (heksadecymalna). W zapisie wartości szesnastkowych mogą być wykorzystywane zarówno małe, jak i wielkie litery alfabetu od *A* do *F*.
- Literały mogą być zapisywane w notacji naukowej, w postaci `X.YeZ`, gdzie `X` to część całkowita, `Y` część dziesiętna, natomiast `Z` to wykładnik potęgi liczby 10. Zapis taki oznacza to samo co $X.Y * 10^Z$.

Przykłady literałów:

`123`	dodatnia całkowita wartość dziesiętna 123
`-123`	ujemna całkowita wartość dziesiętna –123
`012`	dodatnia całkowita wartość ósemkowa równa 10 dziesiętnie
`-024`	ujemna całkowita wartość ósemkowa równa 20 dziesiętnie
`0xFF`	dodatnia całkowita wartość szesnastkowa równa 255 dziesiętnie
`-0x0f`	ujemna całkowita wartość szesnastkowa równa –15 dziesiętnie
`1.1`	dodatnia wartość rzeczywista 1.1
`-1.1`	ujemna wartość rzeczywista –1.1
`0.1E2`	dodatnia wartość rzeczywista równa 10
`1.0E-2`	dodatnia wartość rzeczywista równa 0.01

Typ łańcuchowy

Typ łańcuchowy służy do reprezentacji ciągów znaków (napisów). Ciągi te (inaczej stałe napisowe) powinny być ujęte w znaki cudzysłowu, aczkolwiek dopuszczalne jest również wykorzystanie znaków apostrofu. Przykładowy ciąg ma postać:

```
"abcdefg"
```

Mogą one też zawierać sekwencje znaków specjalnych przedstawione w tabeli 1.1 w rozdziale 1.

Typ logiczny

Typ logiczny (*boolean*) pozwala na określenie dwóch wartości logicznych: **prawda** i **fałsz**. Wartość **prawda** jest w JavaScript reprezentowana przez słowo `true`, natomiast wartość **fałsz** przez słowo `false`. Wartości tego typu są używane przy konstruowaniu wyrażeń logicznych, porównywaniu danych, wskazywaniu czy dana operacja zakończyła się sukcesem itp.

Typ null

Typ **null** jest typem specjalnym, reprezentującym wartość pustą. Wartość ta jest określona słowem `null`.

Typ obiektowy

Typ obiektowy służy do reprezentacji obiektów. Nie ma specjalnego słowa kluczowego oznaczającego ten typ. Najczęściej wykorzystuje się obiekty wbudowane oraz udostępniane przez przeglądarkę.

Zmienne

Poznaliśmy już typy danych, czas zapoznać się ze sposobami deklarowania i wykorzystania zmiennych. Są to konstrukcje programistyczne, które pozwalają nam przechowywać dane. Każda zmienna ma swoją nazwę, która ją jednoznacznie identyfikuje, oraz charakteryzuje się typem, który określa, jakie wartości może ona przyjmować. Nazwa może zawierać litery, cyfry i znak podkreślenia, nie może jednak zawierać znaków narodowych (czyli dopuszczalne są jedynie znaki alfabetu łacińskiego). Wolno stosować zarówno wielkie, jak i małe litery, są też one rozróżniane, co oznacza, że przykładowo: `liczba` i `Liczba` to nazwy dwóch różnych zmiennych. Nazwa zmiennej nie może też zaczynać się od cyfry.

W JavaScript, podobnie jak i w wielu innych skryptowych językach programowania, zmiennych nie trzeba jawnie deklarować przed użyciem, a każda z nich może przyjmować dane z dowolnego typu opisanego

w poprzednim podrozdziale. Ponadto typ danych nie jest przypisywany zmiennej na stałe i może się zmieniać w trakcie działania skryptu.

Utworzenie zmiennej polega na umieszczeniu w kodzie skryptu jej nazwy (można ją poprzedzić słowem `var`) **i przypisaniu wartości**, schematycznie:

```
var nazwa_zmiennej = wartość;
```

lub:

```
nazwa_zmiennej = wartość;
```

Wszystko stanie się jaśniejsze po wykonaniu kolejnych ćwiczeń.

W ćwiczeniach tego oraz kolejnych rozdziałów będzie prezentowany tylko właściwy kod skryptów JavaScript, kod HTML będzie natomiast pomijany, o ile nie będzie niezbędny do funkcjonowania przykładu. Listingi dostępne na ftp uwzględniają natomiast zawsze pełny kod strony, czyli zarówno HTML, jak i JavaScript.

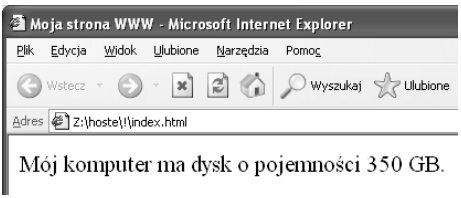
ĆWICZENIE

2.1 Użycie zmiennych w skrypcie

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

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var zmienna1 = "Mój komputer";
var zmienna2 = 350;
document.write(zmienna1 + " ma dysk o pojemności " + zmienna2 + " GB.");
// Koniec kodu JavaScript -->
</script>
```

Po wczytaniu takiej strony na ekranie ukaże się napis `Mój komputer ma dysk o pojemności 350 GB` (rysunek 2.1). Przeanalizujmy więc jak działa ten skrypt. Zadeklarowaliśmy dwie zmienne o nazwach `zmienna1` i `zmienna2`. Zmiennej `zmienna1` przypisaliśmy ciąg znaków `Mój komputer`, zmiennej `zmienna2` natomiast wartość liczbową, dodatnią liczbę całkowitą `350`. 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). Jest to łączenie, inaczej konkatenacja, łańcuchów znakowych.



Rysunek 2.1. *Wyświetlenie na ekranie wartości dwóch zmiennych*

Przekonajmy się teraz, że w JavaScripcie naprawdę w trakcie działania skryptu może się zmieniać typ zmiennej i rodzaj przechowywanych w niej danych.

ĆWICZENIE

2.2 Zmiana typu zmiennej

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.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
zmienna1 = "To jest przykładowy tekst.";
document.write("Pierwsza wartość zmiennej zmienna1: " + zmienna1);
zmienna1 = 24.7;
document.write("<br />Druga wartość zmiennej zmienna1: " + zmienna1);
// Koniec kodu JavaScript -->
</script>
```

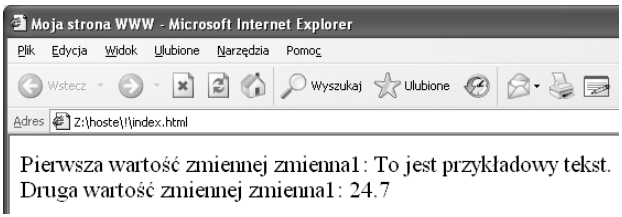
Efekt działania skryptu został przedstawiony na rysunku 2.2. Na początku zmiennej `zmienna1` został przypisany ciąg znaków:

```
zmienna1 = "To jest przykładowy tekst.";
```

który został wyświetlony na ekranie. Następnie zmiennej tej została przypisana wartość rzeczywista 24.7:

```
zmienna1 = 24.7;
```

i tym samym zmieniła ona swój typ. Po pierwszym przypisaniu był to typ łańcuchowy, po drugim — liczbowy. Widać więc wyraźnie, że zmiana typu danych może następować w trakcie działania skryptu.



