Marcin Lis

ĆWICZENIA PRAKTYCZNE

JavaScript

Na kłopoty JavaScript!

Podstawy, czyli do czego służy JavaScript i dlaczego jest aż tak ważny
Elementy języka i zależności między nimi, czyli jak sprawić, żeby to zadziałało
Bardzo dobra witryna, czyli jak wykorzystać wszystkie dostępne możliwości

Helion

Wydanie III

Redaktor prowadzący: Ewelina Burska
Projekt okładki: Maciej Pasek
Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

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

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?cwjas3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Przykłady omawiane w książce oraz kody źródłowe można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/cwjas3.zip

ISBN: 978-83-246-4796-5

Printed in Poland.

Spis treści

	Wstęp	**7**
Rozdział 1.	**Podstawy**	**11**
	Skrypty w kodzie strony	11
	Wyświetlanie informacji	14
	Używanie znaczników formatujących dane	17
	Gdy przeglądarka nie obsługuje skryptów	19
	JavaScript to NIE skrypty Javy!	20
Rozdział 2.	**Elementy języka**	**21**
	Komentarze	21
	Typy danych	23
	Zmienne	26
	Operatory	29
Rozdział 3.	**Instrukcje sterujące**	**45**
	Instrukcje warunkowe	45
	Operator warunkowy	50
	Instrukcja wyboru switch	51
	Pętle	54
	Przerywanie i kontynuowanie pętli	60
Rozdział 4.	**Funkcje**	**65**
	Definiowanie funkcji	65
	Argumenty	66
	Zwracanie wartości	69

Zasięg zmiennych 70
Funkcje zagnieżdżone (wewnętrzne) 74
Alternatywne definiowanie funkcji 76

Rozdział 5. Tablice 79

Tworzenie tablic 79
Odczyt i zapis tablic 81
Konstruktor tablicy 84
Wykonywanie operacji na tablicach 85

Rozdział 6. Programowanie obiektowe 95

Obiekty w JavaScripcie 95
Tworzenie obiektów za pomocą literałów 96
Konstruktor typu obiektowego 99
Metody obiektów 101
Iteracja po właściwościach 102
Funkcje to też obiekty 105
Prototypy, czyli dziedziczenie w JavaScripcie 107

Rozdział 7. Wyjątki 111

Zgłaszanie wyjątków 111
Przechwytywanie wyjątków 114
Blok finally 118

Rozdział 8. Obiekty i funkcje globalne 121

Funkcje globalne 121
Przetwarzanie wyrażeń 123
Przetwarzanie wartości liczbowych 125
Matematyka 129
Data i czas 134
Ciągi znaków 143

Rozdział 9. Współpraca z przeglądarką 147

Obiekty udostępniane przez przeglądarkę 147
Obiekt window 148
Obiekt document 156
Obiekt history 158
Obiekt location 159
Obiekt navigator 163

Rozdział 10. Elementy witryny (model DOM) 167
Jak przeglądarka „widzi" dokument? 167
Dostęp do elementów strony 170
Odczyt i zmiana atrybutów oraz stylów CSS 174
Odwołania do istniejących węzłów 177
Dynamiczne tworzenie fragmentów strony 180
Rozdział 11. Zdarzenia 183
Zdarzenia na stronie WWW 183
Zdarzenia jako właściwości elementów witryny 186
Rejestrowanie procedur obsługi 190
Usuwanie procedur obsługi 193
Obsługa klawiatury i myszy 195
Rozdział 12. Obsługa interfejsu użytkownika 199
Elementy witryny 199
Pola wyboru typu checkbox 201
Pola wyboru typu radio 202
Pola tekstowe (text) 205
Rozszerzone pola tekstowe (textarea) 206
Listy 209

2

Elementy języka

Komentarze

W treści skryptu można umieszczać jedynie taką treść, która będzie interpretowana przez przeglądarkę[1] jako instrukcje JavaScriptu. W przypadku bardziej skomplikowanego kodu warto jednak dodać komentarze objaśniające działanie poszczególnych jego fragmentów. Do dyspozycji są dwa rodzaje komentarzy: wierszowy i blokowy. Oba mają taką samą postać jak w innych typowych językach programowania opartych na składni wywodzącej się z języków C i C++.

