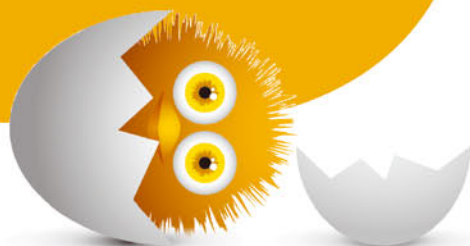


JavaScript

PRZEWODNIK
DLA ABSOLUTNIE
POCZĄTKUJĄCYCH



Tytuł oryginału: JavaScript Absolute Beginner's Guide

Tłumaczenie: Lech Lachowski

ISBN: 978-83-283-3180-8

Authorized translation from the English language edition, entitled: JAVASCRIPT ABSOLUTE BEGINNER'S GUIDE, First Edition; ISBN 0789758067; by Kirupa Chinnathambi; published by Pearson Education, Inc, publishing as QUE Publishing.
Copyright © 2017 by Pearson Education.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education Inc.

Polish language edition published by HELION S.A.. Copyright © 2017.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/javpap.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/javpap>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Wprowadzenie	13
1 Witaj, świecie!	17
Co to jest JavaScript?	19
Prosty przykład	20
Narzędzia do edycji kodu	20
Dokument HTML	20
Analiza kodu — polecenia i funkcje	22

Część I. Podstawy

2 Wartości i zmienne	25
Używanie zmiennych	26
Więcej informacji na temat zmiennych	27
Nazywanie zmiennych	27
Więcej informacji na temat deklarowania i inicjowania zmiennych	28
3 Funkcje	31
Czym jest funkcja?	33
Prosta funkcja	33
Tworzenie funkcji przyjmującej argumenty	35
Tworzenie funkcji zwracającej dane	37
Słowo kluczowe return	38
Wcześniejsze wychodzenie z funkcji	38
4 Instrukcje warunkowe: if, else i switch	41
Instrukcja if-else	42
Poznaj operatory warunkowe	44
Tworzenie bardziej złożonych wyrażeń	45
Wariacje na temat instrukcji if-else	46
Samodzielna instrukcja if	46
Przerażająca instrukcja if-else/if-else	47
Uff	47

Instrukcje switch	48
Korzystanie z instrukcji switch	48
Podobieństwo do instrukcji if-else	50
Wybór właściwej instrukcji	51
5 Poznaj pętle: for, while oraz do...while!	53
Pętla for	54
Warunek rozpoczynający	56
Warunek kończący (czyli czy już skończyliśmy?)	56
Docieramy do końca	57
Zbierając wszystko razem	57
Kilka przykładów pętli for	58
Wcześniejsze przerywanie pętli	58
Pomijanie iteracji	58
Odliczanie wstecz	59
Nie trzeba używać liczb	59
Tablica! Tablica! Tablica!	59
O nie, nie zrobił tego!	60
Pozostałe pętle	60
Pętla while	60
Pętla do...while	61
6 Timery	63
Poznaj trzy timery	64
Opóźnianie za pomocą funkcji setTimeout	64
Zapętlenie za pomocą funkcji setInterval	65
Płynne animowanie za pomocą funkcji requestAnimationFrame	66
7 Zakres zmiennych	69
Zakres globalny	70
Zakres lokalny	71
Różne krętownice przy ustalaniu zakresu	72
Deklaracje wykorzystujące słowo kluczowe var nie obsługują zakresu bloku	72
Jak JavaScript przetwarza zmienne	74
Domknięcia	75

8	Domknięcia	77
	Funkcje wewnątrz funkcji	78
	Gdy funkcje wewnętrzne nie są samowystarczalne	81
9	Gdzie należy umieścić kod?	87
	Wszystkie opcje na stół	88
	Podejście nr 1: cały kod jest umieszczony w dokumencie HTML	90
	Podejście nr 2: kod znajduje się w osobnym pliku	91
	Plik JavaScript	91
	Odwoływanie się do pliku JavaScript	92
	Które podejście zastosować?	94
	Tak, mój kod będzie używany w wielu dokumentach	94
	Nie, mój kod jest używany tylko raz w pojedynczym dokumencie HTML	95
10	Komentowanie kodu	97
	Czym są komentarze?	98
	Komentarze jednoliniowe	99
	Komentarze wieloliniowe	99
	Dobre praktyki komentowania	101

Część II. Świat pełen obiektów

11	O pizzy, typach, prymitywach i obiektach	103
	Najpierw porozmawiajmy o pizzy	104
	Od pizzy do języka JavaScript	106
	Czym są obiekty?	108
	Predefiniowane obiekty na swobodzie	109
12	Łańcuchy znaków	111
	Podstawy	112
	Właściwości i metody obiektu String	114
	Uzyskiwanie dostępu do poszczególnych znaków	114
	Łączenie (czyli konkatencja) łańcuchów znaków	115
	Tworzenie podłańcuchów z łańcuchów znaków	116
	Dzielenie łańcucha znaków — metoda split	117
	Szukanie czegoś wewnątrz łańcucha znaków	118
	Łańcuchy znaków z wielkimi i małymi literami	119