Rysunek 2.2. *Efekt działania kodu z ćwiczenia 2.2*

Operatory

W JavaScripcie, podobnie jak i w innych językach programowania, występuje wiele operatorów, które pozwalają na wykonywanie rozmaitych operacji. Operatory możemy podzielić na następujące grupy:

- arytmetyczne,
- porównywania (relacyjne),
- bitowe,
- logiczne,
- przypisania,
- pozostałe.

Operatory arytmetyczne

Nietrudno się domyślić, że operatory z tej grupy służą do wykonywania operacji arytmetycznych, czyli dodawania, odejmowania, mnożenia itp. Zostały one zebrane w tabeli 2.1. W tej grupie występują jednak również operatory inkrementacji (zwiększania) i dekrementacji (zmniejszania). Operatory `*`, `/`, `+`, `-`, `%` są dwuargumentowe, natomiast `++` i `--` są jednoargumentowe.

ĆWICZENIE

2.3 Wykonywanie operacji arytmetycznych

Zadeklaruj kilka zmiennych, wykonaj na nich standardowe operacje arytmetyczne i wyświetl wyniki na ekranie.

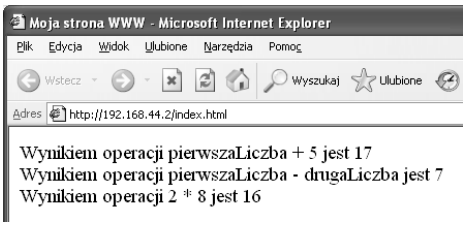
Tabela 2.1. *Operatory arytmetyczne*

Operator	Wykonywane działanie	Przykład
*	mnożenie	x * y
/	dzielenie	x / y
+	dodawanie	x + y
-	odejmowanie	x - y
%	dzielenie modulo (reszta z dzielenia)	x % y
++	inkrementacja (zwiększanie)	x++, ++x
--	dekrementacja (zmniejszanie)	x--, --x

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var pierwszaLiczba = 12;
var drugaLiczba = 5;
var trzeciaLiczba = pierwszaLiczba - drugaLiczba;
document.write("Wynikiem operacji pierwszaLiczba + 5 jest ");
document.write(pierwszaLiczba + 5);
document.write("<br />Wynikiem operacji pierwszaLiczba - drugaLiczba
jest ");
document.write(trzeciaLiczba);
document.write("<br />Wynikiem operacji 2 * 8 jest ");
document.write(2 * 8);
// Koniec kodu JavaScript -->
</script>
```

Wynik działania przedstawionego skryptu z pewnością nie jest żadnym zaskoczeniem (rysunek 2.3). Zostały w nim zadeklarowane trzy zmienne: `pierwszaLiczba`, `drugaLiczba`, `trzeciaLiczba`. Pierwszym dwóm zostały przypisane wartości całkowite `12` i `5`, a ostatniej wartość wynikająca z działania `pierwszaLiczba - drugaLiczba`, czyli `7`. Na ekranie zostały wyświetlone wyniki działań:

- `pierwszaLiczba + 5` (czyli 17),
- `pierwszaLiczba - drugaLiczba` (czyli 7),
- `2 * 8` (czyli 16).



Rysunek 2.3. *Wynik działania kilku operacji arytmetycznych*

Widać więc wyraźnie, że operacje arytmetyczne mogą być wykonywane zarówno na dwóch zmiennych, na zmiennej i liczbie, jak i dwóch liczbach.

Do operatorów arytmetycznych należy również `%`, przy czym, jak zostało to zaznaczone w tabeli 2.1, nie oznacza on obliczania procentów, ale dzielenie modulo, czyli resztę z dzielenia. Przykładowo, działanie `10 % 3` da w wyniku 1. Trójka zmieści się bowiem w dziesięciu 3 razy, pozostawiając resztę 1 (`3 * 3 = 9`, `9 + 1 = 10`). Podobnie `21 % 8 = 5`, ponieważ `2 * 8 = 16`, `16 + 5 = 21`.

Ciekawym 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. Oznacza to, że 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 operacja `x++` zwiększa wartość zmiennej po jej wykorzystaniu, natomiast `++x` przed jej wykorzystaniem. Takie rozróżnienie może być bardzo pomocne podczas pisania skryptów. Przyjrzyjmy się zatem bliżej operatorowi inkrementacji.

ĆWICZENIE

2.4 Operator inkrementacji

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

```
<script type="text/javascript">
<!-- Ukrycie przed przegl±darkami nie obsługuj±cymi JavaScriptu
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>
```

Wynikiem działania skryptu (dla ułatwienia opisu wiersze zostały ponumerowane) będzie ciąg znaków `13 13 14 14 16`, tak jak jest to widoczne na rysunku 2.4. Dlaczego? Otóż w wierszu oznaczonym numerem 1. najpierw jest zwiększana wartość zmiennej `x` o 1 (czyli `x = 13`), a następnie ten wynik jest wyświetlany. W linii 3. najpierw jest wyświetlana aktualna wartość zmiennej `x` (czyli `13`), a następnie jest ona zwiększana o 1 (czyli `x = 14`). W wierszu 5. jest wyświetlana aktualna wartość zmiennej `x`, czyli `14`. W wierszu 7. zmiennej `y` jest przypisywana wartość zmiennej `x`, a następnie zmienna `x` jest zwiększana o 1 (czyli `y = 14`, `x = 15`). W wierszu 10. 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 nieco skomplikowane, ale po dokładnym przeanalizowaniu i samodzielnym wykonaniu kilku własnych ćwiczeń operator ten nie powinien sprawiać żadnych kłopotów.

Rysunek 2.4. *Wynik ćwiczenia dotyczącego operatora dekrementacji*

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

ĆWICZENIE

2.5 Operator dekrementacji

Zmień kod z ćwiczenia 2.4 tak, aby operator ++ został zastąpiony operatorem --. Następnie przeanalizuj jego działanie i sprawdź, czy otrzymany wynik jest taki sam jak na ekranie przeglądarki.