Komentarz wierszowy (liniowy) zaczyna się od znaków `//` i obowiązuje do końca danej linii skryptu. Wszystko, co występuje po tych dwóch znakach, aż do końca bieżącej linii, jest ignorowane przez przeglądarkę przetwarzającą skrypt.

W ćwiczeniach z tego oraz kolejnych rozdziałów będzie prezentowany tylko właściwy kod skryptów JavaScript; kod HTML będzie pomijany, o ile nie jest niezbędny do funkcjonowania przykładu (w szczególności dotyczy to nagłówków stron). Listingi dostępne na FTP uwzględniają natomiast zawsze pełny kod strony, czyli zarówno HTML, jak i JavaScript.

[1] Przez zawarty w przeglądarce interpreter JavaScriptu. Mówi się również o „silniku" JavaScriptu (ang. *JavaScript engine*).

ĆWICZENIE

2.1 Użycie komentarza wierszowego

Użyj w kodzie skryptu komentarza wierszowego.

```
<body>
<script type="text/javascript">
// Wyświetlenie napisu powitalnego
document.write("Witam na mojej stronie!");
</script>
</body>
```

Komentarz blokowy rozpoczyna się od znaków /* i kończy znakami */. Wszystko, co znajduje się pomiędzy, jest pomijane przy przetwarzaniu kodu przez przeglądarkę. Komentarz blokowy można umieścić praktycznie w dowolnym miejscu skryptu, może on się nawet znaleźć w środku instrukcji (pod warunkiem że nie zostanie przedzielone żadne słowo). Komentarzy tego typu nie wolno natomiast zagnieżdżać, można jednak stosować wewnątrz nich komentarze wierszowe.

ĆWICZENIE

2.2 Wykorzystanie komentarza blokowego

Użyj w kodzie skryptu komentarza blokowego.

```
<body>
<script type="text/javascript">
/*
  Przykładowy skrypt wyświetlający treść strony.
  Powstanie warstwa wraz z akapitem tekstowym.
*/
document.write("<div>");
document.write("<p>To jest treść akapitu tekstowego.</p>");
document.write("</div>");
</script>
</body>
```

Typy danych

Typ danych to po prostu określenie rodzaju danych. Przykładowo, typ całkowitoliczbowy określa liczby całkowite. Występujące w JavaScripcie typy danych można podzielić następująco:

- typ liczbowy,
- typ łańcuchowy,
- typ logiczny,
- typ obiektowy,
- typy specjalne.

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 (stałych napisowych, z ang. *string constant*, *literal constant*) liczbowych, czyli ciągów znaków składających się na liczbę, np. `24` (umieszczony w kodzie skryptu tekst `24` to dwa znaki, dwójka i czwórka, które razem stanowią literał — stałą napisową — interpretowany jako liczba 24). Obowiązują przy tym następujące zasady:

- Jeżeli ciąg cyfr nie jest poprzedzony żadnym znakiem lub jest poprzedzony znakiem +, reprezentuje wartość dodatnią, jeżeli natomiast jest poprzedzony znakiem –, reprezentuje wartość ujemną.
- Jeżeli literał rozpoczyna się od cyfry *0*, 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 wykładniczej, 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 $X.Y * 10^Z$.

- W liczbach rzeczywistych separatorem dziesiętnym jest kropka (zamiast występującego w notacji polskiej przecinka, np. zapis `3.14` oznacza wartość 3,14, czyli trzy i czternaście setnych).

Przykłady literałów liczbowych:

- `123` — dodatnia całkowita wartość dziesiętna 123,
- `-123` — ujemna całkowita wartość dziesiętna –123,
- `012` — dodatnia całkowita wartość ósemkowa równa 10 w systemie dziesiętnym,
- `-024` — ujemna całkowita wartość ósemkowa równa 20 w systemie dziesiętnym,
- `0xFF` — dodatnia całkowita wartość szesnastkowa równa 255 w systemie dziesiętnym,
- `-0x0f` — ujemna całkowita wartość szesnastkowa równa –15 w systemie dziesiętnym,
- `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 (inaczej, typ *string*) 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"
```

Napisy mogą też zawierać sekwencje znaków specjalnych przedstawione w tabeli 2.1.

Znaki cudzysłowu (apostrofu) są tu wyróżnikami ciągu (napisu). Należy na to zwrócić uwagę, gdy sam ciąg ma zawierać jeden z tych znaków. Jeżeli w ciągu ma wystąpić znak użyty jako wyróżnik ciągu, znak ten musi być zastąpiony przez sekwencję specjalną.

Tabela 2.1. *Sekwencje znaków specjalnych*

Sekwencja	Znaczenie
\b	backspace (ang. *backspace*)
\n	nowa linia (ang. *new line*)
\r	powrót karetki (ang. *carriage return*)
\f	nowa strona (ang. *form feed*)
\t	tabulacja (ang. *horizontal tab*)
\"	cudzysłów (ang. *double quote*)
\'	apostrof (ang. *single quote*)
\\	lewy ukośnik (ang. *backslash*)

ĆWICZENIE

2.3 Wyróżniki ciągów

Użyj instrukcji `document.write` do wyświetlenia kilku napisów. Napisy powinny zawierać znaki będące wyróżnikami ciągów.

```
<body>
<script type="text/javascript">
// Wyróżnikami ciągu są znaki cudzysłowu.
// Konieczne jest użycie sekwencji specjalnej \"
document.write("<p>To jest cytat: \"Być albo nie być...\".</p>");

// Wyróżnikami ciągu są znaki apostrofu.
// Konieczne jest użycie sekwencji specjalnej \'
document.write(
  '<p>Prawa Murphy\'ego to zbiór humorystycznych powiedzeń.</p>');

// Sekwencji specjalnych nie trzeba używać po zamianie
// znaków wyróżniających ciąg.
document.write('<p>To jest cytat: "Być albo nie być...".<p>');
document.write(
  "<p>Prawa Murphy'ego to zbiór humorystycznych powiedzeń.<p>");
</script>
</body>
```

Typ logiczny

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

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.

Typy specjalne

Można wyróżnić dwa rodzaje typów specjalnych: `null` i `undefined`. Choć podobne, nie są tożsame. Otóż, `null` określa wartość pustą (czy też brak wartości) i najczęściej używany jest podczas korzystania z obiektów i programowania obiektowego. W tym kontekście, jeśli jakiś byt programistyczny (zmienna, obiekt, właściwość itd.) ma wartość `null`, oznacza to, że jest pusty.

Z kolei typ `undefined` określa wartość niezdefiniowaną. W tym kontekście wartość `undefined` ma niezainicjowana zmienna, zmienna, której jawnie przypisano wartość `undefined`, bądź też nieistniejąca właściwość obiektu.

Zmienne

Zmienna to miejsce w skrypcie, w którym można przechowywać dane, czyli liczby, napisy itp. Miejsce to oznaczone jest nazwą. A zatem każda zmienna ma swoją nazwę, która pozwala na jej jednoznaczną identyfikację w treści skryptu, inaczej mówiąc, „w kodzie”. Zmienna charakteryzuje się również typem, który określa rodzaj danych, jakie

przechowuje. W celu utworzenia zmiennej należy ją **zadeklarować**, tzn. podać nazwę oraz początkową wartość. Można to zrobić następująco[2]:

```
var nazwa_zmiennej = wartość;
```

W JavaScripcie, podobnie jak i w wielu innych skryptowych językach programowania, nie określa się jawnie typu zmiennej, a więc 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.

Nazwa zmiennej może zawierać litery, cyfry i znak podkreślenia[3]. Wolno stosować zarówno wielkie, jak i małe litery, są też one rozróżniane, co oznacza, że przykładowo: `liczba` i `Liczba` to nazwy dwóch różnych zmiennych. Dopuszczalne są również znaki narodowe, np. polskie litery — wielu programistów korzysta jednak wyłącznie z liter alfabetu łacińskiego, nawet gdy tworzą nazwy pochodzące z języka narodowego. Nazwa zmiennej nie może też zaczynać się od cyfry.

ĆWICZENIE

2.4 Użycie zmiennych w skrypcie

Zadeklaruj dwie zmienne, przypisz im dowolne ciągi znaków i wyświetl ich wartości na ekranie.

