3

Język JavaScript

3.1. Wprowadzenie

Techniki służące do dynamicznej zmiany treści, wyglądu lub zachowania się dokumentu HTML umożliwiają interakcję użytkownika ze stroną internetową. W skład dynamicznego HTML wchodzą technologie: DOM, CSS, HTML, SVG, języki skryptowe (JavaScript, VBScript i inne).

DOM (ang. *Document Object Model*) to sposób reprezentacji dokumentów XML i HTML w postaci modelu obiektowego. Jest on niezależny od platformy i języka programowania. Standard zdefiniowany dla DOM przez organizację W3C opiera się na zespole klas i interfejsów. Za ich pomocą możliwy jest dostęp do struktury dokumentu oraz możliwa jest jego modyfikacja. Standard W3C definiuje interfejsy DOM tylko dla języków JavaScript i Java.

SVG (ang. *Scalable Vector Graphics*) to uniwersalny format grafiki wektorowej (statycznej i dynamicznej) stworzony z myślą o zastosowaniu na stronach internetowych. Może być integrowany z językami opartymi na technologii XML. Został opracowany przez organizację W3C na potrzeby internetu. W SVG oprócz standardowych obiektów (prostokątów, elips, krzywych) można opisywać efekty specjalne, maski przezroczystości oraz sposób animacji elementów.

Języki skryptowe to języki interpretowane, zaprojektowane z myślą o interakcji z użytkownikiem. Skrypty są wykonywane wewnątrz określonej aplikacji w odróżnieniu od programów nieskryptowych, które wykonują się niezależnie od innych aplikacji. Język skryptowy działający po stronie przeglądarki jest najważniejszym elementem dynamicznego HTML.

3.2. Struktura języka JavaScript

3.2.1. Wprowadzenie

Utworzenie kodu kontrolującego zachowanie strony oraz określającego interakcje z użytkownikiem jest kolejnym, po napisaniu kodu HTML i zdefiniowaniu arkuszy CSS,

etapem budowy strony internetowej. Do tego celu stosuje się języki skryptowe. Najpopularniejszym z nich jest JavaScript. Odgrywa on bardzo dużą rolę w projektowaniu interaktywnych stron internetowych — umożliwia nie tylko prostą manipulację danymi, ale i dynamiczne modelowanie struktury strony, obsługę rozszerzeń multimedialnych oraz tworzenie odrębnych aplikacji.

Język JavaScript pozwala na tworzenie i umieszczanie w kodzie HTML krótkich programów, które mogą wykonywać różne zadania, na przykład obsługiwanie zdarzeń, weryfikowanie danych wprowadzanych do formularza, nawigowanie między stronami.

Przy użyciu odpowiednich bibliotek języka JavaScript (na przykład jQuery) można tworzyć zaawansowane aplikacje sieciowe o atrakcyjnych interfejsach.

3.2.2. Opis języka

JavaScript jest obiektowym skryptowym językiem programowania nawiązującym do języka C. Jest językiem interpretowanym, co oznacza, że efekty jego użycia można zobaczyć bez konieczności kompilowania kodu. Potrzebna jest tylko przeglądarka internetowa, która obsługuje język JavaScript. Tworzone skrypty zapewniają interaktywność strony internetowej poprzez reagowanie na zdarzenia, budowanie elementów nawigacji oraz sprawdzanie poprawności formularzy.

Typowe zastosowania języka JavaScript to:

- modyfikowanie wyglądu bieżącego dokumentu,
- wyświetlanie prostych okien dialogowych,
- kontrola poprawności wypełnienia formularza,
- manipulowanie informacją dotyczącą daty i czasu,
- lokalne generowanie dokumentów HTML.

3.2.3. JavaScript w HTML

Kod źródłowy napisany w języku JavaScript może być umieszczony wewnątrz dokumentu HTML między znacznikami `<script>` i `</script>`.

Znaczniki skryptu mogą być wstawiane w dowolnym miejscu dokumentu, ale zalecane jest umieszczanie ich na początku strony, w sekcji `<head>`. W obrębie strony można używać znaczników `<script>` wielokrotnie, na przykład w sekcjach `<head>` i `<body>`.

Przykład 3.1

Skrypt umieszczony w dokumencie HTML:

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>JavaScript - Przykładowy program</title>
</head>
<body>
<script>
document.write("Napisz program w języku JavaScript");
</script>
</body>
</html>
```

Dobrą praktyką programistyczną jest wielokrotne wykorzystywanie raz napisanego kodu. Dlatego kod źródłowy skryptu, który będzie wielokrotnie używany, można umieścić w osobnym pliku. Będzie to plik tekstowy, który powinien mieć rozszerzenie *js*. Kod zapisany w takim pliku nie zawiera znaczników `<script>`. Jeżeli skrypt został zapisany w zewnętrznym pliku, to w kodzie HTML powinna pojawić się instrukcja dołączana w postaci znacznika `<script>` i atrybutu `src`. Znacznik ten może zostać umieszczony w sekcji `<head>` lub w sekcji `<body>`. To rozwiązanie, podobnie jak w przypadku zewnętrznych arkuszy CSS, ułatwia wprowadzanie zmian w kodzie strony bez ingerencji w pliki HTML.

Przykład 3.2

Skrypt umieszczony w pliku zewnętrznym o nazwie *skrypt.js*:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Przykładowy program</title>
<script src="skrypt.js">
</script>
</head>
<body>
<p>Skrypt znajduje się w pliku "skrypt.js".</p>
</body>
</html>
```

Zawartość pliku *skrypt.js*:

```
document.write("Napisz program w języku JavaScript");
```

3.2.4. Instrukcja document.write

Instrukcja `document.write()` zapisuje ciąg tekstowy w dokumencie HTML i najczęściej jest używana do testowania.

`document` jest to obiekt JavaScript, który reprezentuje aktualnie wyświetlaną stronę, natomiast `write()` to jego metoda, czyli funkcja wykonująca określone działania na obiekcie — w tym wypadku wypisuje ona tekst.

Tekst umieszczamy w nawiasach, to argument wywołania metody.

Ogólny zapis: `obiekt.metoda(argumenty metody);`.

Znaczniki HTML w document.write

Wewnątrz instrukcji `document.write()` można używać znaczników HTML. Należy je tylko w odpowiednich miejscach otwierać i zamykać.

Przykład 3.3

```
document.write("<h1>JavaScript</h1>");
```

Przykład 3.4

```
document.write("<a href='spis.html'>Spis treści</a>");
```

Metoda `write()` w podanych przykładach przyjmuje jako argument łańcuch znakowy — dlatego jest on umieszczony w cudzysłowie. Można jako argument podawać również liczbę całkowitą lub zmiennoprzecinkową.

Przykład 3.5

```
document.write(201);
```

Argumentem metody `write()` może być ciąg powstały z połączenia tekstu i wartości zmiennych.

Przykład 3.6

```
var i = 30;
document.write("Zmienna i ma wartość "+ i +"<br>");
```

3.3. Składnia języka JavaScript

Składnia języka jest ściśle określonym zbiorem reguł, których należy przestrzegać. W języku JavaScript instrukcje zawierają wyrażenia, słowa kluczowe i komentarze. W wyrażeniach występują zmienne, literały i operatory. Do elementów języka należy również zbiór predefiniowanych obiektów i funkcji (na przykład tablica, data, funkcje matematyczne).

Przykład 3.7

```
var a, b, c;        // deklaracja zmiennych
a = 4; b = 9;       // przypisanie wartości
c = a + b;          // wyrażenie
```

3.3.1. Instrukcje

Instrukcje w języku JavaScript mogą być pisane pojedynczo, w jednym wierszu lub w kilku wierszach. Średnik odgrywa rolę ogranicznika instrukcji, ale nie jest wymagany, jeżeli po instrukcji wystąpi znak nowego wiersza. Jeśli kilka instrukcji zostanie zapisanych w jednym wierszu, muszą być one oddzielone średnikiem.