```
<script type="text/javascript">
<!-- Ukrycie przed przegl±darkami nie obsługuj±cymi JavaScriptu
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>
```

Wynikiem działania skryptu będzie ciąg znaków `11 11 10 10 8`. W wierszu 1. najpierw jest zmniejszana wartość `x` o 1 (czyli `x = 11`), a następnie ten wynik jest wyświetlany. W wierszu 3. najpierw jest wyświetlana aktualna wartość `x` (czyli `11`), a następnie jest ona zmniejszana o 1 (czyli `x = 10`). W wierszu 5. jest wyświetlana aktualna wartość `x`, czyli `10`. W wierszu 7. zmiennej `y` jest przypisywana wartość `x`, a następnie zmienna `x` jest zmniejszana o 1 (czyli `y = 10`, `x = 9`). W wierszu 10. najpierw jest zmniejszana wartość `x` o 1 (czyli `x = 8`), a następnie wartość ta jest przypisywana zmiennej `y` (czyli `y = 8` i `x = 8`). Na zakończenie w linii 11. wartość `y` jest wyświetlana na ekranie.

Operatory porównywania (relacyjne)

Operatory porównania, czyli relacyjne, służą oczywiście do porównywania argumentów. Wynikiem takiego porównania jest wartość logiczna `true` (jeśli jest ono prawdziwe) lub `false` (jeśli jest fałszywe). Zatem wynikiem operacji `argument1 == argument2` będzie `true`, jeżeli

argumenty są sobie równe, oraz `false`, jeżeli argumenty są różne. Czyli `4 == 5` ma wartość `false`, a `2 == 2` ma wartość `true`. Do dyspozycji mamy operatory porównania zawarte w tabeli 2.2. Przykłady ich wykorzystania znajdują się w podrozdziale dotyczącym instrukcji warunkowych. Operatory `===` i `!==` zostały wprowadzone w JavaScript 1.3 i JScript 3.0.

Tabela 2.2. *Operatory porównywania*

Operator	Opis	Przykład
`==`	Wynikiem jest `true`, jeśli argumenty są sobie równe.	`a == b`
`!=`	Wynikiem jest `true`, jeśli argumenty są różne.	`a != b`
`===`	Wynikiem jest `true`, jeśli oba argumenty są tego samego typu i są sobie równe.	`a === b`
`!==`	Wynikiem jest `true`, jeśli argumenty są różne, bądź są różnych typów.	`a !== b`
`>`	Wynikiem jest `true`, jeśli argument lewostronny jest większy od prawostronnego.	`a > b`
`<`	Wynikiem jest `true`, jeśli argument lewostronny jest mniejszy od prawostronnego.	`a < b`
`>=`	Wynikiem jest `true`, jeśli argument lewostronny jest większy od prawostronnego lub równy mu.	`a >= b`
`<=`	Wynikiem jest `true`, jeśli argument lewostronny jest mniejszy od prawostronnego lub równy mu.	`a <= b`

Operatory bitowe

Operatory bitowe pozwalają na wykonywanie operacji na poszczególnych bitach liczb i zostały przedstawione w tabeli 2.3. Są to: iloczyn bitowy (koniunkcja bitowa, operacja AND), suma bitowa (alternatywa bitowa, operacja OR), negacja bitowa (uzupełnienie do jedynki, operacja NOT), suma bitowa modulo 2 (alternatywa bitowa wykluczająca, różnica symetryczna, operacja XOR) oraz operacje przesunięć bitów. Wszystkie operatory są dwuargumentowe, oprócz operatora bitowej negacji, który jest jednoargumentowy.

Tabela 2.3. *Operatory bitowe*

Operator	Wykonywane działanie	Przykład
`&`	iloczyn bitowy `AND`	`a & b`
`|`	suma bitowa `OR`	`a | b`
`~`	negacja bitowa `NOT`	`~a`
`^`	bitowa różnica symetryczna `XOR`	`a ^ b`
`>>`	przesunięcie bitowe w prawo	`a >> n`
`<<`	przesunięcie bitowe w lewo	`a << n`
`>>>`	przesunięcie bitowe w prawo z wypełnieniem zerami	`a >>> n`

Operatory logiczne

Operacje logiczne mogą być wykonywane na argumentach, które posiadają wartość logiczną: prawda (`true`) lub fałsz (`false`). Operatory logiczne zostały przedstawione w tabeli 2.4. Operatory `&&` i `||` są dwuargumentowe, natomiast operator `!` jest jednoargumentowy.

Tabela 2.4. *Operatory logiczne*

Operator	Wykonywane działanie	Przykład
`&&`	iloczyn logiczny (`AND`)	`a && b`
`||`	suma logiczna (`OR`)	`a || b`
`!`	negacja logiczna (`NOT`)	`!a`

Iloczyn logiczny

Wynikiem iloczynu logicznego jest wartość `true` wtedy i tylko wtedy, kiedy oba argumenty mają wartość `true`. W każdym innym przypadku wynikiem jest `false`. Obrazuje to tabela 2.5.

Tabela 2.5. *Działanie iloczynu logicznego*

Argument 1	Argument 2	Wynik
`true`	`true`	`true`
`true`	`false`	`false`
`false`	`true`	`false`
`false`	`false`	`false`

Suma logiczna

Wynikiem sumy logicznej jest wartość `false` wtedy i tylko wtedy, kiedy oba argumenty mają wartość `false`. W każdym innym przypadku wynikiem jest `true`. Obrazuje to tabela 2.6.

Tabela 2.6. *Działanie sumy logicznej*

Argument 1	Argument 2	Wynik
`true`	`true`	`true`
`true`	`false`	`true`
`false`	`true`	`true`
`false`	`false`	`false`

Negacja logiczna

Operacja logicznej negacji zamienia wartość argumentu na przeciwną. Czyli jeśli argument miał wartość `true`, będzie miał wartość `false`, i odwrotnie, jeśli miał wartość `false`, będzie miał wartość `true`. Obrazuje to tabela 2.7.

Tabela 2.7. *Działanie negacji logicznej*

Argument	Wynik
`true`	`false`
`false`	`true`

Operatory przypisania

Operatory przypisania są dwuargumentowe i powodują przypisanie wartości argumentu znajdującego się z prawej strony operatora argumentowi znajdującemu się z lewej strony. Taką najprostszą operację już poznaliśmy, odbywa się ona przy wykorzystaniu operatora = (równa się). Jeśli napiszemy `liczba = 10`, oznacza to, że zmiennej `liczba` chcemy przypisać wartość `10`.

W JavaScripcie istnieje jednak również cały zestaw operatorów łączących operację przypisania z inną operacją. Przykładowo istnieje operator +=, który oznacza: przypisz argumentowi umieszczonemu z lewej

strony wartość wynikającą z dodawania argumentu znajdującego się z lewej strony i argumentu znajdującego się z prawej strony operatora.

Choć brzmi to z początku nieco zawile, w rzeczywistości jest bardzo proste i znacznie upraszcza niektóre konstrukcje programistyczne. Po prostu przykładowy zapis:

```
liczba += 5
```

tłumaczymy jako:

```
liczba = liczba + 5
```

i oznacza: przypisz zmiennej `liczba` wartość wynikającą z dodawania `liczba + 5`.

W JavaScripcie występuje cała grupa tego typu operatorów, zostały one zebrane w tabeli 2.8.

Tabela 2.8. *Operatory przypisania i ich znaczenie*

Argument 1	Operator	Argument 2	Znaczenie
x	=	y	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 = x >> y
x	>>>=	y	x = x >>> y
x	&=	y	x = x & y
x	|=	y	x = x | y
x	^=	y	x = x ^ y

Pozostałe operatory

W JavaScripcie występuje jeszcze kilka innych operatorów, których jednak nie będziemy omawiać. Są to m.in. operator indeksowania tablic, wywołania funkcji, rozdzielania wyrażeń, tworzenia obiektów itp. W podrozdziale dotyczącym instrukcji warunkowych zostanie natomiast omówiony operator warunkowy.

Priorytety operatorów

Sama znajomość operatorów to jednak nie wszystko. Niezbędna jest jeszcze wiedza na temat tego, jaki mają one priorytet, czyli jaka jest kolejność ich wykonywania. Wiadomo np., że mnożenie jest „silniejsze" od dodawania, zatem najpierw mnożymy, potem dodajemy (tę kolejność można zmienić, stosując nawiasy okrągłe, dokładnie w taki sam sposób, w jaki zmienia się kolejność działań w matematyce). W JavaScripcie jest podobnie — siła każdego operatora jest ściśle określona. Przedstawia to tabela 2.9. Im wyższa pozycja w tabeli, tym wyższy priorytet operatora. Operatory znajdujące się na jednym poziomie (w jednym wierszu) mają ten sam priorytet[1].

Tabela 2.9. *Priorytety operatorów*

Lp.	Nazwy	Symbole
1	indeks tablicy, wywołanie funkcji	`[]`, `()`
2	inkrementacja i dekrementacja, ustalenie znaku, negacja bitowa i logiczna, utworzenie obiektu, ustalenie typu zmiennej, usunięcie składowej	`++`, `--`, `+`, `-`, `~`, `!`, `new`, `typeof`, `delete`
3	mnożenie, dzielenie, reszta z dzielenia	`*`, `/`, `%`
4	dodawanie, odejmowanie	`+`, `-`
5	przesunięcie bitowe w lewo, w prawo, w prawo z wypełnieniem zerami	`<<`, `>>`, `>>>`
6	mniejsze, większe, mniejsze lub równe, większe lub równe, porównanie typów	`<`, `>`, `<=`, `>=`
7	równe, różne	`==`, `!=`
8	iloczyn bitowy	`&`
9	bitowa różnica symetryczna	`^`
10	suma bitowa	`|`
11	iloczyn logiczny	`&&`
12	suma logiczna	`||`
13	warunkowy	`? :`
14	operatory przypisania	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`
15	rozdzielanie wyrażeń	`,`

[1] Tabela uwzględnia również operatory, które nie były omawiane w książce.

Instrukcje warunkowe

Instrukcja if…else

Bardzo często w programie zachodzi potrzeba sprawdzenia jakiegoś warunku i w zależności od tego, czy jest on prawdziwy, czy fałszywy, dalszego wykonywania różnych instrukcji. Do takiego sprawdzania służy właśnie instrukcja warunkowa `if…else`. Ma ona ogólną postać:

```
if (wyrażenie warunkowe){
  //instrukcje do wykonania, jeżeli warunek jest prawdziwy
}
else{
  //instrukcje do wykonania, jeżeli warunek jest fałszywy
}
```

ĆWICZENIE

2.6 Użycie instrukcji warunkowej

Wykorzystaj instrukcję warunkową `if…else` do stwierdzenia, czy wartość zmiennej typu całkowitego jest mniejsza od zera.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var liczba = -12;
if(liczba > 0){
  document.write("Zmienna liczba jest większa od 0.");
}
else{
  document.write("Zmienna liczba nie jest większa od 0.");
}
// Koniec kodu JavaScript -->
</script>
```

W skrypcie została zadeklarowana zmienna `liczba` o wartości `-12`. Do stwierdzenia, czy wartość tej zmiennej jest większa od zera, czy też nie, została użyta instrukcja warunkowa `if` zawierająca wyrażenie warunkowe w postaci:

```
liczba > 0
```

wykorzystujące operator warunkowy `>`. Takie wyrażenie przyjmuje wartość `true`, jeśli zmienna jest większa od `0`, oraz wartość `false` w przeciwnym przypadku. Całą instrukcję `if` należy więc rozumieć

następująco: jeśli wartość zmiennej `liczba` jest większa od `0`, wykonaj instrukcję:

```
document.write("Zmienna liczba jest większa od 0.");,
```

a w przeciwnym przypadku instrukcję:

```
document.write("Zmienna liczba nie jest większa od 0.");.
```

Instrukcja if…else if

Instrukcja `if…else if` pozwala na zbadanie wielu warunków. Ma ona schematyczną postać:

```
if (warunek1){
  instrukcje1;
}
else if (warunek2){
  instrukcje2;
}
else if (warunek3){
..instrukcje3;
}
...
else if (warunekn){
..instrukcjen;
}
else{
..instrukcjem;
}
```

Co oznacza: jeżeli *`warunek1`* jest prawdziwy, to zostaną wykonane *`instrukcje1`*, w przeciwnym przypadku, jeżeli jest prawdziwy *`warunek2`*, to zostaną wykonane *`instrukcje2`*, w przeciwnym przypadku, jeśli jest prawdziwy *`warunek3`*, to zostaną wykonane *`instrukcje3`* itd. Jeżeli żaden z warunków nie będzie prawdziwy, to zostaną wykonane *`instrukcjem`*. Ostatni blok `else` jest jednak opcjonalny i nie musi być stosowany.

Taka konstrukcja może być wykorzystana np. w sytuacji, kiedy chcemy wykonać wiele różnych instrukcji w zależności od stanu zmiennej.

ĆWICZENIE

2.7 Złożona instrukcja warunkowa

Użyj złożonej instrukcji warunkowej do określenia wartości wybranej zmiennej.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var liczba = 20;
if(liczba == 10){
  document.write("Zmienna liczba jest równa 10.");
}
else if(liczba == 20){
  document.write("Zmienna liczba jest równa 20.");
}
else if(liczba == 30){
  document.write("Zmienna liczba jest równa 30.");
}
else{
  document.write("Zmienna liczba nie jest równa ani 10, ani 20, ani
30.");
}
// Koniec kodu JavaScript -->
</script>
```

Przedstawiona w skrypcie złożona instrukcja warunkowa bada po kolei warunki: `liczba == 10`, `liczba == 20`, `liczba == 30`, czyli sprawdza, czy zmienna liczba ma wartość 10, 20 lub 30. Jeśli którykolwiek z tych warunków jest prawdziwy, za pomocą instrukcji `document.write` jest wyświetlany odpowiedni komunikat. W przypadku, kiedy wszystkie warunki są fałszywe, jest wykonywany blok `else`, czyli instrukcja:

```
document.write ("Zmienna liczba nie jest równa ani 10, ani 20, ani
30.");
```

Operator warunkowy

Operator warunkowy pozwala w niektórych przypadkach na wygodne zastąpienie bloku `if…else`. Konstrukcja taka wygląda następująco:

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

Należy rozumieć to w sposób następujący: jeżeli *wyrażenie warunkowe* jest prawdziwe, czyli ma wartość `true` — całość przyjmuje wartość

wartość1, w przeciwnym przypadku — *wartość2*. Najłatwiej zrozumieć ten zapis, wykonując kolejne ćwiczenie.

ĆWICZENIE

2.8 Jak działa operator warunkowy

Wykorzystaj operator warunkowy do zmodyfikowania wartości dowolnej zmiennej.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var x = 1, y;
y = (x < 0) ? 10 : 20;
document.write("Wartość y to " + y + ".");
// Koniec kodu JavaScript -->
</script>
```

Najważniejsza jest tu oczywiście linia `y = (x < 0) ? 10 : 20;`. Po lewej stronie operatora przypisania = znajduje się zmienna (`y`), natomiast po stronie prawej — wyrażenie warunkowe, czyli linia ta oznacza: przypisz zmiennej `y` wartość wyrażenia warunkowego. Jaka jest ta wartość? Trzeba przeanalizować samo wyrażenie: `(x < 0) ? 10 : 20`. Oznacza ono, zgodnie z tym, co zostało napisane wcześniej: jeżeli wartość zmiennej `x` jest mniejsza od zera, przypisz wyrażeniu wartość `10`, w przeciwnym przypadku (zmienna `x` większa lub równa zero) przypisz wyrażeniu wartość `20`. Ponieważ zmiennej `x` na początku skryptu została przypisana wartość `1`, wartością całego wyrażenia będzie `20` i ta właśnie wartość zostanie przypisana zmiennej `y`.