13	Kiedy prymitywy zachowują się jak obiekty	121
	Łańcuchy znaków nie są jedynym problemem	122
	Tak czy inaczej, przyczepmy się do łańcuchów znaków	122
	Dlaczego ma to znaczenie	124
14	Tablice	127
	Tworzenie tablicy	128
	Uzyskiwanie dostępu do wartości tablicy	129
	Dodawanie elementów do tablicy	130
	Usuwanie elementów z tablicy	132
	Szukanie elementów w tablicy	134
	Scalanie tablic	134
15	Liczby	137
	Używanie liczb	138
	Operatory	138
	Wykonywanie prostych działań matematycznych	139
	Inkrementacja i dekrementacja	139
	Wartości specjalne — nieskończoność i NaN	141
	Nieskończoność	141
	NaN	141
	Obiekt Math	142
	Stałe	142
	Zaokrąglanie liczb	143
	Funkcje trygonometryczne	144
	Potęgi i pierwiastki kwadratowe	145
	Uzyskanie wartości bezwzględnej	145
	Liczby losowe	145
16	Głębsze spojrzenie na obiekty	147
	Poznaj typ Object	148
	Tworzenie obiektów	149
	Określanie właściwości	152
	Tworzenie niestandardowych obiektów	154
	Słowo kluczowe this	158

17	Rozszerzanie obiektów wbudowanych	161
	Przywitaj się ponownie z prototypem — w pewnym sensie	162
	Rozszerzanie wbudowanych obiektów jest kontrowersyjne	166
	Nie kontrolujemy przeszłości wbudowanego obiektu	166
	Niektóre funkcjonalności nie powinny być rozszerzane ani nadpisywane	166
18	Wartości logiczne i ściślejsze operatory === oraz !==	169
	Obiekt Boolean	170
	Funkcja Boolean	170
	Operatory identyczności i nieidentyczności	172
19	Wartości null i undefined	175
	Null	176
	Undefined	177
20	Natychmiastowo wywoływane wyrażenia funkcyjne	179
	Pisanie prostego IIFE	181
	Pisanie IIFE, które przyjmuje argumenty	182
	Kiedy używać IIFE	183
	Unikanie kolizji kodu	183
	Domknięcia i stan zablokowania	184
	Zapewnianie prywatności elementów	188

Część III. Praca z DOM

21	JS, przeglądarka i model DOM	193
	Do czego służą HTML, CSS i JavaScript	194
	HTML definiuje strukturę	194
	Upiększ mój świat, CSS!	196
	Nadszedł czas na JavaScript!	196
	Poznaj obiektowy model dokumentu	197
	Obiekt window	199
	Obiekt document	200
22	Szukanie elementów w strukturze DOM	203
	Poznaj rodzinę querySelector	204
	Funkcja querySelector	204
	Funkcja querySelectorAll	205
	To naprawdę jest składnia selektora CSS	206

23	Modyfikowanie elementów DOM	209
	Elementy DOM są obiektami — w pewnym sensie!	210
	Przejdźmy do modyfikowania elementów DOM	212
	Zmiana wartości tekstowej elementu	213
	Wartości atrybutów	214
24	Nadawanie stylu zawartości	217
	Dlaczego należy ustawiać style przy użyciu JavaScriptu?	218
	Przypowieść o dwóch podejściach nadawania stylów	218
	Ustawianie stylu bezpośrednio	219
	Dodawanie i usuwanie klas za pomocą classList	220
	Dodawanie wartości klas	220
	Usuwanie wartości klas	220
	Przełączanie wartości klas	221
	Sprawdzanie, czy wartość klasy istnieje	221
	Idąc dalej	221
25	Przechodzenie przez drzewo DOM	223
	Szukanie drogi	224
	Obchodzenie się z rodzeństwem i rodzicami	226
	Zróbmy sobie dzieci!	227
	Zbierają wszystko razem	228
	Sprawdzanie, czy istnieje element potomny	228
	Uzyskiwanie dostępu do wszystkich elementów potomnych	228
	Przechodzenie przez DOM	229
26	Tworzenie i usuwanie elementów DOM	231
	Tworzenie elementów	232
	Usuwanie elementów	237
	Klonowanie elementów	238
27	Narzędzia dla programistów wbudowane w przeglądarki	243
	Poznaj Narzędzia dla programistów	244
	Badanie struktury DOM	246
	Debugowanie kodu JavaScript	249
	Poznaj konsolę	254
	Badanie obiektów	255
	Rejestrowanie wiadomości	256

Część IV. Obsługa zdarzeń

28	Zdarzenia	259
	Czym są zdarzenia?	260
	Zdarzenia i JavaScript	262
	1. Nasłuchiwanie zdarzeń	262
	2. Reagowanie na zdarzenia	264
	Prosty przykład	264
	Argumenty zdarzenia oraz typ Event	266
29	Bąbelkowanie i przechwytywanie zdarzeń	269
	Zdarzenie wędruje w dół, a potem w górę	270
	Poznaj fazy	273
	Kogo to obchodzi?	275
	Przerwanie zdarzenia	276
30	Zdarzenia myszy	281
	Poznaj zdarzenia myszy	282
	Pojedyncze i dwukrotne kliknięcie	282
	Zdarzenia mouseover i mouseout	284
	Przypominające kliknięcie zdarzenia wciskania i zwalniania przycisku myszy	285
	Zdarzenie podsłuchane ponownie... i ponownie... i ponownie	286
	Menu kontekstowe	287
	Właściwości obiektu MouseEvent	288
	Globalna pozycja myszy	288
	Pozycja myszy w przeglądarce	289
	Wykrywanie, który przycisk został kliknięty	289
	Obsługa kółka myszy	291
31	Zdarzenia klawiatury	293
	Poznaj zdarzenia klawiatury	294
	Używanie tych zdarzeń	295
	Właściwości zdarzenia klawiatury	296
	Kilka przykładów	296
	Sprawdzanie, czy naciśnięty został konkretny klawisz	297
	Wykonanie jakiejś akcji po naciśnięciu klawiszy strzałek	297
	Wykrywanie naciśnięcia wielu klawiszy	298