```
<script type="text/javascript">
var zmienna1 = "Mój komputer";
var zmienna2 = 2;
document.write(zmienna1 + " ma dysk o pojemności " + zmienna2 + " TB.");
</script>
```

Po wczytaniu takiej strony na ekranie ukaże się napis *Mój komputer ma dysk o pojemności 2 TB*. Jak działa ten skrypt? Zadeklarowane zostały dwie zmienne o nazwach `zmienna1` i `zmienna2`. Pierwszej z nich przypisano ciąg znaków `Mój komputer`, a drugiej — wartość liczbową, dodatnią liczbę całkowitą `2`. Obu zmiennych użyto następnie w instrukcji `docu ➥ment.write`, wyświetlającej dane na ekranie. Konieczne było przy tym połączenie napisów, tak aby otrzymać jeden ciąg, który ukazał się na

[2] Można też zastosować konstrukcję bez słowa `var`: *`nazwa_zmiennej = wartość;`*. Nie są to jednak sposoby w pełni równoważne. Zostanie to bliżej wyjaśnione w jednym z kolejnych rozdziałów.

[3] Dopuszczalny jest również znak *$*, jednak lepiej go unikać przy nazywaniu własnych zmiennych.

ekranie. Posłużył do tego znak + (operator łączenia łańcuchów; patrz też podrozdział „Operatory"). Zostało więc wykonane łączenie, inaczej konkatenacja, łańcuchów znakowych.

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

ĆWICZENIE

2.5 Zmiana typu zmiennej

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

```
<script type="text/javascript">
var zmienna1 = "To jest przykładowy tekst.";
document.write("Pierwsza wartość zmiennej zmienna1: " + zmienna1);
zmienna1 = 24.7;
document.write("<br />Druga wartość zmiennej zmienna1: " + zmienna1);
</script>
```

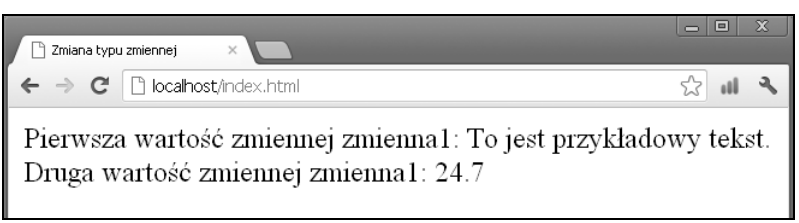
Efekt działania skryptu jest widoczny na rysunku 2.1. Na początku została zadeklarowana zmienna `zmienna1` i przypisano jej ciąg znaków:

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

który dzięki instrukcji `document.write` pojawił się 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 2.1. *Efekt działania kodu z ćwiczenia 2.5*

Operatory

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

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

Można więc wykonywać operacje dodawania, odejmowania, porównywania, przypisania i wiele innych.

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 2.2. W tej grupie występują jednak również operatory inkrementacji (zwiększania) i dekrementacji (zmniejszania). Operatory `*`, `/`, `+`, `-`, `%` są dwuargumentowe, natomiast `++` i `--` są jednoargumentowe.

Tabela 2.2. *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`

ĆWICZENIE

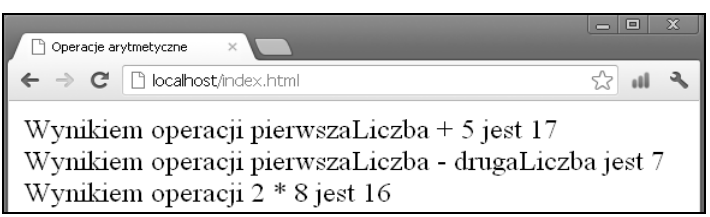
2.6 Wykonywanie operacji arytmetycznych

Umieść w skrypcie kilka zmiennych, użyj ich do wykonania standardowych operacji arytmetycznych (jedną operację arytmetyczną wykonaj bez używania zmiennych). Wyświetl wyniki na ekranie.

```
<script type="text/javascript">
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);
</script>
```

Wynik działania przedstawionego skryptu z pewnością nie jest żadnym zaskoczeniem (rysunek 2.2). 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 2.2. *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 tak: ++x, natomiast przyrostkowa tak: 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

2.7 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">
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);
</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 2.3. Dlaczego? Otóż w wierszu 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 2.3. *Wynik ćwiczenia dotyczącego operatora dekrementacji*

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

ĆWICZENIE

2.8 Operator dekrementacji

Zmień kod z ćwiczenia 2.7 tak, aby operator `++` został zastąpiony operatorem `--`. Następnie przeanalizuj działanie skryptu i sprawdź, czy otrzymany wynik jest taki sam jak na ekranie przeglądarki.