Język JavaScript ignoruje powtarzające się spacje. Spacje można umieszczać w dowolnych miejscach, aby zwiększyć czytelność skryptu. Wskazane jest wstawianie spacji wokół operatorów =, +, –, *, /, na przykład:

```
var c = a + b;
```

Instrukcje mogą być grupowane w bloki za pomocą nawiasów klamrowych `{}`. Instrukcje zgrupowane w bloki są wykonywane razem jedna po drugiej.

3.3.2. Wielkość liter

W języku JavaScript rozróżniana jest wielkość liter oraz wspierany jest standard znaków Unicode. W kodzie skryptu przy zapisywaniu nazw stosuje się następujące zasady:

- Słowa kluczowe pisane są małymi literami, na przykład `for`.
- Nazwy obiektów wbudowanych pisane są od dużej litery, a pozostałe litery są małe, na przykład `Date`.
- Nazwy obiektów DOM zapisywane są małymi literami, ale w nazwach metod tych obiektów dopuszczane są małe i duże litery, na przykład `toLowerCase`.
- Białe znaki, typu spacja, znak tabulacji itp., są nieistotne.

3.3.3. Słowa kluczowe

Słowa kluczowe określają czynności, które powinny zostać wykonane przez instrukcję zawierającą te słowa. Słowa kluczowe są słowami zastrzeżonymi i nie mogą być nazwami zmiennych.

3.3.4. Komentarze

Komentarze są ignorowane w trakcie przetwarzania kodu. Mogą być umieszczane w dowolnym miejscu kodu. Komentarzem jest każdy wiersz rozpoczynający się od znaków `//`.

Przykład 3.8

```
// tutaj znajduje się komentarz
```

Komentarz rozpoczynający się od `//` można także umieszczać po instrukcji.

```
a = b + c // suma wartości
```

Jeżeli komentarz zawiera wiele wierszy, można użyć ograniczników `/*` (do rozpoczęcia komentarza) i `*/` (do zakończenia komentarza).

Tak zapisane komentarze dopuszczalne są tylko wewnątrz znaczników `<script>` i `</script>`.

Najczęściej stosowane są komentarze jednowierszowe. Natomiast komentarze blokowe są wykorzystywane w dokumentacjach.

3.3.5. Zmienne

Zmienne służą do przechowywania danych i wyników w celu dalszego ich wykorzystywania. W języku JavaScript zmienne deklaruje się za pomocą słowa kluczowego `var` poprzedzającego nazwę zmiennej. Można również stosować nowy sposób deklaracji zmiennej za pomocą słowa kluczowego `let`. Nazwa zmiennej może zawierać litery, cyfry oraz znak podkreślenia, natomiast nie może zaczynać się od cyfry. Wielkość liter używanych w nazwach ma znaczenie. Tworzone zmienne nie mają określonego typu. Typ jest przypisywany do zmiennej po nadaniu jej wartości. Typ danych nie jest przypisany do zmiennej na stałe i może ulegać zmianie. Dobrą praktyką jest deklarowanie wszystkich zmiennych na początku skryptu.

Po deklaracji zmienna nie ma wartości. Do przypisania jej wartości służy operator =.

Przykład 3.9

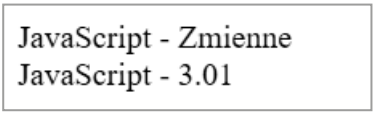
```
var x, y, Nazwa;
var miasto = "Warszawa";
var Miasto = 5;
let tekst = "Dowolny tekst";
```

Przykład 3.10

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Deklaracja zmiennych</title>
<script>
var x = "JavaScript - ";
var x = "JavaScript - ";
```

```
var y = "Zmienne";
document.write(x + y + "<br>");
var y = 3.01;
document.write(x + y);
</script>
</head>
<body>
</body>
</html>
```

W podanym przykładzie zmiennym `x` i `y` zostały przypisane ciągi znaków i połączony ciąg znaków został wyświetlony na ekranie. Następnie zmiennej `y` przypisano wartość liczbową i ponownie połączony ciąg znaków został wyświetlony na ekranie. Zmienna `y` zmienia swój typ w zależności od typu przypisanej wartości (rysunek 3.1).



Rysunek 3.1. Rezultat zmiany typu zmiennej y

Jeżeli zadeklarowanej zmiennej nie zostanie przypisana wartość, to będzie ona miała wartość `undefined`.

Ćwiczenie 3.1

Utwórz zmienne: `nazwisko`, `imie`, `wiek`, `klasa`. Przypisz tym zmiennym przykładowe dane. Wyświetl w przeglądarce internetowej informację o treści:

Uczeń naszej szkoły nazywa się *(tu powinno pojawić się: imię nazwisko)*.

Uczęszcza do klasy *(tu powinna pojawić się klasa)*.

Ma *(tu powinien pojawić się wiek)* lat.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Deklaracja zmiennych</title>
</head>
<body>
<script>
```

```
var nazwisko = "Witkowski";
var imie = "Maciej";
var wiek = "15";
var klasa = "2a";
document.write("Uczeń naszej szkoły nazywa się " + nazwisko +
" " + imie + ".<br>");
document.write("Uczęszcza do klasy " + klasa + ".<br>");
document.write("Ma " + wiek + " lat." + "<br>");
</script>
</body>
</html>
```

3.3.6. Literały

Stałe wartości w języku JavaScript są nazywane **literałami**. Literałem może być liczba lub ciąg tekstowy.

Liczby są zapisywane w postaci liczb całkowitych lub w postaci dziesiętnej. Do oddzielenia części dziesiętnej służy znak kropki, na przykład `157.23`.

Ciągi znakowe zapisywane są w podwójnym lub pojedynczym cudzysłowie, na przykład `"Programowanie w JavaScript"`.

3.3.7. Identyfikatory

Identyfikatory to nazwy zmiennych, funkcji, etykiet oraz słów kluczowych. Definiowanym zmiennym, obiektom, funkcjom można nadawać nazwy pisane w dowolny sposób.

Nazwy powinny być tworzone według następujących zasad:

- Oprócz liter można w nich stosować cyfry, znak podkreślenia i znak dolara.
- Nazwa może zaczynać się literą, znakiem podkreślenia lub znakiem dolara.
- Pierwszym znakiem nazwy nie może być liczba.
- W nazwach jest rozróżniana wielkość liter.
- Nazwą nie może być słowo zarezerwowane.

Do łączenia słów w nazwie nie można używać jako łącznika znaku -. Łącznikiem może być znak podkreślenia, na przykład `Data_urodzenia`, lub zbiór małych i dużych liter, na przykład `KodPocztowy`, `kolorOczu`. W języku JavaScript przyjęte jest zapisywanie nazwy od małej litery, na przykład `dobraKawa`.

3.3.8. Skalarne typy danych

Język JavaScript jest językiem słabo typowanym, co oznacza, że w momencie deklarowania zmiennej nie trzeba określać jej typu. Dopuszczane są następujące typy danych:

- typ liczbowy `number`,
- typ łańcuchowy `string`,
- typ logiczny `boolean`,
- typ `null`,
- typ `undefined`.

Typ liczbowy number

Służy do zapisywania liczb. Można zapisywać liczby w formatach wykładniczym, dziesiętnym, ósemkowym i szesnastkowym. Jeżeli liczba zostanie poprzedzona cyfrą zero (prefiks `0`), to jest traktowana jako wartość ósemkowa (na przykład `042`). Jeżeli zostanie poprzedzona ciągiem znaków `0x` lub `0X`, to jest traktowana jako wartość szesnastkowa, inaczej heksadecymalna (na przykład `0x23A`). Wartości liczbowe mogą być zapisywane w notacji wykładniczej (na przykład `3e-2`). Jeżeli liczba nie jest poprzedzona żadnym znakiem lub jest poprzedzona znakiem `+`, jest to wartość dodatnia. Jeżeli jest poprzedzona znakiem `–`, jest to wartość ujemna.