32	Zdarzenia ładowania strony i inne kwestie	301
	Co się dzieje podczas ładowania strony	302
	Etap numero uno	303
	Etap numero dos	303
	Etap numero trzy	304
	Zdarzenia DOMContentLoaded i load	304
	Skrypty i ich lokalizacja w strukturze DOM	306
	Elementy script — atrybuty async i defer	308
	Atrybut async	308
	Atrybut defer	309
33	Obsługa zdarzeń dla wielu elementów	311
	Jak to wszystko zrobić?	314
	Okropne rozwiązanie	315
	Dobre rozwiązanie	315
	Podsumowanie	318
34	Podsumowanie	321
	Słowniczek	323
	Skorowidz	326

W TYM ROZDZIALE

- nauczysz się, jak wielokrotnie uruchamiać jakiś kod;
- popracujesz z pętlami `for`, `while` oraz `do...while`.



5

POZNAJ PĘTLE: FOR, WHILE ORAZ DO...WHILE!

Gdy coś kodujemy, czasem chcemy powtórzyć pewną akcję lub uruchomić jakiś kod kilka razy. Załóżmy, że mamy na przykład funkcję o nazwie `saySomething` (powiedz coś), którą chcemy wywołać 10 razy.

Jednym ze sposobów, żeby to zrobić, jest po prostu wywołanie tej funkcji 10 razy przy użyciu kopiowania i wklejania:

```
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
saySomething();
```

Ten sposób działa i realizuje to, co zaplanowaliśmy, ale nie należy robić czegoś takiego. Powielanie kodu nigdy nie jest dobrym pomysłem.

Nawet jeśli zdecydujesz się kilka razy ręcznie powielić jakiś kod, takie podejście w praktyce się nie sprawdza. Często nie będziesz wiedzieć, ile razy trzeba uruchomić kod. Liczba uruchomień kodu będzie zależać od pewnego czynnika zewnętrznego, takiego jak liczba elementów w kolekcji danych, wynik otrzymany z wywołania usługi internetowej, liczba liter w słowie, oraz od różnych innych rzeczy, które ciągle się zmieniają. Nie zawsze będzie to stała liczba, taka jak 10. Jest więcej niż prawdopodobne, że liczba powtórzeń jakiegoś kodu może być bardzo, BARDZO duża. Nie chcemy kopiować i wklejać czegoś kilkaset lub kilka tysięcy razy, aby powtórzyć jakiś kod. To byłoby straszne.

Potrzebujemy ogólnego rozwiązania dla powtarzania kodu, umożliwiającego kontrolę nad tym, jak wiele razy kod jest powtarzany. W języku JavaScript takie rozwiązanie jest dostarczane w formie zwanej **pętlą**. Możemy tworzyć trzy rodzaje pętli:

- pętla `for`;
- pętla `while`;
- pętla `do...while`.

Każda z tych trzech odmian pętli umożliwia określenie kodu, który chcemy powtórzyć (czyli zapętlić) oraz sposobu zatrzymywania powtarzania, gdy spełniony zostanie pewien warunek. W kolejnych podrozdziałach dowiesz się, jak z nich korzystać.

Do dzieła!

Pętla for

Jednym z najbardziej powszechnych sposobów tworzenia pętli jest użycie instrukcji `for` w celu utworzenia tzw. pętli `for`. Pętla `for` pozwala na wielokrotne uruchamianie jakiegoś kodu, dopóki określone przez nas wyrażenie nie zwróci `false`. Prawdopodobnie na razie nie ma to dla Ciebie zbyt wiele sensu, więc spróbujmy wyjaśnić tę definicję, analizując przykład.

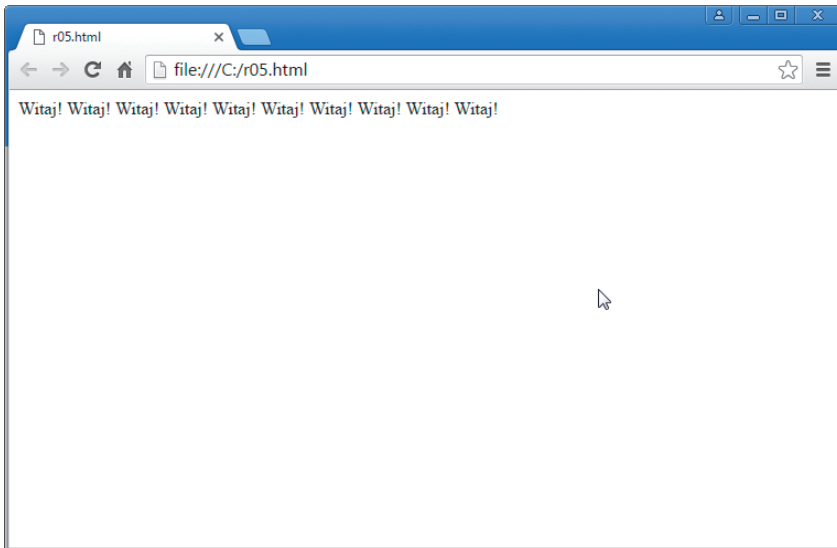
Gdybyśmy mieli przetłumaczyć nasz wcześniejszy przykład funkcji `saySomething`, używając instrukcji `for`, wyglądałoby to następująco:

```
var count = 10;

function saySomething() {
    document.writeln("Witaj!");
}

for (var i = 0; i < count; i++) {
    saySomething();
}
```

Jeśli umieścimy ten kod pomiędzy znacznikami `script` i uruchomimy go w przeglądarce, zobaczymy coś podobnego do tego, co zostało pokazane na rysunku 5.1.