```
<script type="text/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);
</script>
```

Wynikiem działania skryptu będzie ciąg znaków `11 11 10 10 8`. W wierszu 1. najpierw wartość `x` jest zmniejszana 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 2.3. Operatory relacyjne będą używane w rozdziale 3., w którym omówiono instrukcje warunkowe. Tam też zostaną przedstawione konkretne przykłady ich wykorzystania.

Operatory bitowe

Operatory bitowe pozwalają na wykonywanie operacji na poszczególnych bitach liczb i zostały przedstawione w tabeli 2.4. 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 porównywania*

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

Iloczyn bitowy

Iloczyn bitowy to operacja powodująca, że włączone pozostają tylko te bity, które były włączone w obu argumentach. Wynik operacji `AND` na pojedynczych bitach zobrazowano w tabeli 2.5. Jeśli zatem wykonamy operację:

```
var liczba = 179 & 38;
```

to zmiennej `liczba` zostanie przypisana wartość 34.

Tabela 2.4. *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`

Tabela 2.5. *Wyniki operacji AND dla pojedynczych bitów*

Argument 1	Argument 2	Wynik
1	1	1
1	0	0
0	1	0
0	0	0

Dlaczego 34? Najłatwiej pokazać to, jeśli obie wartości (czyli 179 i 38) przedstawi się w postaci dwójkowej. 179 w postaci dwójkowej to `10110011`, natomiast 38 to `00100110`. Operacja `AND` będzie zatem miała postać:

```
10110011 (179)
00100110 (38)
--------
00100010 (34)
```

Wynikiem jest więc 34.

Suma bitowa

Suma bitowa to operacja powodująca, że pozostają włączone te bity, które były włączone przynajmniej w jednym z argumentów. Wynik operacji `OR` na pojedynczych bitach można zobaczyć w tabeli 2.6. Jeśli zatem wykonamy operację:

Tabela 2.6. Wyniki operacji OR dla pojedynczych bitów

Argument 1	Argument 2	Wynik
1	1	1
1	0	1
0	1	1
0	0	0

```
var liczba = 34 | 65;
```

to zmiennej `liczba` zostanie przypisana wartość 99.

Jeśli obie liczby rozpiszemy w postaci dwójkowej, otrzymamy `00100010` (34) i `01000001` (65). Zatem całe działanie będzie miało postać:

```
00100010 (34)
01000001 (65)
--------
01100011 (99)
```

Negacja bitowa

Negacja bitowa powoduje zmianę stanu bitów. A zatem tam, gdzie dany bit miał wartość `0`, będzie miał `1`, natomiast tam, gdzie miał wartość `1`, będzie miał `0`. Działanie operacji `NOT` na pojedynczych bitach zobrazowano w tabeli 2.7.

Tabela 2.7. Wyniki operacji NOT dla pojedynczych bitów

Argument	Wynik
1	0
0	1

Bitowa różnica symetryczna

Bitowa różnica symetryczna, czyli operacja `XOR`, powoduje, że włączone zostają te bity, które miały różne stany w obu argumentach, a pozostałe zostają wyłączone. Wynik operacji `XOR` na pojedynczych bitach można zobaczyć w tabeli 2.8.

Tabela 2.8. *Wyniki operacji XOR dla pojedynczych bitów*

Argument 1	Argument 2	Wynik
1	1	0
1	0	1
0	1	1
0	0	0

Wykonanie przykładowej operacji:

```
var liczba = 34 ^ 118;
```

spowoduje przypisanie zmiennej `liczba` wartości 84. Jeśli bowiem zapiszemy obie wartości w postaci dwójkowej, to 34 przyjmie postać `00100010`, natomiast 118 — `01110110`. Operacja `XOR` będzie zatem wyglądała następująco:

```
00100010 (34)
01110110 (118)
--------
01010100 (84)
```

Przesunięcie bitowe w lewo

Przesunięcie bitowe w lewo to operacja polegająca na przesunięciu wszystkich bitów argumentu znajdującego się z lewej strony operatora w lewo o liczbę miejsc wskazaną przez argument znajdujący się z jego prawej strony. Wykonanie przykładowej operacji:

```
var liczba = 84 << 1;
```

spowoduje przypisanie zmiennej `liczba` wartości 168. Działanie `84 << 1` oznacza bowiem: „Przesuń wszystkie bity wartości 84 o jedno miejsce w lewo”. Skoro 84 w postaci dwójkowej ma postać `01010100`, to po przesunięciu powstanie `10101000`, czyli 168.

Warto zauważyć, że przesunięcie bitowe w lewo odpowiada mnożeniu wartości przez wielokrotność liczby 2. I tak przesunięcie w lewo o jedno miejsce to pomnożenie przez 2, o dwa miejsca — przez 4, o trzy miejsca — przez 8 itd.

Przesunięcie bitowe w prawo

Analogicznie do powyższego, **przesunięcie bitowe w prawo** polega na przesunięciu wszystkich bitów argumentu znajdującego się z lewej strony operatora w prawo o liczbę miejsc wskazaną przez argument, który znajduje się z prawej strony operatora. A zatem wykonanie operacji:

```
var liczba = 84 >> 1;
```

spowoduje przypisanie zmiennej `liczba` wartości 42. Oznacza to bowiem przesunięcie wszystkich bitów wartości `01010100` (84 dziesiętnie) o jedno miejsce w prawo, czyli powstanie wartości `00101010` (42 dziesiętnie).

Tu również należy zwrócić uwagę, że przesunięcie bitowe w prawo odpowiada podzieleniu wartości przez wielokrotność liczby 2, czyli przesunięcie w prawo o jedno miejsce to podzielenie przez 2, o dwa miejsca — przez 4, o trzy miejsca — przez 8 itd. (należy jednak pamiętać, że jeżeli dzielona liczba będzie nieparzysta, w wyniku takiego dzielenia zostanie utracona część ułamkowa).

Operatory logiczne

Operacje logiczne mogą być wykonywane na argumentach, które posiadają wartość logiczną: prawda (`true`) lub fałsz (`false`). Operatory logiczne zostały przedstawione w tabeli 2.9. Operatory `&&` i `||` są dwuargumentowe, natomiast operator `!` jest jednoargumentowy.

Tabela 2.9. *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`. Zobrazowano to w tabeli 2.10.

Tabela 2.10. *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`. Zobrazowano to w tabeli 2.11.

Tabela 2.11. *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`. Zobrazowano to w tabeli 2.12.

Tabela 2.12. *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. Taka najprostsza operacja była już wykonywana w ćwiczeniach, odbywa się ona przy wykorzystaniu operatora = (równa się). Napisanie `liczba = 10` oznacza po prostu, 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
```

tłumaczy się jako:

```
liczba = liczba + 5
```

co oznacza: przypisz zmiennej `liczba` wartość wynikającą z dodawania `liczba + 5` lub — jeszcze prościej — zwiększ wartość zmiennej `liczba` o 5.

W JavaScripcie występuje cała grupa tego typu operatorów, zostały one zebrane w tabeli 2.13.

Operator łączenia łańcuchów znakowych

Jak zostało to wspomniane wyżej (ćwiczenie 2.4), do łączenia łańcuchów znakowych (napisów) służy operator zapisywany jako + (mówimy o konkatenacji łańcuchów znakowych). Łatwo jednak zauważyć, że ten sam znak jest jednocześnie symbolem operacji arytmetycznego dodawania (tabela 2.2). Sytuacja jest jasna, gdy dodajemy dwa ciągi lub dwie liczby. W pierwszym przypadku wykonana będzie konkatenacja (łączenie), np.:

```
var str = "abc" + "def";
```

Tabela 2.13. *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

a w drugim — dodawanie arytmetyczne:

```
var liczba = 123 + 456;
```

Co się jednak stanie, gdy spróbujemy dodać liczbę do ciągu znaków lub ciąg znaków do liczby? Można to sprawdzić, wykonując ćwiczenie 2.9.