Przykład 3.11

```
var a = 12;
var b = 037;
var c = 0xACB;
var d = 0.12e-2;
```

Typ łańcuchowy string

Zawiera ciągi znaków o dowolnej długości. Ciąg znaków musi być umieszczony w ogranicznikach typu cudzysłów lub apostrof, na przykład `"Anna"`, `'Nowak'`.

Przykład 3.12

```
var a = "Warszawa";
var b = 'Kraków';
```

Typ logiczny boolean

Przyjmuje jedną z dwóch wartości: prawda (`true`) lub fałsz (`false`). Wartości typu logicznego są wykorzystywane przy budowaniu wyrażeń logicznych, porównywaniu danych, określaniu, czy wykonywana operacja zakończyła się sukcesem.

Przykład 3.13

```
var k = true;
```

Typ null

Jest to typ specjalny określający wartość pustą (`null`). Nie przechowuje żadnej wartości. W języku JavaScript ten typ danych jest obiektem.

Typ undefined

Jest to typ zawierający zmienne, którym nie została nadana żadna wartość. Wartość oraz typ zmiennej są nieznane, na przykład `var x;`.

Zmiennej `x` nie przypisano wartości. Ma typ i wartość `undefined`.

3.3.9. Złożone typy danych

W języku JavaScript zostały zdefiniowane również złożone typy danych. Oznacza to, że wewnątrz nich można przechowywać więcej niż jedną wartość. Takie zmienne są obiektami i są typu referencyjnego. Zmienne obiektowe nie mają przypisanej bezpośrednio wartości, tylko wskazują adres w pamięci, gdzie dane są przechowywane.

Obiekty

Służą do reprezentacji obiektów. Wykorzystywane są obiekty wbudowane oraz udostępniane przez przeglądarkę. Właściwości obiektów są zapisywane w nawiasach klamrowych jako para *nazwa: wartość* i są oddzielane przecinkami.

Przykład 3.14

```
var osoba = {nazwisko: "Nowak", imie: "Paweł", wiek: 18};
```

Tablice

Służą do przechowywania wielu wartości. Tablice są zapisywane w nawiasach kwadratowych, a elementy tablicy są oddzielane przecinkami.

Przykład 3.15

```
var liczby = [10, 23, 57, 94, 32];
```

3.3.10. Operatory

Operatory w języku JavaScript zostały podzielone na następujące grupy:

- arytmetyczne,
- porównania,
- bitowe,
- logiczne,
- przypisania,
- pozostałe.

Operatory arytmetyczne

Służą do wykonywania operacji arytmetycznych. W tej grupie znajdują się też operatory **inkrementacji** (zwiększania) i **dekrementacji** (zmniejszania). Operatory arytmetyczne są dwuargumentowe, natomiast operatory inkrementacji i dekrementacji są jednoargumentowe (tabela 3.1).

Tabela 3.1. Operatory arytmetyczne

Operator	Działanie	Przykład
+	dodawanie	a + b
-	odejmowanie	a - b
*	mnożenie	a * b
/	dzielenie	a / b
%	modulo (reszta z dzielenia)	a % b
++	inkrementacja	x++, ++x
--	dekrementacja	x--, --x

Przykład 3.16

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript - Operatory</title>
<script>
var a = 12;
var b = 5;
var c = a - b;
document.write("Wynikiem odejmowania jest ");
document.write(c);
document.write("<br>Wynikiem dodawania jest ");
document.write(a + b);
document.write("<br>Wynikiem mnożenia jest ");
document.write(a * b);
</script>
</head>
<body>
...
</body>
</html>
```

Operator inkrementacji powoduje zwiększenie wartości o jeden. Może występować w postaci przedrostkowej (`++x`) lub przyrostkowej (`x++`).

Operacja `x++` zwiększa wartość zmiennej po jej wykorzystaniu.

Operacja `++x` zwiększa wartość zmiennej przed jej wykorzystaniem.

Operatory `x++` i `++x` zwiększają wartość zmiennej, ale nie są równoważne.

Operator dekrementacji działa analogicznie, tylko zamiast zwiększać wartości zmiennych, zmniejsza je.

Przykład 3.17

```
<script>
for (var i = 0; i < 10; i++) {
  document.write("<br> Ile razy zostanie wykonana pętla? " + i);
}
</script>
```

Operatory porównania

Operatory porównania porównują argumenty. Wynikiem tej operacji jest wartość logiczna `true` (prawda) lub `false` (fałsz) (tabela 3.2).

Tabela 3.2. Operatory porównania

Operator	Działanie	Przykład
==	Wynik `true`, gdy argumenty są równe.	`a == b`
!=	Wynik `true`, gdy argumenty są różne.	`a != b`
===	Wynik `true`, gdy argumenty są tego samego typu i są równe.	`a === b`
!==	Wynik `true`, gdy argumenty są różne lub są różnych typów.	`a !== b`
>	Wynik `true`, gdy argument pierwszy jest większy od drugiego.	`a > b`
<	Wynik `true`, gdy argument pierwszy jest mniejszy od drugiego.	`a < b`
>=	Wynik `true`, gdy argument pierwszy jest większy od drugiego lub jest mu równy.	`a >= b`
<=	Wynik `true`, gdy argument pierwszy jest mniejszy od drugiego lub jest mu równy.	`a <= b`

Przykład 3.18

Sprawdzenie, czy liczby są równe:

```
<script>
var a = 10;
var b = '10';
if (a == b) {
  document.write("Liczby są równe.");
}
else {
  document.write("Liczby nie są równe.");
}
</script>
```

Przykład 3.19

Sprawdzenie, czy liczby są identyczne:

```
<script>
var a = 10;
var b = '10';
if (a === b) {
  document.write("Liczby są identyczne.");
}
else {
  document.write("Liczby nie są identyczne.");
}
</script>
```

Operatory bitowe

Operatory bitowe umożliwiają wykonywanie operacji na poszczególnych bitach liczb (tabela 3.3).

Tabela 3.3. Operatory bitowe

Operator	Działanie	Wyrażenie	Przykład (binarna reprezentacja)	Wynik (binarna reprezentacja)
`&`	iloczyn bitowy (AND)	`a & b`	`0101 & 0001`	`0001`
`|`	suma bitowa (OR)	`a | b`	`0101 | 0001`	`0101`
`~`	negacja bitowa (NOT)	`~ a`	`~ 0101`	`1010`

Operator	Działanie	Wyrażenie	Przykład (binarna reprezentacja)	Wynik (binarna reprezentacja)
^	bitowa różnica symetryczna	a ^ b	0101 ^ 0001	0100
>>	przesunięcie bitowe w prawo	a >> n	0101 >> 1	0010
<<	przesunięcie bitowe w lewo	a << n	0101 << 1	1010
>>>	przesunięcie bitowe w prawo z wypełnieniem zerami	a >>> n	0101 >>> 1	0010

Operatory logiczne

Przy użyciu operatorów logicznych wykonuje się operacje na argumentach, które mają przypisaną wartość logiczną — `true` lub `false` (tabela 3.4).

Tabela 3.4. Operatory logiczne

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

Wynikiem iloczynu logicznego jest wartość `true` tylko wtedy, gdy obydwa argumenty mają wartość `true`. W pozostałych przypadkach wynik ma wartość `false`.

Wynikiem sumy logicznej jest wartość `false` tylko wtedy, gdy obydwa argumenty mają wartość `false`. W pozostałych przypadkach wynik ma wartość `true`.

Negacja logiczna zmienia wartość argumentu na przeciwną.

Operatory przypisania

Za pomocą operatorów przypisania można przypisać wartości argumentom znajdującym się po lewej stronie operatora. Oprócz prostej operacji przypisania pozwalają na połączenie operacji przypisania z inną operacją, na przykład dodawania.

Zapis `i += 7` oznacza w praktyce to samo co zapis `i = i + 7`. Stosowanie skróconych zapisów upraszcza tworzenie bardziej rozbudowanych skryptów. W języku JavaScript istnieje duża grupa operatorów tego typu (tabela 3.5).

Tabela 3.5. Niektóre operatory przypisania

Operator	Przykład	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

3.4. Instrukcje sterujące

Instrukcje sterujące pozwalają zmienić kolejność wykonywania poleceń zapisanych w skrypcie w zależności od spełnienia lub niespełnienia określonych warunków. W języku JavaScript istnieje cała grupa instrukcji sterujących.

3.4.1. Instrukcja warunkowa

Instrukcja warunkowa pozwala na sprawdzenie w programie warunku i w zależności od tego, czy wynik jest prawdą, czy fałszem, dalsze wykonywanie instrukcji. Do takiego sprawdzania służy instrukcja `if ... else`.