W tym miejscu ponownie zajrzyjmy do tabeli 2.9. Z podanych w niej informacji wynika, że operator warunkowy ma mniejszy priorytet niż operatory relacyjne. W związku z tym nawiasy okrągłe wprowadzone do wyrażenia w celu zwiększenia czytelności zapisu mogą zostać pominięte i może mieć ono również postać:

```
x < 0 ? 10 : 20;
```

Instrukcja switch

Instrukcja wyboru `switch` (nazywana również instrukcją `switch…case`) pozwala w wygodny sposób sprawdzić ciąg warunków i wykonać różne

instrukcje w zależności od wyników porównywania. Ma ona ogólną postać:

```
switch(wyrażenie){
  case wartość1 :
    instrukcje1;
    break;
  case wartość2 :
    instrukcje2;
    break;
  case wartość3 :
    instrukcje3;
    break;
  default :
    instrukcje4;
}
```

którą należy rozumieć następująco: sprawdź wartość wyrażenia *wyrażenie*, jeśli wynikiem jest *wartość1*, to wykonaj *intrukcje1* i przerwij wykonywanie bloku `switch` (instrukcja `break`). Jeśli wynikiem jest *wartość2*, to wykonaj *intrukcje2* i przerwij wykonywanie bloku `switch`, jeśli jest *wartość3*, to wykonaj *intrukcje3* i przerwij wykonywanie bloku `switch`. Jeśli nie zachodzi żaden z wymienionych przypadków, wykonaj *instrukcje4* i zakończ blok `switch`. Blok `default` jest jednak opcjonalny i może zostać pominięty.

Łatwo zauważyć, że jest to odpowiednik złożonej instrukcji `if…else if` w postaci:

```
if(wyrażenie == wartość1){
  instrukcje1;
}
else if(wyrażenie == wartość2){
  instrukcje2;
}
else if(wyrażenie == wartość3){
  instrukcje3;
}
else{
  instrukcje4;
}
```

Potwierdźmy to, wykonując kolejne ćwiczenie.

ĆWICZENIE

2.9 Instrukcja wyboru switch

Zmodyfikuj kod ćwiczenia 2.7 w taki sposób, aby działanie skryptu było identyczne, ale została użyta instrukcja `switch`.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var liczba = 20;
switch(liczba){
  case 10 :
    document.write("Zmienna liczba jest równa 10.");
    break;
  case 20:
    document.write("Zmienna liczba jest równa 20.");
    break;
  case 30:
    document.write("Zmienna liczba jest równa 30.");
    break;
  default:
    document.write("Zmienna liczba nie jest równa ani 10, ani 20, ani 30.");
}
// Koniec kodu JavaScript -->
</script>
```

Na początku została utworzona zmienna `liczba` o wartości `20`. Dalej znajduje się instrukcja `switch`, która najpierw oblicza wartość wyrażania występującego w nawiasie okrągłym. Ponieważ w tym przypadku jako wyrażenie występuje nazwa zmiennej, wartością wyrażenia staje się wartość tej zmiennej (czyli 20). Wartość ta jest porównywana do wartości występujących po słowach `case`, czyli `10`, `20` i `30`. Jeśli zgodność zostanie stwierdzona, zostaną wykonane instrukcje występujące w danym bloku `case`. Jeśli nie uda się dopasować wartości wyrażenia do żadnej z wartości występujących po słowach `case`, jest wykonywany blok `default`. Ponieważ wartością wyrażenia jest `20`, zgodność jest stwierdzana w drugim bloku `case` i są wykonywane instrukcje znajdujące się w tym bloku, czyli:

```
document.write ("Zmienna liczba jest równa 10.");
break;
```

Występująca po `document.write` instrukcja `break` przerywa wykonywanie bloku `case`.

Na instrukcję break należy zwrócić szczególną uwagę. Jej przypadkowe pominięcie może doprowadzić do nieoczekiwanych wyników i błędów w skrypcie. Aby przekonać się, w jaki sposób działa instrukcja switch bez instrukcji break, zmodyfikujmy skrypt z ćwiczenia 2.9.

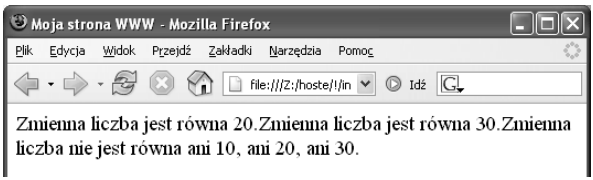
ĆWICZENIE

2.10 Switch bez break

Zmodyfikuj kod z ćwiczenia 2.9, usuwając z niego wszystkie instrukcje break. Zaobserwuj działanie skryptu.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var liczba = 20;
switch(liczba){
  case 10 :
    document.write("Zmienna liczba jest równa 10.");
  case 20:
    document.write("Zmienna liczba jest równa 20.");
  case 30:
    document.write("Zmienna liczba jest równa 30.");
  default:
    document.write("Zmienna liczba nie jest równa ani 10, ani 20, ani 30.");
}
// Koniec kodu JavaScript -->
</script>
```

Po uruchomieniu skryptu w przeglądarce pojawi się obraz widoczny na rysunku 2.5. Wyraźnie nie spełnia on swojego zadania. Zmienna nie może przecież jednocześnie spełniać kilku przeciwstawnych warunków. Jak więc działa przedstawiony kod? Otóż jeśli w którymś z bloków (przypadków) case zostanie wykryta zgodność z wyrażeniem występującym za switch, zostaną wykonane wszystkie dalsze instrukcje, aż do napotkania instrukcji break lub dotarcia do końca instrukcji switch. W kodzie ćwiczenia zgodność jest stwierdzana już w drugim bloku case, jest więc wykonywana znajdująca się w nim instrukcja document.write. Ponieważ jednak w bloku tym nie ma instrukcji break, są wykonywane instrukcje znajdujące się w kolejnym bloku case (case 30). W tym bloku również brakuje break, a zatem są wykonywane instrukcje znajdujące się po słowie default. Tym samym wykonane zostaną prawie wszystkie znajdujące się w kodzie instrukcje document.write.