RYСУNEK 5.1.

Słowo „Witaj!” zostanie powtórzono dziesięć razy na stronie

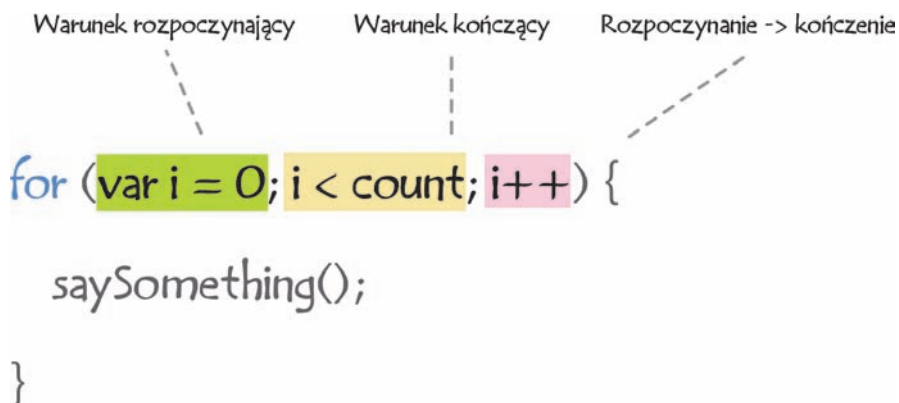


UWAGA Słowa są wyświetlane w przeglądarce, ponieważ użyliśmy metody `document.writeln`. Jest to kolejna metoda (poza `alert`), którą będziemy wykorzystywać, aby szybko coś przetestować. Metoda `document.writeln` przyjmuje jako argumenty cokolwiek, co chcemy wypisać, a gdy zostaje uruchomiona, to wszystko, co znajduje się na stronie, zastępuje zawartością wybraną przez nas do wyświetlenia.

Jest to możliwe dzięki pętli `for`, więc wyrazimy swoje podziękowania, poznając szczegółowo sposób jej działania. Po pierwsze, przyjrzyjmy się naszej gwiazdce:

```
for (var i = 0; i < count; i++) {
    saySomething();
}
```

Jest to pętla for. Prawdopodobnie bardzo różni się od pozostałych instrukcji, jakie widziałeś do tej pory, a to dlatego, że tak naprawdę jest zupełnie inna. Aby zrozumieć różnice, sprowadźmy pętlę for do postaci ogólnej przedstawionej na rysunku 5.2.



RYSUNEK 5.2.

Nasza pętla for i trzy regiony, na których się skupimy

Każdy z tych trzech zaznaczonych na rysunku innym kolorem regionów odgrywa bardzo ważną rolę w działaniu pętli for. Aby móc prawidłowo korzystać z pętli for, musisz wiedzieć, jakie funkcje pełni każdy region. To prowadzi nas do...

Warunek rozpoczynający

W pierwszym regionie definiujemy **warunek rozpoczynający**. Typowy warunek rozpoczynający zwykle obejmuje deklarowanie i inicjowanie zmiennej. W naszym przykładzie utworzyłem nową zmienną o nazwie `i` oraz zainicjowałem ją do liczby 0:

```
for (var i = 0; i < count; i++) {
  saySomething();
}
```

Gdybyś się nad tym zastanawiał, to informuję Cię, że nazwą zmiennej w takich przypadkach jest tradycyjnie pojedyncza litera, a najczęściej jest to litera `i`. Wartością, do której inicjowana jest ta zmienna, jest również tradycyjnie 0. Istnieje jeden naprawdę dobry powód, aby użyć liczby 0, ale wytłumaczę to nieco później.

Warunek kończący (czyli czy już skończyliśmy?)

Gdy zdefiniujemy nasz punkt początkowy, w kolejnym regionie możemy określić **warunek kończący**. Zasadniczo jest to fantazyjny sposób na powiedzenie tego, jak długo należy wykonywać pętlę. Jest to obsługiwane przez wyrażenie warunkowe (takie jakie widziałeś w poprzednim rozdziale), które zwraca wartość `true` lub `false`. W naszym przykładzie warunek jest taki, że zmienna `i` ma być mniejsza niż wartość `count`, który wynosi od 10:

```
for (var i = 0; i < count; i++) {  
    saySomething();  
}
```

Jeśli zmienna `i` jest mniejsza niż 10, wyrażenie ewaluuje do wartości `true` i pętla jest wykonywana dalej. Jeśli zmienna `i` jest równa lub większa niż 10, warunek ma wartość `false`, a pętla zostaje przerwana. Być może zastanawiasz się teraz, co powoduje, że wartość zmiennej `i` faktycznie zmienia się z jej wartości początkowej 0. To omówimy w kolejnym punkcie.