```
if (warunek) {
 instrukcje
} else {
 instrukcje
}
```

Przykład 3.20

```
<!DOCTYPE html>
<html>
<head>
<title>Instrukcja warunkowa</title>
<script>
var a = -5;
var b = 3;
if (a > b) {
  document.write("Wartość zmiennej a jest większa od wartości
zmiennej b.");
} else {
  document.write("Wartość zmiennej a jest równa lub mniejsza od wartości
zmiennej b.");
}
</script>
</head>
<body>
...
</body>
</html>
```

Ćwiczenie 3.2

Zmodyfikuj kod z przykładu 3.20 w taki sposób, aby po porównaniu wartości zmiennych wyświetlana była odpowiedź w postaci:

a > *b*: Zmienna *a* jest większa od zmiennej *b*.

a = *b*: Zmienna *a* jest równa zmiennej *b*.

a < *b*: Zmienna *a* jest mniejsza od zmiennej *b*.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Deklaracja zmiennych</title>
</head>
<body>
<script>
var a = 12;
var b = 17;
if (a > b) {
  document.write("Zmienna a jest większa od zmiennej b.");
} else {
  if (a < b) {
    document.write("Zmienna a jest mniejsza od zmiennej b.");
  } else {
    document.write("Zmienna a jest równa zmiennej b.");
  }
}
</script>
</body>
</html>
```

Ćwiczenie 3.3

Przypisz zmiennym następujące wartości: *x* = 10; *y* = 17; *z* = 23.

Porównaj wartości podanych zmiennych i wyświetl wynik porównania wszystkich zmiennych w przeglądarce internetowej według podanego wzoru:

Zmienna *y* jest większa od zmiennej *x*, ale jest mniejsza od zmiennej *z*.

Zmienna *x* jest mniejsza od zmiennej *y* i jest mniejsza od zmiennej *z*.

Zmienna *z* jest większa od zmiennej *y* i jest większa od zmiennej *x*.

Ćwiczenie 3.4

Zdefiniuj trzy zmienne i przypisz im dowolne wartości. Wykorzystując poznane operatory, sprawdź, czy wszystkie zmienne są:

- dodatnie — jeżeli tak, wyświetl komunikat: `Wszystkie zmienne są dodatnie`,
- ujemne — jeżeli tak, wyświetl komunikat: `Wszystkie zmienne są ujemne`,
- jeżeli zmienne mają różne znaki, wyświetl komunikat: `Zmienne mają różne znaki`.

3.4.2. Instrukcja switch

Instrukcja `switch` jest instrukcją wyboru. Pozwala sprawdzić zestaw warunków i wykonać różne działania w zależności od wyników porównania.

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

Działanie instrukcji wygląda następująco:

„Sprawdź, jaką wartość ma *wyrażenie*, i jeżeli wynikiem jest *wartość1*, wykonaj *instrukcje1* i wyjdź z bloku `switch` (polecenie `break`). Jeżeli wynikiem jest *wartość2*, to wykonaj *instrukcje2* i wyjdź z bloku `switch`. Jeżeli wynikiem jest *wartość3*, to wykonaj *instrukcje3* i wyjdź z bloku `switch`. Jeżeli wartość jest inna, wykonaj *instrukcje4* i zakończ blok `switch`”.

Przykład 3.21

```
<script>
var a = 20;
var b = 7;
switch (a * b) {
  case 10:
    document.write("Wynik mnożenia wynosi 10.");
    break;
  case 40:
    document.write("Wynik mnożenia wynosi 40.");
    break;
  case 100:
    document.write("Wynik mnożenia wynosi 100.");
    break;
  default:
    document.write("Nieznany wynik mnożenia.");
}
</script>
```

Ćwiczenie 3.5

Utwórz nowy skrypt. Przypisz zmiennej `dzien` wartość bieżącego dnia tygodnia. Za pomocą instrukcji `switch` sprawdź, jaka wartość została przypisana do zmiennej, i wyświetl informację według podanego wzorca.

Wartość zmiennej `dzien`:

poniedziałek → Cały tydzień przed nami.

wtorek → Kiedy będzie wolne?

środa → Dopiero środek tygodnia.

czwartek → Już czwartek.

piątek → Wreszcie piątek.

sobota → Czas na odpoczynek.

niedziela → Jutro znowu poniedziałek.

3.4.3. Pętle

Pętle są używane do wykonywania powtarzających się czynności. W języku JavaScript występują następujące rodzaje pętli: `for`, `while`, `do ... while`.

Pętla for

Pętlę typu `for` wykorzystuje się, gdy znana jest liczba wykonań pętli oraz znany jest warunek, który musi być spełniony, aby kolejny raz wykonać pętlę. Składnia instrukcji jest następująca:

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

- *wyrażenie początkowe* — inicjuje zmienną, która jest używana jako licznik pętli,
- *wyrażenie warunkowe* — określa warunek, który musi być spełniony, aby pętla została wykonana kolejny raz,
- *wyrażenie modyfikujące* — modyfikuje zmienną, która jest licznikiem.

Przykład 3.22

```
<script>
for (var i = 0; i < 5; i++) {
  document.write("Pętla wykonana " + i + " raz/y<br>");
}
</script>
```

W uproszczonej postaci pętla może zostać pozbawiona wyrażenia modyfikującego.

Przykład 3.23

```
<script>
for (var i = 0; i < 5;) {
  document.write("Pętla wykonana " + i + " raz/y<br>");
  i++;
}
</script>
```

W przykładzie 3.23 zwiększanie licznika zostało przeniesione z pętli do bloku instrukcji. Należy pamiętać o tym, aby średnik występujący po wyrażeniu `i < 5` pozostał w pętli, ponieważ jest on niezbędny do jej prawidłowego działania.

W podobny sposób można postąpić z wyrażeniem początkowym, przenosząc je do bloku przed pętlą.

Przykład 3.24

```
<script>
var i = 0;
for (;i < 5;) {
  document.write("Pętla wykonana " + i + " raz/y<br>");
  i++;
}
</script>
```

W tym przypadku również średnik występujący przed wyrażeniem `i < 5` powinien pozostać w pętli. Podobnie jak poprzednio, jest on niezbędny do jej prawidłowego działania.