ĆWICZENIE

2.9 Dodawanie i konkatenacja

Napisz skrypt, w którym zostaną wykonane różne wersje dodawania liczb i ciągów znakowych. Sprawdź otrzymane wyniki.

```
<script type="text/javascript">
var str = "abc" + "def";
var liczba = 123 + 456;
var zmienna1 = "abc" + 123;
var zmienna2 = 123 + "def";
var zmienna3 = "123" + "456";
document.write('Wynik dodawania "abc" + "def": ' + str + "<br />");
document.write('Wynik dodawania 123 + 456: ' + liczba  + "<br />");
document.write('Wynik dodawania "abc" + 123: ' + zmienna1  + "<br />");
document.write('Wynik dodawania 123 + "def": ' + zmienna2  + "<br />");
document.write('Wynik dodawania "123" + "456": ' + zmienna3  + "<br />");
</script>
```

Po uruchomieniu powyższego skryptu okaże się, że dodawanie arytmetyczne dotyczyło tylko drugiego przypadku (`var liczba = 123 + 456;`). We wszystkich pozostałych wykonane zostało łączenie łańcuchów znakowych. Oznacza to, że po wykryciu, iż jednym z argumentów operatora + jest ciąg znaków, drugi argument zawsze konwertuje się również na ciąg znaków i wykonywana jest operacja łączenia tych ciągów.

Pozostałe operatory

W JavaScripcie występuje jeszcze kilka innych operatorów, które jednak nie będą osobno omawiane. Są to m.in. operator indeksowania tablic, wywołania funkcji, rozdzielania wyrażeń, tworzenia obiektów itp. Pojawią się one w dalszej części książki w trakcie omawiania kolejnych tematów, zostały też uwzględnione w tabeli prezentującej priorytety operatorów (np. w rozdziale 3., w części dotyczącej instrukcji warunkowych, zostanie przedstawiony 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. Przedstawiono to w tabeli 2.14. Im wyższa pozycja w tabeli, tym wyższy priorytet operatora. Operatory znajdujące się na jednym poziomie (w jednym wierszu) mają ten sam priorytet[4].

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

Tabela 2.14. *Priorytety operatorów*

L.p.	Operatory	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	`<`, `>`, `<=`, `>=`, `instanceof`
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	`,`

JavaScript. ĆWICZENIA PRAKTYCZNE

Wydanie III

JavaScript – musisz go poznać!

JavaScript jest dziś tak głęboko zakorzeniony w świecie witryn internetowych, że mało prawdopodobne jest, by coś mogło mu zagrozić – chyba że usuniemy z nich HTML, wyrzucimy do kosza biblioteki takie jak jQuery i uznamy, że czas na rewolucję. Na razie jednak zanosi się na to, że JavaScript będzie nam towarzyszyć i ułatwiać korzystanie z internetu jeszcze przez wiele długich lat. Dzięki temu językowi możemy przecież dokonać mnóstwa operacji na stronach WWW i sprawić, by były one znacznie ciekawsze. Pora więc w końcu nauczyć się przynajmniej jego podstaw!

Jeśli chcesz zrobić to szybko i bez nerwów, a ponadto od razu wykorzystać swoją wiedzę w praktyce, sięgnij po tę książkę. Znajdziesz w niej podstawowe wiadomości o najnowszej wersji języka i zaczniesz samodzielnie tworzyć kod. Kolejne ćwiczenia zaznajomią Cię ze skryptami i elementami JavaScriptu i pozwolą Ci opanować zagadnienia związane z instrukcjami, funkcjami czy tablicami. Dowiesz się, jak wygląda programowanie obiektowe w tym języku, jak obsłużyć wyjątki, do czego przydają się obiekty i funkcje globalne oraz jak wykorzystać zdarzenia. Potem zgłębisz zasady współpracy z przeglądarką i z zamkniętymi oczami będziesz mógł tworzyć niezbędne elementy witryny oraz interfejsu użytkownika. Trzy, dwa, jeden... kup!

- Skrypty w kodzie strony, wyświetlanie informacji, używanie znaczników
- Komentarze, typy danych, zmienne i operatory
- Instrukcje sterujące
- Funkcje
- Tablice
- Programowanie obiektowe
- Wyjątki
- Obiekty i funkcje globalne
- Współpraca z przeglądarką
- Elementy witryny (model DOM)
- Zdarzenia
- Obsługa interfejsu użytkownika

helion.pl
księgarnia internetowa

Nr katalogowy: 12308

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

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

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

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-246-4796-5

9 788324 647965

Cena 34,90 zł