Docieramy do końca

Do tej pory poznaliśmy punkt początkowy oraz wyrażenie, które musi ewaluować do `false`, jeśli chcemy przerwać pętlę. Brakuje ostatniego regionu, w którym opisujemy, jak przejść *od punktu początkowego do punktu kończącego*:

```
for (var i = 0; i < count; i++) {  
    saySomething();  
}
```

W naszym przykładzie przy każdej iteracji pętli zwiększamy wartość zmiennej `i` o 1. Na wypadek, gdybyś nie był zaznajomiony z tą składnią, wyjaśniam, że znaki `++` po `i` oznaczają, iż zwiększamy o 1 bieżącą wartość zmiennej `i` bez względu na to, jaka ona jest. Przy każdym przejściu pętli wykonywane będzie to, co zostało w niej określone. W tym przypadku zwiększana będzie nasza wartość `i`.

Zbierając wszystko razem

Skoro przyjrzeliliśmy się już każdej części instrukcji `for` w najdrobniejszych szczegółach, przeanalizujemy całość jeszcze raz, aby upewnić się, że wszystko zrozumieliśmy. Nasz kompletny przykład wygląda następująco:

```
var count = 10;  
  
for (var i = 0; i < count; i++) {  
    saySomething();  
}  
  
function saySomething() {  
    document.writeln("Witaj!");  
}
```

Gdy kod po raz pierwszy dociera do pętli `for`, zmienna `i` jest tworzona i inicjowana do wartości 0. Następnie kod sprawdza, czy wartość `i` jest mniejsza od wartości wskazywanej przez zmienną `count`..., która wynosi 10. Na tym etapie wszystko się zgadza i wykonywany jest kod, który znajduje się wewnątrz pętli. W naszym przypadku jest to funkcja `saySomething`. Po wykonaniu poleceń wewnątrz pętli zaczyna działać ostatnia część instrukcji `for`. Zmienna `i` jest zwiększana o 1.

Teraz pętla zaczyna się od nowa, z tym wyjątkiem, że zmienna `i` nie jest ponownie inicjowana. Jej wartość jest ustawiana na wartość zwiększoną o 1, a to oznacza, że wciąż przechodzi test `i < count`. Funkcja `saySomething` jest wywoływana ponownie, a wartość `i` jest ponownie zwiększana. Teraz wartość zmiennej `i` wynosi 2.

Cały ten proces powtarza się, aż wartość `i` będzie równa 10. Na tym etapie test `i < count` nie powiedzie się, a pętla zostanie przerwana po pomyślnym wykonaniu funkcji `saySomething` 10 razy.

Kilka przykładów pętli for

W poprzednim podrozdziale zrobiliśmy sekcję prostej pętli `for` i opisaliśmy wszystkie jej wewnętrzne mechanizmy działania. W przypadku pętli `for`, a przede wszystkim w całym języku JavaScript, chodzi o to, że prosty przykład nie zawsze obejmuje wszystko, czego możesz potrzebować. Najlepszym rozwiązaniem jest przeanalizowanie kilku kolejnych przykładów pętli `for` i to właśnie będziemy robić w kilku kolejnych punktach tego podrozdziału.

Wcześniejsze przerywanie pętli

Czasami możemy chcieć zakończyć pętlę, zanim dotrze do końca. Do zakończenia pętli używamy słowa kluczowego `break`. Poniżej znajduje się przykład:

```
for (var i = 0; i < 100; i++) {
    document.writeln(i);

    if (i == 45) {
        break;
    }
}
```

Gdy wartość `i` będzie równa 45, słowo kluczowe `break` powstrzyma dalsze wykonywanie pętli. Chociaż ten przykład jest nieco naciągany, to gdy w prawdziwym kodzie napotkasz przypadek wymagający zakończenia pętli, będziesz wiedział, co robić.

Pomijanie iteracji

Niekiedy może być konieczne, aby pętla pominęła swoją bieżącą iterację i przeszła do następnej. Jest to sprytnie obsługiwane przez słowo kluczowe `continue`:

```
var floors = 28;

for (var i = 1; i <= floors; i++) {
    if (i == 13) {
        // tu nie ma piętra
        continue;
    }

    document.writeln("Piętro: " + i + "<br>");
}
```


W przeciwieństwie do słowa kluczowego `break`, które powoduje po prostu zatrzymanie wykonywania pętli, słowo kluczowe `continue` wskazuje, aby zatrzymać bieżącą iterację i przejść do następnej iteracji. Przekonasz się, że będziesz często używać słowa kluczowego `continue` podczas obsługi błędów, gdzie będziesz chciał, aby pętla po prostu przeszła do kolejnego elementu.

Odliczanie wstecz

Nie ma żadnego powodu, który zmuszałby nas do zaczynania odliczania od 0 i następnie zwiększania wartości:

```
for (var i = 25; i > 0; i--) {  
    document.writeln("Witaj");  
}
```

Można równie dobrze zacząć od wysokiej wartości, a następnie zmniejszać ją aż do momentu, kiedy warunek pętli zwróci `false`.

Mogłeś słyszeć opinie, że takie postępowanie zwiększa wydajność pętli. Za wcześniej jeszcze na ocenę, czy dekrementacja jest rzeczywiście szybsza niż inkrementacja, ale zachęcam do eksperymentowania i sprawdzania, czy można zaobserwować jakiegokolwiek korzyści związane z wydajnością.