Ćwiczenie 3.6

Wykorzystując pętlę `for`, utwórz skrypt, który wyświetli w przeglądarce internetowej ciąg liczbowy powtórzony pięć razy w sposób pokazany na rysunku 3.2.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Pętla for</title>
</head>
<body>
<script>
for (var j = 1; j < 6; j++) {
  for (var i = 1; i < 6; i++) {
    document.write( i + "  ");
  }
  document.write("<br>");
}
</script>
</body>
</html>
```

1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5

Rysunek 3.2.
Ciąg liczbowy dla pętli for

Ćwiczenie 3.7

Wykorzystując pętlę `for`, utwórz skrypt, który wyświetli w przeglądarce internetowej ciąg liczbowy powtórzony w sposób pokazany na rysunku 3.3.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Pętla for</title>
</head>
<body>
<script>
var i = 1;
var z = 1;
for (var j = 1; j < 6; j++) {
  for (; i < 6; i++) {
    document.write(i + "  ");
  }
  z = z + 1;
  i = z;
  document.write("<br>");
}
</script>
</body>
</html>
```

1 2 3 4 5
2 3 4 5
3 4 5
4 5
5

Rysunek 3.3.
Ciąg liczbowy dla pętli for

Pętla while

Pętla `while` jest zwykle wykorzystywana wtedy, gdy liczba wykonywanych powtórzeń nie jest znana. Składnia instrukcji jest następująca:

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

Blok instrukcji jest wykonywany w pętli, dopóki *`wyrażenie warunkowe`* jest prawdziwe. Konstrukcja ta oznacza: „Dopóki *`wyrażenie warunkowe`* jest prawdziwe, wykonuj instrukcje”.

Przykład 3.25

```
<script>
var i = 0;
while (i++ < 5) {
  document.write("Pętla wykonana " + i + " raz/y<br>");
}
</script>
```

Ćwiczenie 3.8

Wykorzystując pętlę `while`, utwórz skrypt, który wyświetli w przeglądarce internetowej powtórzone bloki tekstu w sposób pokazany na rysunku 3.4. Wyświetlanie bloków zakończ dla `i = 10`.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Przykład</title>
</head>
<body>
<script>
var text = "";
var i = 0;
while (i < 10) {
  text += "<br>Liczba i jest równa " + i;
  document.write(text + "<br>");
  i++;
}
</script>
</body>
</html>
```

Liczba i jest równa 0

Liczba i jest równa 0
Liczba i jest równa 1

Liczba i jest równa 0
Liczba i jest równa 1
Liczba i jest równa 2

Liczba i jest równa 0
Liczba i jest równa 1
Liczba i jest równa 2
Liczba i jest równa 3

Liczba i jest równa 0
Liczba i jest równa 1
Liczba i jest równa 2
Liczba i jest równa 3
Liczba i jest równa 4

Liczba i jest równa 0
Liczba i jest równa 1
Liczba i jest równa 2
Liczba i jest równa 3
Liczba i jest równa 4
Liczba i jest równa 5

Rysunek 3.4.
Bloki tekstu dla pętli while

Pętla do ... while

Pętla `do ... while` jest odmianą pętli `while`. Jej składnia jest następująca:

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

Konstrukcja ta oznacza: „Wykonuj instrukcje, dopóki *`wyrażenie warunkowe`* jest prawdziwe".

W pętli `do ... while` blok instrukcji jest wykonywany co najmniej raz, nawet jeżeli warunek zapisany jako *`wyrażenie warunkowe`* jest fałszywy — ponieważ najpierw wykonywany jest ciąg instrukcji, a dopiero potem sprawdzany jest warunek.

Przykład 3.26

```
<script>
var i = 1;
do {
  document.write("Pętla wykonana" + i + "raz/y<br>");
} while (i++ <= 5);
</script>
```

Instrukcja break

Instrukcja `break` jest instrukcją modyfikującą zachowanie pętli. Służy do przerwania jej wykonywania.

Przykład 3.27

```
<script>
var i = 0, k = 5;
do {
  var j = k * i;
  document.write("Wynik " + j + "<br>");
  if (j > 30) break;
} while (i++ < 10);
</script>
```

Pętla `do ... while` będzie wykonywana, dopóki `i < 10`, ale jeżeli wartość zmiennej `j` obliczanej w pętli przekroczy `30`, nastąpi przerwanie wykonywania powtarzających się instrukcji i wyjście z pętli.

Ćwiczenie 3.9

Przypisz zmiennej `x` wartość 10. Za pomocą instrukcji `while` utwórz pętlę, która będzie w nieskończoność zwiększała wartość zmiennej `x` o 1 i wyświetlała wartości tej zmiennej. Przerwij wykonywanie pętli, gdy zmienna `x` osiągnie wartość 27.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Przykład</title>
<meta charset="UTF-8">
</head>
<body>
<script>
var x = 10;
while (true) {
  document.write("Wartość zmiennej i wynosi: " + x + "<br>");
  if (x++ >= 27) break;
}
</script>
</body>
</html>
```

Instrukcja continue

Instrukcja `continue`, podobnie jak `break`, służy do modyfikowania zachowania pętli. Po jej napotkaniu następuje przerwanie wykonywania bieżącej iteracji i przejście na jej początek.

Przykład 3.28

```
<script>
for (var i = 0; i <= 30; i++) {
  if ((i % 3) != 0) {
    continue;
  }
  document.write(i + "; ");
}
</script>
```

W wyniku wykonania podanego kodu zostaną wyświetlone liczby z zakresu od `0` do `30` podzielne przez 3. Jeżeli wynik dzielenia przez 3 nie jest liczbą całkowitą, następuje przerwanie wykonywania pętli i powrót na jej początek (liczby niepodzielne przez 3 nie będą wyświetlane).

Ćwiczenie 3.10

Wykorzystując instrukcje sterujące wykonaniem skryptu, utwórz skrypt, który będzie wyświetlał tabliczkę mnożenia w postaci podanej na rysunku 3.5. Wartość zmiennej `x` zmienia się od 1 do 10, a zmiennej `n` — od 1 do 9.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Tabliczka mnożenia</title>
<meta charset="UTF-8">
</head>
<body>
<script>
for (var n = 1; n <= 10; n++) {
  for (var m = 1; m < 10; m++) {
    document.write(n, "*", m, " = ", (n * m), "<br>");
  }
}
</script>
</body>
</html>
```

1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15

Rysunek 3.5. Tabliczka mnożenia

Ćwiczenie 3.11

Utwórz skrypt, który będzie wyświetlał liczby od 1 z interwałem równym 4. Zakończenie wyświetlania nastąpi, gdy suma wyświetlanych liczb przekroczy 100.

3.5. Funkcje

Funkcja w języku JavaScript to jedno lub więcej poleceń zgrupowanych w całość za pomocą nawiasów klamrowych `{}`. Tak zdefiniowana funkcja może zawierać listę argumentów umieszczonych w nawiasach okrągłych `()` i oddzielonych przecinkami oraz może zwracać wartość. Do funkcji można odwołać się poprzez nazwę.

3.5.1. Definiowanie funkcji

W języku JavaScript można definiować własne funkcje. Definicja funkcji musi zawierać słowo kluczowe `function`, po którym następuje nazwa funkcji, po czym w nawiasach okrągłych powinny zostać wymienione argumenty funkcji oddzielone przecinkami. W nawiasach klamrowych jest zapisywana treść funkcji, a na jej końcu powinna zostać wstawiona instrukcja `return`, określająca zwracaną przez funkcję wartość. Ponieważ funkcja zawsze zwraca jakąś wartość, to jeżeli wartość ta nie zostanie podana w jawny sposób (z wykorzystaniem instrukcji `return`), automatycznie zostanie zwrócona wartość `undefined`.

Instrukcja `return` powoduje przerwanie wykonywania poleceń funkcji i powrót do miejsca, z którego funkcja została wywołana. Definicja funkcji ma postać:

```
function nazwa_funkcji (argumenty_funkcji) {instrukcje}
```

Można definiować funkcje bezparametrowe:

```
function nazwa_funkcji () {instrukcje}
```

Przykład 3.29

```
function suma(a, b) {
  var c = a + b;
  return c;
}
```

Wywołanie funkcji

Wykonanie funkcji nastąpi, gdy „coś" ją wywoła. Może to być:

- zdarzenie (na przykład kliknięcie myszą przycisku),
- podanie w kodzie skryptu nazwy funkcji (wraz z jej argumentami),
- automatyczne wywołanie funkcji.

Przykład 3.30

```
<script>
function trzy(x) {
  for (var i = 0; i <= x; i++) {
    if ((i % 3) != 0)
    continue;
    document.write(i + "; ");
  }
  document.write("<br>");
```

```
}
trzy(90);
trzy(120);
</script>
```

W podanym przykładzie została zdefiniowana funkcja `trzy(x)`, która wyświetla ciąg liczb podzielnych przez 3. Argument `x` określa zakres wyświetlania liczb. Funkcja `trzy(x)` została wywołana dwukrotnie, raz z argumentem `x = 90`, drugi raz z argumentem `x = 120`.

Przykład 3.31

```
<script>
function wynik(x, y) {
  var s = x + y;
  return s;
}
var suma = wynik(19, 7) + 27;
document.write("Wynik dodawania: " + suma);
</script>
```

Zdefiniowana w przykładzie funkcja `wynik()` oblicza sumę dwóch liczb i zwraca ją jako wartość zmiennej `s` (`return s`). W skrypcie funkcja `wynik()` została wywołana z argumentami `19` i `7` w wyrażeniu, w którym obliczona zostaje wartość zmiennej `suma`.

Argumenty

Przy wywołaniu funkcji należy podać wartości jej argumentów. Jeżeli zostaną one pominięte, JavaScript przypisze argumentom wartość `undefined`. Jeżeli otrzyma ich więcej, niż jest wymagane, zignoruje niepotrzebne argumenty. Możliwe jest tworzenie funkcji, które będą miały zmienną liczbę argumentów. Wykorzystywana jest do tego tablica `arguments`, która jest automatycznie tworzona dla każdej funkcji.

Przykład 3.32

```
function lista_arg() {
  return arguments;
}
```

W wyniku wywołania funkcji bez podania parametrów nie zostanie zwrócona żadna wartość.

```
lista_arg();
```

Rezultat:

```
[ ]
```

W wyniku wywołania funkcji z parametrami zostanie zwrócona lista wartości.

```
lista_arg(1, 4, 5, 9, "A", "koło");
```

Rezultat:

```
[1, 4, 5, 9, "A", "koło"]
```

Wykorzystując tablicę `arguments`, można zdefiniować funkcję, która będzie sumowała dowolną liczbę parametrów.

Przykład 3.33

```
<script>
function suma_dow() {
  var i, wynik = 0;
  var l_param = arguments.length;
  for (i = 0; i < l_param; i++)
  {
    wynik += arguments[i];
  }
  return wynik;
}
</script>
```

Użyta w definicji funkcji właściwość `arguments.length` zwróci liczbę parametrów podanych podczas wywołania funkcji.

Wywołanie funkcji z różną liczbą parametrów powinno dać zawsze poprawny wynik, na przykład:

```
suma_dow(3, 5, 7);
```

da wynik 15,

```
suma_dow(1, 2, 3, 4, 5, 6, 7, 8, 9);
```

da wynik 45.

3.5.2. Zasięg zmiennych

Zasięg zmiennej jest to obszar, w którym można odwoływać się bezpośrednio do zmiennej. Zmienna może być:

- lokalna,
- globalna.

Zmienne **globalne** są widoczne w całym skrypcie. Są to zmienne, które zostały zdefiniowane poza funkcją.

Zmienne **lokalne** mają zasięg lokalny i są definiowane wewnątrz funkcji. Ich zasięg dotyczy tylko funkcji, w której zostały zdefiniowane, i poza nią nie są widoczne.

Jeżeli zmienna zostanie zadeklarowana bez użycia słowa kluczowego `var`, będzie miała zasięg globalny. Należy starać się ograniczać liczbę zmiennych globalnych i deklarować zmienne ze słowem kluczowym `var`.

Przykład 3.34

```
<script>
function suma_dow() {
  var i;
  wynik = 0;
  var l_param = arguments.length;
  for (i = 0; i < l_param; i++)
  {
    wynik += arguments[i];
  }
  return wynik;
}
suma_dow(3, 5, 7);
document.write("Suma argumentów: " + wynik);
</script>
```

W podanym przykładzie zmienna `wynik` zadeklarowana wewnątrz funkcji jest zmienną globalną, ponieważ została zadeklarowana bez użycia operatora `var`. Może zatem być wykorzystana w dalszej części skryptu.

Zasięg blokowy

Zadeklarowanie zmiennej za pomocą słowa kluczowego `let` powoduje, że taka zmienna ma zasięg blokowy. Oznacza to, że jest widoczna w bloku otoczonym nawiasami klamrowymi `{}`.

Przykład 3.35

```
<script>
let a = 10;
{
  let a = 15;
  document.write("Zmienna w bloku: " + a);
}
document.write("Zmienna globalna: " + a);
</script>
```

3.5.3. Funkcje wbudowane

W języku JavaScript istnieje duża grupa funkcji wbudowanych (predefiniowanych). Są to między innymi:

- `parseInt()`,
- `parseFloat()`,
- `isNaN()`,
- `isFinite()`,
- `alert()`.

parseInt()

Funkcja pobiera argument dowolnego typu (najczęściej tekstowego) i zamienia go na liczbę. Gdy operacja nie powiedzie się, zwraca wartość `NaN`. Funkcja może pobierać jeszcze drugi argument, określający podstawę liczby (dziesiętna, szesnastkowa, binarna). Jeżeli ten argument nie zostanie podany, domyślnie podstawa jest dziesiętna. Ale jeżeli pierwszy argument zaczyna się od `0x`, podstawa będzie szesnastkowa, natomiast gdy zaczyna się od `0`, podstawa będzie ósemkowa.

`NaN` oznacza, że wartość nie jest liczbą.

Przykład 3.36

Funkcja	Wynik
`parseInt("123");`	123
`parseInt("123AB", 16);`	74667
`parseInt("123", 8);`	83
`parseInt("123AB", 8);`	83
`parseInt("0377");`	255
`parseInt("0x373");`	883

Przykład 3.37

```
<!DOCTYPE html>
<html>
<head>
<title>Funkcja JS</title>
<meta charset="UTF-8">
<script>
var c1 = "12";
var c2 = "34";
var wynik = c1 + c2;
```

```
document.write("Zwykłe dodawanie ciągów: c1 + c2 = ");
document.write(wynik);
document.write("<br />");

c1 = parseInt(c1);
c2 = parseInt(c2);
wynik = c1 + c2;

document.write("Dodawanie po konwersji na liczby: c1 + c2 = ");
document.write(wynik);
document.write("<br>");

</script>
</head>
<body>
</body>
</html>
```

parseFloat()

Funkcja działa podobnie do `parseInt()`, ale można ją zastosować także do wartości ułamkowych. Pobiera argument dowolnego typu (najczęściej tekstowego) i zamienia go na liczbę, która może zawierać część ułamkową. Funkcja poprawnie interpretuje zapis liczby w postaci wykładniczej.

Przykład 3.38