Rysunek 2.5. *Wynik pominięcia niezbędnych instrukcji break*

Pętle

Pętle to wyrażenia służące do wykonywania powtarzających się czynności. Przykładowo gdybyśmy chcieli 100 razy wypisać na stronie **pewien tekst**, to można by w tym celu użyć 100 instrukcji `document.write`, ale byłoby to niewątpliwie niewygodne rozwiązanie. Pętle pozwalają na automatyzację takich czynności. W JavaScript mamy do dyspozycji następujące rodzaje pętli:

- `for`,
- `for..in`,
- `while`,
- `do…while`.

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 liczby wykonań 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. Najlepiej pokazać to na konkretnym przykładzie.

ĆWICZENIE

2.11 Prosta pętla typu for

Użyj pętli typu `for`, aby 10 razy wyświetlić na ekranie dowolny napis.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
for (var i = 0; i < 10; i++){
  document.write("Pętla typu for <br />");
}
// Koniec kodu JavaScript -->
</script>
```

Taką konstrukcję należy rozumieć następująco: utwórz zmienną `i` i przypisz jej wartość zero (`i = 0`), następnie tak długo jak wartość `i` jest mniejsza od `10 (i < 10)` wykonuj instrukcje znajdujące się wewnątrz pętli (instrukcja `document.write`) oraz zwiększaj `i` o jeden (`i++`). Tym samym na ekranie pojawi się 10 razy napis `Pętla typu for`, tak jak jest to widoczne na rysunku 2.6. Zmienna `i` jest nazywana zmienną iteracyjną, czyli kontrolującą kolejne przebiegi (iteracje) pętli.

Rysunek 2.6. *Efekt wykorzystania pętli for do wielokrotnego wyświetlenia napisu*

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

```
for (wyrażenie początkowe; wyrażenie warunkowe;){
  instrukcje do wykonania
  wyrażenie modyfikujące
}
```

ĆWICZENIE

2.12 Wyrażenie modyfikujące wewnątrz pętli

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

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
for (var i = 0; i < 10;){
  document.write ("Pętla typu for <br />");
  i++;
}
// Koniec kodu JavaScript -->
</script>
```

Ważne jest, aby pamiętać o średniku występującym po wyrażeniu `i < 10` — jest on bowiem niezbędny dla prawidłowego funkcjonowania skryptu.

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

ĆWICZENIE

2.13 Wyrażenie początkowe przed pętlą

Przenieś wyrażenie początkowe przed pętlę `for`.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var i = 0;
for (; i < 10;){
  document.write ("Pętla typu for <br />");
  i++;
}
// Koniec kodu JavaScript -->
</script>
```

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

ĆWICZENIE

2.14 Pętla bez wyrażeń

Umieść wyrażenie warunkowe i modyfikujące we wnętrzu pętli, natomiast wyrażenie początkowe przenieś poza nią.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var i = 0;
for (; ;){
  document.write ("Pętla typu for <br />");
  if (i++ >= 9) break;
}
// Koniec kodu JavaScript -->
</script>
```

Przy stosowaniu tego typu konstrukcji pamiętajmy, że oba średniki w nawiasach okrągłych występujących po `for` są niezbędne do prawidłowego funkcjonowania kodu. Warto też zwrócić uwagę na zmianę kierunku nierówności. We wcześniejszych przykładach sprawdzaliśmy bowiem, czy `i` jest mniejsze bądź równe 10, a teraz, czy jest większe bądź równe 9. Dzieje się tak, ponieważ poprzednio sprawdzaliśmy, czy pętla ma się dalej wykonywać, natomiast obecnie, czy ma się zakończyć. Przy okazji wykorzystaliśmy też kolejną instrukcję, mianowicie `break`. Służy ona do natychmiastowego przerwania wykonywania pętli.

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

2.15 Użycie instrukcji continue

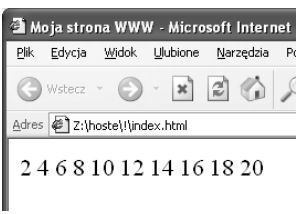
Wyświetl na ekranie liczby pomiędzy 1 a 20 podzielne przez 2. Skorzystaj z pętli `for` i instrukcji `continue`.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
for (var i = 1; i <= 20; i++){
  if ((i % 2) != 0)
    continue;
```

```
  document.write (i + " ");
}
// Koniec kodu JavaScript -->
</script>
```

Efekt działania kodu został przedstawiony na rysunku 2.7. Wewnątrz pętli znajduje się instrukcja `if` sprawdzająca warunek `(i % 2) != 0`. Wykorzystuje on operator dzielenia modulo (`%`) do stwierdzenia, czy dana liczba jest podzielna przez 2. Jeśli bowiem reszta z dzielenia przez 2 jest równa 0 (`i % 2` równe 0), to dana wartość jest podzielna przez 2, jeśli natomiast reszta z dzielenia przez 2 jest różna od 0 (`i % 2` różne od 0), to dana wartość jest niepodzielna przez dwa. Stąd dla każdej wartości zmiennej `i` niepodzielnej przez dwa zostanie wykonana instrukcja `continue` rozpoczynająca kolejną iterację pętli, a dzięki temu liczby nieparzyste nie pojawią się na ekranie.

Rysunek 2.7. *Liczby podzielne przez dwa*



Oczywiście do realizacji zadania przedstawionego w ćwiczeniu 2.15 nie jest niezbędne wykorzystanie instrukcji `continue`. Można je również wykonać, używając samej instrukcji `if`.

ĆWICZENIE

2.16 Liczby podzielne przez dwa

Wykonaj zadanie z ćwiczenia 2.15 bez użycia instrukcji `continue`.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
for (var i = 1; i <= 20; i++){
  if ((i % 2) == 0)
    document.write (i + " ");
}
// Koniec kodu JavaScript -->
</script>
```

Pętla for…in

Pętla typu `for…in` pozwala na odczytanie wartości oraz nazw właściwości obiektów (w tym również tablic). Ma ona schematyczną postać:

```
for (var nazwa in obiekt){
  //instrukcje
}
```

W takim przypadku we wnętrzu pętli pod identyfikator *nazwa* podstawiane są kolejne właściwości obiektu *obiekt*. Przykład jej użycia zostanie podany w dalszej części książki.

Pętla while

O ile pętla typu `for` służy zwykle do wykonywania znanej z góry liczby operacji, to w przypadku pętli `while` liczba ta z reguły nie jest znana. Nie jest to jednak obligatoryjny podział, ponieważ obie pętle można napisać w taki sposób, aby były swoimi funkcjonalnymi odpowiednikami. Ogólna konstrukcja pętli typu `while` jest następująca:

```
while (wyrażenie warunkowe)
{
  instrukcje
}
```

Instrukcje są wykonywane dopóty, dopóki wyrażenie warunkowe jest prawdziwe.