Nie trzeba używać liczb

Podczas wypełniania pętli `for` nie trzeba używać wyłącznie liczb ani wykonywać tradycyjnej operacji inkrementacji (dekrementacji):

```
for (var i = "a" ; i != "aaaaaaa" ; i += "a" ) {  
    document.writeln("hmm...");  
}
```

Można używać, czego tylko się chce, ale pod warunkiem, że pętla ostatecznie osiągnie punkt, w którym będzie mogła się zakończyć. Zwróć uwagę, że w tym przykładzie używam do uruchomienia pętli litery `a` jako mojej waluty. Gdy „dodajemy” do siebie litery (tak jak w tym przykładzie), rezultat jest następujący: `a`, `aa`, `aaa`, `aaaa`, `aaaaa` itd.

Tablica! Tablica! Tablica!

Wiem, że nie przyglądaliśmy się jeszcze szczegółowo tablicom, ale jedna z największych historii miłosnych wszech czasów rozgrywa się pomiędzy strukturą danych zwaną tablicą a pętlą `for`. W większości przypadków będziesz używać pętli do przechodzenia przez wszystkie dane przechowywane w tablicy, rzućmy więc okiem na poniższy kod i objaśnienie. Odświeżymy te informacje w sposób bardziej szczegółowy, kiedy oficjalnie będziemy omawiać tablice w rozdziale 15.

```
var myArray = ["jeden" , "dwa" , "trzy" ];
for (var i = 0; i < myArray.length; i++) {
    document.writeln(myArray[i]);
}
```

Mówiąc w dużym skrócie i bez przymuszania, tablica jest kolekcją elementów. Wylizanie i uzyskiwanie dostępu do wszystkich elementów w tablicy wymagają pewnego rodzaju pętli, a zazwyczaj wybierana jest w tym celu pętla for.

W każdym razie pomyślałem, że zapoznam Cię z tą parą: tablicą i pętlą for. Podobnie jak jest w przypadku dwóch pozornie losowych postaci pojawiających się na początku filmu Quentina Tarantino, w tym momencie ich rzeczywisty wkład do historii wydaje się nieistotny.

O nie, nie zrobił tego!

O tak! Zrobiłem. Poszedłem tam, zrobiłem zdjęcie, zamieściłem je na Facebooku i wróciłem:

```
var i = 0;
var yay = true;

for (; yay;) {
    if (i == 10) {
        yay = false;
    }
    i++;
    document.writeln("dziwne");
}
```

Nie trzeba wypełniać wszystkich trzech sekcji pętli for, aby ta pętla działała. Dopóki jesteś w stanie ostatecznie spełnić warunek kończący pętlę, możesz robić, co chcesz — tak jak zrobiłem w powyższym przykładzie. Pamiętaj jednak, że chociaż możesz coś zrobić, to nie znaczy jeszcze, że powinieneś. Ten przykład wyraźnie podpada pod kategorię „nie powinieneś tego robić!”.

Pozostałe pętle

Warianty pętli while i do...while żyją w cieniu ukochanej pętli for. Te dwa warianty pętli wyraźnie służą jakiemuś celowi, ale nigdy nie udało mi się go odkryć. Mimo to w trosce o kompletność i w celu zapoznania Cię z kodem, który można spotkać w naturalnym środowisku, przyjrzyjmy się pokrótce obu z nich.

Pętla while

Pętla while powtarza pewien kod, dopóki jego warunek testowy (kolejne wyrażenie) nie zwróci false:

```
var count = 0;

while (count < 10) {
    document.writeln("zapętlamy!");

    count++;
}
```

W tym przykładzie ten warunek jest reprezentowany przez wyrażenie `count < 10`. Przy każdej iteracji nasza pętla zwiększa wartość `count` o 1. Gdy wartość `count` będzie równa 10, pętla zatrzyma się, ponieważ wyrażenie `count < 10` zwróci `false`. To wszystko na temat pętli `while`.

Pętla `do...while`

Przejdźmy teraz do Meg Griffin wariantów pętli. Będzie to pętla `do...while`, której cel jest jeszcze mniej określony niż cel pętli `while`. Podczas gdy w pętli `while` wyrażenie warunkowe znajduje się na początku przed wykonywanym w niej kodem, pętla `do...while` ma swoje wyrażenie warunkowe na końcu.

Oto przykład:

```
var count = 0;

do {
    count++;

    document.writeln("Nie wiem, co ja tu robię!");
} while (count < 10);
```

Główną różnicą między pętlami `while` oraz `do...while` jest to, że zawartość pętli `while` nigdy nie zostanie wykonana, gdy jej wyrażenie warunkowe od samego początku będzie zwracać `false`:

```
while (false) {
    document.writeln("Tego nie tkniesz!");
}
```

Zawartość pętli `do...while` zawsze zostanie uruchomiona co najmniej raz, ponieważ wyrażenie warunkowe jest ewaluowane dopiero po jednej iteracji:

```
do {
    document.writeln("Ten kod zostanie raz uruchomiony!");
} while (false);
```

Aaa... (ziewnięcie). W każdym razie, zanim przejdziemy dalej, muszę Ci przekazać jeszcze jedną informację. Instrukcje `break` i `continue`, które widzieliśmy wcześniej jako część niesamowitej pętli `for`, działają również podobnie, gdy są stosowane wewnątrz wariantów pętli `while` i `do...while`.