```
parseFloat("432.37");
```

isNaN()

Funkcja sprawdza, czy wartość podana jako parametr nie jest liczbą. Za jej pomocą można na przykład zweryfikować, czy funkcji `parseInt()` udało się zamienić podaną wartość na liczbę. Funkcja zwraca wartość `true`, gdy argument wejściowy nie jest liczbą, lub `false`, gdy argument ten jest liczbą.

Przykład 3.39

Funkcja	Wynik
`isNaN(NaN)`	`true`
`isNaN(567)`	`false`
`isNaN(37.2)`	`false`
`isNaN(parseInt("zx23"))`	`true`

isFinite()

Funkcja sprawdza, czy wartość podana jako parametr to liczba różna od `Infinity` oraz od `NaN`. Gdy argument wejściowy ma wartość `Infinity` lub `NaN`, funkcja zwraca `false`, a gdy argument ten jest liczbą, zwraca `true`.

Przykład 3.40

Funkcja	Wynik
`isFinite(Infinity)`	`false`
`isFinite(-Infinity)`	`false`
`isFinite(67)`	`true`
`isFinite(2E12)`	`true`

alert()

Funkcji `alert()` nie ma w specyfikacji języka, ale można jej używać w środowisku przeglądarki. Służy do wyświetlania komunikatów w oknie dialogowym.

Przykład 3.41

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<script>
function pokaz(){
  alert("Uwaga, alarm!");
}
</script>
</head>
<body>
<input type="button" onclick="pokaz()" value="Pokaż okno alarmu">
</body>
</html>
```

W podanym przykładzie skrypt został umieszczony wewnątrz kodu HTML. W sekcji `<body>` przy użyciu znacznika `<input type="button">` utworzono przycisk, do którego zostało przypisane zdarzenie `onclick` (przy kliknięciu myszą), i została z nim związana funkcja `pokaz()` zdefiniowana w skrypcie.

3.6. Obiekty

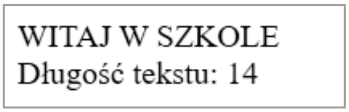
Język JavaScript jako język zorientowany obiektowo udostępnia wiele wbudowanych obiektów. Daje również możliwość odczytywania ich właściwości oraz korzystania z ich metod. Właściwości obiektu reprezentują jego cechy (na przykład liczbę znaków łańcucha, rozmiary okna) lub pozwalają określić jego stan. Nazywane są również polami obiektu, zmiennymi lub zmiennymi składowymi. Metody to funkcje, które wykonują różne operacje na obiekcie.

Do właściwości i metod można się odwołać podobnie jak do zwykłych zmiennych i funkcji, trzeba tylko przed ich nazwą umieścić nazwę obiektu, którego są elementami (na przykład nazwę zmiennej, która przechowuje dany obiekt), i kropkę.

Przykład 3.42

```
var napisz = "Witaj w szkole";
document.write(napisz.toUpperCase() + "<br>");
document.write("Długość tekstu: " + napisz.length);
```

W podanym przykładzie zmienna `napisz` przechowuje obiekt. Funkcja `toUpperCase()` jest metodą obiektu, a `napisz.length` jego właściwością. Wynikiem będzie wypisanie tekstu pokazanego na rysunku 3.6.



Rysunek 3.6. Zastosowanie metody obiektu i jego właściwości

Każdy element strony internetowej jest traktowany jako obiekt związany z obiektem `document`. Obiekty języka JavaScript zawierają informacje opisujące stronę i jej środowisko.

3.6.1. Tworzenie obiektów

Obiekty w języku JavaScript można tworzyć:

- używając literału obiektu,
- za pomocą konstruktora obiektu i słowa kluczowego `new`.

Tworzenie obiektu z użyciem literału

Aby utworzyć nowy obiekt z użyciem literału, należy skorzystać z konstrukcji, która zdefiniuje nazwę obiektu oraz pozwoli utworzyć jego właściwości i metody.

Ponieważ obiekty są zmiennymi, które mogą zawierać wiele wartości, można utworzyć zmienną, która jest obiektem. Każda jej wartość będzie właściwością obiektu, zostanie zapisana jako para nazwa i wartość (*nazwa: wartość*) i umieszczona w nawiasach

klamrowych {}. Działania, jakie można wykonywać na takim obiekcie, zostaną zdefiniowane jako metody i zapisane jako definicje funkcji.

Przykład 3.43

```
var osoba = {
  nazwisko: "Nowacki",
  imie: "Marek",
  zawod: "informatyk",
  pokaz: function() {
    document.write(this.nazwisko + ' ' + this.imie);
  }
};
```

Utworzony obiekt ma trzy właściwości (`nazwisko`, `imie`, `zawod`) i jedną metodę, `pokaz`, która wyświetla nazwisko i imię. Metoda jest funkcją zdefiniowaną w obrębie obiektu. Przy definiowaniu obiektu deklarowane właściwości i funkcje muszą być oddzielone przecinkami. Słowo kluczowe `this` pozwala odwołać się do właściwości lub metod danego obiektu z jego wnętrza. W tym wypadku metoda `pokaz` odwołuje się do właściwości obiektu `osoba`.

Dostęp do właściwości obiektu uzyskamy po podaniu nazwy obiektu i jego właściwości oddzielonych znakiem kropki (na przykład `osoba.nazwisko`). W ten sam sposób można wywołać metodę (`osoba.pokaz`).

Przykład 3.44

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Obiekt osoba</title>
</head>
<body>
<script>
var osoba = {
  nazwisko: "Nowacki",
  imie: "Marek",
  zawod: "informatyk",
```

```
  pokaz: function() {
    document.write(this.nazwisko + ' ' + this.imie);
  }
};
osoba.pokaz();
</script>
</body>
</html>
```

W podanym przykładzie po utworzeniu obiektu `osoba` za pomocą konstrukcji `osoba.pokaz()` została wywołana metoda, która wyświetli nazwisko i imię osoby.

Dla istniejących obiektów można deklarować nowe właściwości i metody.

Przykład 3.45

```
var osoba = {
  nazwisko: "Nowacki",
  imie: "Marek",
  zawod: "informatyk",
  pokaz: function() {
    document.write(this.nazwisko + ' ' + this.imie);
  }
};
osoba.wiek = 19;
osoba.wypisz_wiek = function() {
  document.write('Wiek: ' + this.wiek + ' lat');
}
osoba.wypisz_wiek();
```

W podanym przykładzie po utworzeniu obiektu `osoba` i zdefiniowaniu jego właściwości i metod zostały zadeklarowane nowa właściwość `wiek` oraz nowa metoda `wypisz_wiek()`. W kolejnym kroku została wywołana nowa metoda `osoba.wypisz_wiek()`, za pomocą której został wyświetlony wiek osoby.

Ćwiczenie 3.12

Utwórz w języku JavaScript obiekt opisujący pojazd. Zdefiniuj dla niego właściwości: `marka`, `model`, `rok_produkcji`, `przebieg` oraz metodę `wyswietlDane()` do wyświetlenia marki, modelu i roku produkcji. Wywołaj metodę `wyswietlDane()`, a następnie dodaj właściwość `numer_rej` i metodę `wyswietlDetal()` do wyświetlenia marki samochodu i numeru rejestracyjnego.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Obiekt - literał</title>
</head>
<body>
<h2>Ćwiczenie - Tworzenie obiektu za pomocą literału</h2>
<script>
var pojazd = {
  marka: "Ford",
  model: "Mustang",
  rok_produkcji: "2020",
  przebieg: 15000,
  wyswietlDane: function() {
    document.write(this.marka + ' ' + this.model + ', rok produkcji: '
+ this.rok_produkcji + "<br>");
  }
};
pojazd.wyswietlDane();

pojazd.numer_rej = "WA34789";
pojazd.wyswietlDetal = function() {
  document.write('Marka: ' + this.marka + ', numer rejestracyjny: '
+ this.numer_rej);
}
pojazd.wyswietlDetal();

</script>
</body>
</html>
```

3.6.2. Tworzenie obiektów z użyciem konstruktora

W języku JavaScript istnieje możliwość tworzenia wielu obiektów mających podobne właściwości. W tym celu można posłużyć się konstruktorem obiektu. Konstruktor przypomina zwykłą funkcję.