ĆWICZENIE

2.17 Konstrukcja pętli while

Użyj pętli `while`, aby 10 razy wyświetlić na ekranie dowolny napis.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
i = 0;
while(i++ < 10){
  document.write ("Pętla typu while<br />");
}
// Koniec kodu JavaScript -->
</script>
```

ĆWICZENIE

2.18 Liczby nieparzyste i pętla while

Korzystając z pętli `while`, napisz skrypt wyświetlający na ekranie liczby od 1 do 20 niepodzielne przez 2.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
i = 1;
while (i <= 20){
 if (i % 2 == 0)
   document.write (i + " ");
 i++;
}
// Koniec kodu JavaScript -->
</script>
```

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

Pętla do…while

Pętla `do…while` jest odmianą pętli `while`. Ma ona następującą postać:

```
do{
  instrukcje;
}
while(warunek);
```

Konstrukcję tę należy rozumieć: wykonuj *instrukcje*, dopóki *warunek* jest prawdziwy.

ĆWICZENIE

2.19 Użycie pętli do…while

Korzystając z pętli `do…while`, napisz program wyświetlający na ekranie 10 razy dowolny napis.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var i = 0;
do{
  document.write ("Pętla do…while <br />");
}
```

```
while (i++ < 9);
// Koniec kodu JavaScript -->
</script>
```

Wydawać by się mogło, że to przecież to samo, co zwykła pętla `while`. Jest jednak pewna różnica. Otóż w przypadku pętli `do…while` instrukcje wykonane są co najmniej jeden raz, nawet jeśli warunek jest na pewno fałszywy.

ĆWICZENIE

2.20 Pętla do…while z fałszywym warunkiem

Napisz pętlę `do…while` zawierającą fałszywy warunek. We wnętrzu pętli umieść instrukcję wyświetlającą dowolny napis na ekranie. Zaobserwuj działanie skryptu.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
do{
  document.write ("Pętla do…while <br />");
}
while (false);
// Koniec kodu JavaScript -->
</script>
```

W przedstawionej pętli warunek jej kontynuacji jest na pewno fałszywy (`false`), a mimo to na ekranie pojawi się napis. Dzieje się tak dlatego, że w przypadku tej pętli najpierw są wykonywane instrukcje umieszczone w jej wnętrzu, a dopiero potem jest sprawdzany warunek.

Funkcje

Definiowanie funkcji

Funkcje są to wydzielone bloki kodu przeznaczone do wykonywania określonych zadań. Schematyczna definicja funkcji ma postać:

```
function nazwa_funkcji(argumenty_funkcji)
{
```

```
  //kod funkcji
}
```

Nazwa funkcji, podobnie jak inne identyfikatory, może składać się z liter (alfabetu łacińskiego), cyfr oraz znaków podkreślenia, nie może natomiast zawierać znaków narodowych. Wolno stosować zarówno wielkie, jak i małe litery, które są rozróżniane, co oznacza, że przykładowe nazwy: `funkcja` i `Funkcja` są traktowane jako różne. Nazwa funkcji nie może zaczynać się od cyfry.

Aby wykonać instrukcje znajdujące się wewnątrz funkcji (pomiędzy znakami nawiasu klamrowego), należy ją wywołać. Wywołanie polega na umieszczeniu w kodzie skryptu nazwy funkcji wraz z występującymi po niej znakami nawiasu okrągłego. Zobaczmy, jak tego typu konstrukcja będzie działała w praktyce.

ĆWICZENIE

2.21 Utworzenie i wywołanie funkcji

Utwórz funkcję, której zadaniem będzie wyświetlenie napisu, a następnie wywołaj ją w kodzie skryptu.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
function wyswietl_napis()
{
  document.write("Instrukcja document.write z wnętrza funkcji.");
}

wyswietl_napis();

// Koniec kodu JavaScript -->
</script>
```

W skrypcie najpierw została zdefiniowana funkcja o nazwie `wyswietl_napis`, a następnie została ona wywołana przez podanie jej nazwy zakończonej znakami nawiasu okrągłego. Wewnątrz funkcji (stosuje się też termin: w ciele funkcji) umieszczona jest instrukcja `document.write`. Ponieważ wywołanie funkcji jest równoznaczne z wykonaniem znajdujących się w niej instrukcji, w przeglądarce zobaczymy zdefiniowany napis.

Argumenty funkcji

Funkcjom można przekazywać argumenty, czyli wartości (dane), które mogą wpływać na ich „zachowanie” lub też być przez nie przetwarzane. Listę argumentów należy umieścić w nawiasie okrągłym za nazwą funkcji, oddzielając je od siebie znakami przecinka. A zatem taka konstrukcja ma schematyczną postać:

```
funtion nazwa_funkcji(argument1, argument2, ... , argumentN)
{
  //instrukcje wnętrza funkcji
}
```

Napiszmy więc funkcję, której zadaniem będzie wyświetlenie wartości przekazanego jej argumentu i wywołajmy ją w kodzie skryptu.

ĆWICZENIE

2.22 Funkcja przyjmująca argument

Napisz funkcję przyjmującą jeden argument i wyświetlającą jego wartość na ekranie. Wywołaj ją z różnymi argumentami.

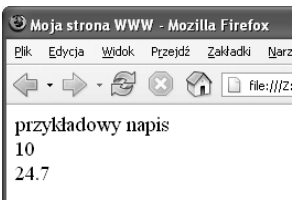
```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
function wyswietl_wartosc(wartosc)
{
  document.write(wartosc);
}

wyswietl_wartosc("przykładowy napis");
document.write("<br />");
wyswietl_wartosc(10);
document.write("<br />");
wyswietl_wartosc(24.7);

// Koniec kodu JavaScript -->
</script>
```

Powstała tu funkcja `wyswietl_wartosc`, która przyjmuje jeden argument o nazwie `wartosc`. Wewnątrz funkcji można odczytać wartość tego argumentu, odwołując się do jego nazwy. W ten sposób można przekazać argument dowolnego typu, typ nie jest bowiem z góry określony. Tym samym instrukcja `document.write(wartosc);` powoduje wyświetlenie tej wartości na ekranie. W dalszym kodzie skryptu funkcja `wyswietl_wartosc` została wywołana trzykrotnie, za każdym razem z innym argumentem. W pierwszym przypadku był to ciąg znaków, w drugim — liczba

całkowita, a w trzecim — liczba rzeczywista. Po uruchomieniu kodu w przeglądarce zobaczymy zatem widok jak na rysunku 2.8.



Rysunek 2.8. *Wynik wywołań funkcji wyswietl_wartosc z różnymi argumentami*

Funkcja może również przyjmować więcej niż jeden argument i wykonywać na nich operacje.

ĆWICZENIE

2.23 Operacje na argumentach

Napisz funkcję przyjmującą dwa argumenty i wyświetlającą wynik ich dodawania.

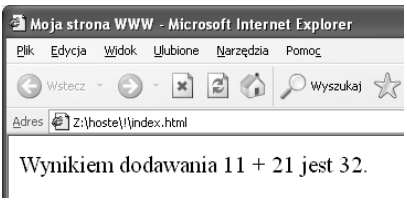
```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu

function dodaj(wart1, wart2)
{
  document.write("Wynikiem dodawania " + wart1 + " + " + wart2 + " jest ");
  document.write(wart1 + wart2);
  document.write(".");
}

dodaj (11, 21);