ABSOLUTNE MINIMUM

Proszę bardzo — oto opis pętli `for` i sposobów jej używania oraz bardzo podstawowe omówienie pętli `while` i `do...while`. Na razie prawdopodobnie nie będziesz jeszcze zbyt często używać pętli. Gdy przejdziemy do bardziej skomplikowanych sytuacji obejmujących m.in. kolekcje danych, elementy HTML w dokumencie, manipulacje tekstem, pętle staną się jednym z naturalnych elementów, na których często będziesz polegać.



Skorowidz

A

abstrakcja, 109
analiza kodu, 22
argumenty
 funkcji, 35
 zdarzenia, 266
attribut, 214
 async, 308
 class, 220
 defer, 308, 309

B

badanie
 obiektów, 255
 struktury DOM, 246
bąbelkowanie, 263, 269
blok, 72

C

cel zdarzenia, 271
CSS, 196, 218, 219

D

debugowanie kodu JavaScript, 249
deklarowanie zmiennych, 28
dekrementacja, 139
dobre praktyki komentowania, 101
dodawanie wartości klas, 220
dokument HTML, 20, 90, 95
DOM, Document Object Model,
 194, 197, 223
 badanie struktury, 246
 klonowanie elementów, 238
 lokalizacja skryptów, 306
 przechodzenie przez drzewo, 223
 tworzenie elementów, 232
 usuwanie elementów, 237
domknięcia, 75, 77, 184
dostęp
 do elementów potomnych, 228
 do wartości tablicy, 129
 do znaków, 114

działania matematyczne, 139
dziedziczenie, 151
dzielenie łańcucha znaków, 117

E

edycja kodu, 20
element script, 308
elementy
 DOM, 210, 225
 potomne, 228
 tablicy, 128

F

faza
 bąbelkowania zdarzeń, 274
 przechwytywania zdarzeń, 273
funkcja, 22, 33, 179
 addEventListener, 262
 Boolean, 170
 querySelector, 204
 querySelectorAll, 204, 205
 requestAnimationFrame, 66
 setInterval, 65
 setTimeout, 64
funkcje
 argumenty, 35
 trygonometryczne, 144
 wewnątrz funkcji, 78
 wewnętrzne, 81
 zewnętrzne, 81
 zwracające dane, 37

G

globalna pozycja myszy, 288

H

HTML, 194

I

IIFE, 181, 192
indeks, 114

inicjowanie zmiennych, 28
inkrementacja, 139
instrukcja warunkowa
 if, 46
 if-else, 42
 if-else/if-else, 47
 switch, 48
interfejs API classList, 221
iteracja, 58

J

JavaScript, 19, 196

K

klawisze strzałek, 297
kliknięcie, 282
klonowanie elementów DOM, 238
koercja typów, 172
kolizje, 183
komentarze, 98
 jednoliniowe, 99
 w stylu JSDOC, 100
 wieloliniowe, 99
konkatenacja, 115
konsola, 254

L

liczby, 138
 losowe, 145
literał, 122
 obiektu, 149
lokalizacja w dokumencie, 93

Ł

ładowanie strony, 302
łańcuch
 prototypów, 151
 znaków, 111, 122
łączenie łańcuchów znaków, 115

M

menu kontekstowe, 287
 metoda
 lastIndexOf, 119
 slice, 116
 split, 117
 substr, 117
 toLowerCase, 119
 toUpperCase, 119
 metody obiektu String, 114
 model
 DOM, 194
 dziedziczenia prototypowego, 160
 modyfikowanie elementów DOM, 212
 mysza, 281

N

nadawanie stylów, 218
 NaN, 141
 narzędzia
 dla programistów, 243, 244, 257
 do edycji kodu, 20
 Web Inspector, 257
 nasłuchiwanie zdarzeń, 262, 311, 317
 nasłuchiwanie zdarzeń klawiatury, 295
 nazwa
 argumentu, 37
 zdarzenia, 262
 zmiennej, 27
 nieskończoność, 141
 notacja literału, 152
 Null, 176

O

obiekt, 107, 147
 Array, 164
 Boolean, 170
 document, 200
 Math, 142
 MouseEvent, 288
 String, 114
 metody, 114
 właściwości, 114
 window, 199

objektowy model dokumentu, 194, 197

obiekt
 nadrzędne, 154
 niestandardowe, 154
 podrzędne, 154
 predefiniowane, 109
 wbudowane, 161
 obsługa
 kółka myszy, 291
 zdarzeń dla wielu elementów, 311
 odwołanie cykliczne, 67
 określanie właściwości, 152
 operator, 138
 identyczności, 172
 nieidentyczności, 172
 operatory warunkowe, 44

P

para klucz-wartość, 148
 parsowanie, 93
 pętla
 do...while, 61
 for, 54
 pomijanie iteracji, 58
 warunek kończący, 56
 warunek rozpoczynający, 56
 wcześniejsze przerywanie, 58
 while, 60
 pierwiastki kwadratowe, 145
 pisanie IIFE, 181, 182
 plik JavaScript, 91
 płynne animowanie, 66
 podłańcuchy, 116
 polecenie, 22
 return, 38
 potęgi, 145
 pozycja myszy, 289
 procedura obsługi zdarzeń, 263
 prototyp, 162
 prymitywy, 107, 122
 prywatne środowisko, 191
 przechodzenie przez drzewo DOM, 223, 229
 przechwytywanie zdarzeń, 263, 269
 przełączanie wartości klas, 221
 przerywanie zdarzenia, 276
 przeszukiwanie łańcucha znaków, 118
 punkt przerywania, 250