// Koniec kodu JavaScript -->
</script>
```

Skrypt zawiera funkcję o nazwie `dodaj`, która przyjmuje dwa argumenty: `wart1` i `wart2`. W jej wnętrzu wykonywana jest operacja dodawania, a jej wynik jest wyświetlany na ekranie (wraz z wartościami obu argumentów). W dalszej części skryptu funkcja została wywołana z argumentami `11` i `21`, a zatem wynik działania skryptu będzie taki, jak zaprezentowany na rysunku 2.9.



Rysunek 2.9. *Wynik działania funkcji wykonującej dodawanie argumentów*

Zwracanie wartości przez funkcje

Przyjmowanie argumentów to nie jedyna cecha funkcji — mogą one również zwracać różne wartości. Czynność ta jest wykonywana za pomocą instrukcji `return`. Jeśli wystąpi ona wewnątrz funkcji, ta jest przerywana i zwraca wartość występującą po `return`. Schematycznie tego typu konstrukcja wygląda następująco:

```
funtion nazwa_funkcji(argumenty)
{
  //instrukcje wnętrza funkcji
  return wartość;
}
```

Jeśli wywołamy tego typu funkcję, to w miejscu jej wywołania zostanie wstawiona zwrócona przez nią wartość, która będzie mogła być wykorzystana w dalszej części skryptu. Warto też wiedzieć, że użycie samej instrukcji `return` bez żadnych argumentów również powoduje przerwanie działania funkcji (nie zwraca ona jednak wtedy żadnej wartości).

ĆWICZENIE

2.24 Zwracanie wyniku działania funkcji

Napisz funkcję przyjmującą dwa argumenty i zwracającą wynik ich dodawania. Wywołaj ją w skrypcie.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu

function dodaj(wart1, wart2)
{
  var wynik = wart1 + wart2;
  return wynik;
}
```

```
var wartosc = dodaj(11, 21);
document.write ("Wynikiem dodawania 11 + 21 jest " + wartosc + ".");

// Koniec kodu JavaScript -->
</script>
```

Funkcja `dodaj` przyjmuje argumenty `wart1` i `wart2`. W jej wnętrzu są one dodawane za pomocą arytmetycznego operatora `+`, a wynik tego działania jest przypisywany zmiennej `wynik`. Wartość tej zmiennej jest z kolei zwracana za pomocą instrukcji `return`. Zmienna `wynik` pełni tu jedynie funkcję pomocniczą zwiększającą czytelność prezentowanej techniki. Równie dobrze całą treścią funkcji `dodaj` mogłaby być jedna tylko instrukcja:

```
return wart1 + wart2;
```

W dalszej części kodu funkcja `dodaj` jest wywoływana z argumentami `11` i `21`, a wynik jej działania (czyli zwrócona przez nią wartość) jest przypisywana zmiennej `wartosc`, która jest następnie używana w instrukcji `document.write` do wyświetlania rezultatu na ekranie.

Instrukcja

```
wartosc = dodaj (11, 21);
```

działa następująco:

- Najpierw jest wywoływana funkcja `dodaj`, a zatem są wykonywane jej instrukcje;
- Następnie wynik działania funkcji `dodaj` jest podstawiany w miejscu jej wywołania. Ponieważ w tym przypadku jest to `32`, instrukcja jest traktowana jako `wartosc = 32`;
- Zmiennej `wartosc` jest przypisywana wartość zwrócona przez funkcję.

Wynik działania skryptu będzie więc taki sam, jak zostało to przedstawione na rysunku 2.9.

Zasięg zmiennych

Na zakończenie rozdziału omówimy jeszcze temat zasięgu zmiennych (używa się również terminu **widoczność zmiennych**). Zasięg możemy określić jako miejsca, w których zmienna jest ważna i można się do niej bezpośrednio odwoływać. Zmienna może być:

- globalna,
- lokalna.

Zasięg globalny

Zmienne globalne (o zasięgu globalnym) to takie, które są widoczne w każdym miejscu skryptu. Zmienna staje się globalną, jeśli jest zdefiniowana poza wnętrzami funkcji. Można się do niej odwoływać w dowolnym miejscu kodu, również we wnętrzach funkcji.

ĆWICZENIE

2.25 Odwołanie do zmiennej globalnej

Umieść w skrypcie zmienną globalną oraz dowolną funkcję. Spróbuj odczytać wartość zmiennej zarówno w funkcji, jak i poza nią.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu
var liczba = 124;

function func()
{
  document.write("Wartość zmiennej w funkcji: " + liczba);
}

func();
document.write("<br />");
document.write("Wartość zmiennej poza funkcją: " + liczba);

// Koniec kodu JavaScript -->
</script>
```

W skrypcie została zadeklarowana globalna zmienna `liczba` o wartości `124` oraz funkcja `func`, której zadaniem jest wyświetlenie wartości wymienionej zmiennej. Ponieważ zmienna ma zasięg globalny, jest to działanie prawidłowe, co pokazuje dalszy kod skryptu, w którym funkcja została wywołana i została również wyświetlona wartość zmiennej.

Zasięg lokalny

Zmienne o zasięgu lokalnym są definiowane wewnątrz funkcji, a ich zasięg jest ograniczony tylko do wnętrza funkcji, w której zostały zdefiniowane. Próba odwołania w innym miejscu skryptu spowoduje powstanie błędu.

ĆWICZENIE

2.26 Zasięg zmiennej lokalnej

Umieść w skrypcie dowolną funkcję i zadeklaruj w niej zmienną. Spróbuj odczytać wartość zmiennej zarówno w funkcji, jak i poza nią.

```
<script type="text/javascript">
<!-- Ukrycie przed przeglądarkami nie obsługującymi JavaScriptu

function func()
{
  var liczba = 124;
  document.write("Wartość zmiennej w funkcji: " + liczba);
}

func();
document.write("<br />");
document.write("Wartość zmiennej poza funkcją: " + liczba);

// Koniec kodu JavaScript -->
</script>
```

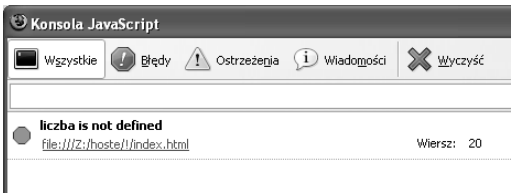
Powyższy skrypt, w przeciwieństwie do przedstawionego w ćwiczeniu 2.25, nie będzie działał prawidłowo. Skoro bowiem zmienna `liczba` została zadeklarowana we wnętrzu funkcji `func`, to jest zmienną lokalną i nie można się do niej odwoływać poza tą funkcją. Stąd instrukcja:

```
document.write("Wartość zmiennej poza funkcją: " + liczba);
```

nie będzie mogła być poprawnie wykonana, a na ekranie pojawi się tylko napis:

```
Wartość zmiennej w funkcji: 124
```

Jeśli używamy przeglądarki dysponującej konsolą JavaScript (jak np. FireFox, Opera), po wywołaniu konsoli (menu *Narzędzia\Konsola JavaScript*) zobaczymy, że faktycznie skrypt spowodował wystąpienie błędu (rysunek 2.10).



Rysunek 2.10. *Odwołanie do zmiennej lokalnej spowodowało wystąpienie błędu*