R

reagowanie na zdarzenia, 264
 rejestrowanie wiadomości, 256
 rodzeństwo, 226
 rodzic, 226
 rozszerzanie
 obiektów wbudowanych, 161, 166

S

scalanie tablic, 134
 selektor atrybutu, 206
 składnia selektora CSS, 206
 skrypt, 93, 306
 słowo kluczowe
 continue, 58
 return, 38
 this, 158
 var, 72
 stałe, 142
 stan zablokowania, 184
 struktura, 194
 drzewiasta, 203
 style, 218
 stylizacja, 196
 suwak zawartości, 183

T

tablica, 59, 127
 dodawanie elementów, 130
 scalanie, 134
 szukanie elementów, 134
 tworzenie, 128
 usuwanie elementów, 132
 uzyskiwanie dostępu, 129
 timer, 64
 tryb debugowania, 251
 tworzenie
 elementów DOM, 232
 funkcji, 35
 niestandardowych obiektów, 154
 obiektów, 149
 podłańcuchów, 116
 tablicy, 128
 typ
 Event, 266
 Object, 122, 147, 148

typy

- objektowe, 107
- podstawowe, 107
- prymitywne, 107

U

umieszczanie kodu

- w dokumencie HTML, 90
- w pliku, 91

Undefined, 177

unikanie kolizji, 183

ustawianie stylu bezpośrednio, 219

usuwanie

- elementów DOM, 237
- nasłuchiacza zdarzeń, 267
- wartości klas, 220

używanie

- IIFE, 183
- komentarzy, 101
- liczb, 138
- zmiennych, 26

W

walidator nazw zmiennych, 28

wartości

- atributów, 214
- logiczne, 169
- specjalne, 141
- tablicy, 128

wartość, 26

- bezwzględna, 145
- Infinity, 141
- NaN, 141
- Null, 176
- Undefined, 177

wcześniejsze przerywanie pętli, 58

węzły, 198

widok

- struktury DOM, 247
- źródła strony, 249

właściwości, 148, 152

- obiektu MouseEvent, 288
- obiektu String, 114
- zdarzenia klawiatury, 296

właściwość

- charCode, 296
- keyCode, 297

wychodzenie z funkcji, 38

wykrywanie kliknięć, 289

wynoszenie zmiennych, 74

wyrażenia funkcyjne, 180

wzorzec Modułu Ujawniającego, 190

Z

zakres

- bloku, 72
- globalny zmiennych, 70
- lokalny zmiennych, 71

zaokrąglenie liczb, 143

zdarzenia, 260

- klawiatury, 294, 296
- ładowania strony, 302
- mysz, 282

zdarzenie

- click, 279
- DOMContentLoaded, 304
- DOMMouseScroll, 291
- keypress, 294
- load, 304

mousemove, 286

mouseout, 284

mouseover, 284

mouseup, 285

zmiana wartości tekstowej

elementu, 213

zmiennie, 26

deklarowanie, 28

inicjowanie, 28

nazwy, 27

zakres globalny, 70

zakres lokalny, 71

znacznik script, 88

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

JavaScript:

dzięki niemu strony WWW stają się piękniejsze,
lepsze i bardziej imponujące!



Naukę języka programowania można porównać do nauki języka obcego: jedno i drugie wymaga sporo wysiłku i systematyczności. Trzeba najpierw przyswoić sobie podstawy, potem przejść do bardziej złożonych zagadnień i jak najwięcej ćwiczyć. JavaScript szczególnie dobrze nadaje się do nauki programowania, a przy tym jest językiem elastycznym i wciąż rozwijającym. W połączeniu z CSS3 i HTML5 pozwala na tworzenie wszechstronnych aplikacji internetowych i mobilnych.

Książka, którą trzymasz w dłoni, jest przystępnie napisanym, praktycznym podręcznikiem programowania w JavaScriptcie. Zawarte tu wyjaśnienia, wskazówki i proste instrukcje pozwolą Ci na bardzo szybkie przyswojenie sobie podstaw tego języka, nawet jeśli dopiero zaczynasz przygodę z kodowaniem. Po przedstawieniu podstaw programowania wyjaśniono nieco bardziej złożone koncepcje, posługując się licznymi – często bardzo zabawnymi – przykładami. Lektura książki jest przy tym niezwykle interesująca i przyjemna. Po zakończeniu ostatniego rozdziału będziesz przygotowany, by sprostać większości wyzwań związanych z językiem JavaScript.

- Jak zacząć pisać kod i gdzie go umieścić
- Wykorzystanie zmiennych, funkcji i pętli
- Co to jest zakres globalny i lokalny
- Czym są domknięcia
- Programowanie obiektowe w JavaScriptcie
- Operacje na danych tekstowych i listy danych



Kirupa Chinnathambi jest menedżerem programowania w Microsoftzie. Od dzieciństwa pisze kod i animacje komputerowe, jednak jego prawdziwą pasją jest tworzenie stron WWW. Od kilkunastu lat uczy innych tej sztuki: publikuje poradniki na swojej stronie *kirupa.com*, pisze artykuły i książki, nagrał również sporo filmików, które można znaleźć na stronach serwisu YouTube. W wolnych chwilach grywa w gry wideo, chodzi na koncerty, ogląda filmy i spędza czas z przyjaciółmi.

Helion

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Informatyka w najlepszym wydaniu

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

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

ISBN 978-83-283-3180-8



cena: 59,00 zł

sięgnij po WIĘCEJ



KOD KORZYŚCI