Przykład 3.46

```
function Klient(nazwisko_k, imie_k, zawod_k) {
  this.nazwisko = nazwisko_k;
  this.imie = imie_k;
  this.zawod = zawod_k;
  this.wypisz = function() {
    alert(this.nazwisko + ' ' + this.imie);
  }
}
```

Został utworzony konstruktor o nazwie `Klient` z właściwościami `nazwisko`, `imie`, `zawod` oraz metodą `wypisz()`. Właściwościom obiektu zostały przypisane wartości parametrów. Użyte słowo kluczowe `this` odnosi się do aktualnego obiektu i pozwala na przypisanie wartości parametru do odpowiedniego pola tego obiektu.

Słowo kluczowe new

Do utworzenia nowego obiektu na podstawie konstruktora stosowane jest słowo kluczowe `new`.

Przykład 3.47

```
var osoba1 = new Klient('Kowalski', 'Jan', 'kierowca');
var osoba2 = new Klient('Nowak', 'Anna', 'sekretarka');
```

Powstały dwa nowe obiekty `osoba1` i `osoba2` należące do klasy `Klient`.

Właściwości obiektu istnieją w porządku, w jakim zostały zdefiniowane. Można się do nich odwoływać na dwa sposoby:

```
nazwa_obiektu.nazwa_właściwości
```

lub

```
nazwa_obiektu["nazwa_właściwości"]
```

Przykład 3.48

```
osoba1.nazwisko;
osoba1.["nazwisko"];
```

Ćwiczenie 3.13

Utwórz konstruktor obiektu `Uczen` z właściwościami: `nazwisko`, `imie`, `wiek`, `klasa`, `sport` oraz metodę `wypisz()`, która wyświetli właściwości: `nazwisko`, `imie`, `wiek` oraz `sport`. Na podstawie konstruktora utwórz pięć obiektów (uczniów), które będą zawierały `nazwisko`, `imie`, `wiek`, `klasa`, `sport`. Przy tworzeniu obiektów posłuż się zaprojektowanym konstruktorem obiektu. Dla każdego ucznia wyświetl jego nazwisko, imię, wiek oraz sport, jaki uprawia.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Konstruktor</title>
</head>
<body>
<h2>Ćwiczenie - Konstruktor</h2>
<script>
function Uczen(nazwisko, imie, wiek, klasa, sport) {
  this.nazwisko_u = nazwisko;
  this.imie_u = imie;
  this.wiek_u = wiek;
  this.klasa_u = klasa;
  this.sport_u = sport;
  this.wypisz = function() {
    document.write(this.nazwisko_u + ' ' + this.imie_u + ' '
+ this.wiek_u + ' lat, sport: ' + this.sport_u + '.' + "<br>");
  }
}

var uczen1 = new Uczen("Nowak", "Paweł", 12, "5b", "tenis");
var uczen2 = new Uczen("Polak", "Anna", 13, "6a", "pływanie");
var uczen3 = new Uczen("Czaja", "Maciej", 13, "6a", "siatkówka");
var uczen4 = new Uczen("Malak", "Julia", 12, "5c", "gimnastyka");
var uczen5 = new Uczen("Wojak", "Michał", 13, "6b", "pływanie");
```

```
uczen1.wypisz();
uczen2.wypisz();
uczen3.wypisz();
uczen4.wypisz();
uczen5.wypisz();


</script>
</body>
</html>
```

Właściwość prototype

Nie można dodawać nowych właściwości i metod do konstruktora obiektów tak, jak dodaje się właściwości i metody do istniejącego obiektu. Nowe właściwości i metody muszą być umieszczane w definicji konstruktora.

Sposobem na deklarowanie nowych metod i właściwości dla konstruktora jest wykorzystanie właściwości `prototype`.

Przykład 3.49

```
function Klient(nazwisko, imie) {
  this.nazwisko_k = nazwisko;
  this.imie_k = imie;
}
Klient.prototype.zawod = 'kierowca';
Klient.prototype.pisz_dane = function() {
  document.write(this.nazwisko_k + ' ' + this.imie_k);
}
var osoba1 = new Klient("Malinowski", "Oskar", "kierowca");
osoba1.pisz_dane();
```

W definicji konstruktora zostały zadeklarowane dwie właściwości. Po użyciu właściwości `prototype` została dodana metoda `pisz_dane()` oraz właściwość `zawod`. Od tej pory każdy obiekt, który będzie tworzony na podstawie konstruktora `Klient`, będzie miał tę dodatkową właściwość i metodę.

Właściwość `prototype` może być również wykorzystana do dołączania dodatkowych metod lub właściwości do istniejących obiektów.

Ćwiczenie 3.14

Uzupełnij kod utworzony w ćwiczeniu 3.13. Zmień imię ucznia `Nowak` z `Paweł` na `Karol` i klasę uczennicy `Malak` z `5c` na `5a`. Do konstruktora dodaj metodę `wypisz-Klasa()`, która wyświetli nazwisko, imię ucznia i klasę. Dla każdego ucznia wywołaj utworzoną metodę.

3.7. Obiekty wbudowane języka JavaScript

Niektóre predefiniowane obiekty języka JavaScript to:

- `String` — łańcuch tekstowy,
- `Array` — tablica,
- `Date` — data,
- `Math` — obiekt do przeprowadzania operacji matematycznych.

3.7.1. Obiekt String

Obiektem zawsze występującym w języku JavaScript jest obiekt `String`. Ma on jedną właściwość, `length`, określającą długość łańcucha.

Przykład 3.50

```
var tekst = "Obiekty języka JavaScript";
var dlug = tekst.length;
```

Zmiennej `dlug` przypisana zostanie wartość `25` określająca długość tekstu.

Obiekt `String` ma dwa typy metod.

Pierwszy odnosi się do utworzonego łańcucha, przykładem jest metoda `substring()`, która zwraca podzbiór tego łańcucha. Jej parametry określają położenie początku i końca podzbioru.

Przykład 3.51

```
var tekst = "Obiekty języka JavaScript";
var x = tekst.substring(15, 19);
```

Zostanie zwrócony łańcuch `Java`.

Metodami, które można wykorzystać do zmiany wielkości liter, są: `toUpperCase()` i `toLowerCase()`. Metoda `toUpperCase()` zamienia wszystkie litery ciągu na duże, a metoda `toLowerCase()` zamienia wszystkie litery ciągu na małe.

Przykład 3.52

```
var tekst = "Obiekty języka JavaScript";
var x = tekst.toLowerCase();
```

Zostanie zwrócony łańcuch `obiekty języka javascript`.

Przykład 3.53

```
var tekst = "Obiekty języka JavaScript";
var x = tekst.toUpperCase();
```

Zostanie zwrócony łańcuch `OBIEKTY JĘZYKA JAVASCRIPT`.

Do istniejących obiektów można dodawać nowe właściwości i można definiować dla nich nowe metody.

Do obiektu `String` dołączymy dodatkową funkcję, której wywołanie spowoduje, że pierwsza litera łańcucha zostanie zmieniona na dużą.

Przykład 3.54

```
String.prototype.duzaLitera = function() {
  return this.charAt(0).toUpperCase() + this.substr(1);
}
```

Metoda `charAt()` zwraca znak z pierwszej pozycji łańcucha znaków. Natomiast metoda `substr()` zwraca podzbiór łańcucha znaków. Jako parametr tej metody został podany indeks pierwszego znaku podzbioru. Zadeklarowana metoda została dołączona do obiektu `String` i od tej pory każdy nowy tekst będzie miał metodę, która zamieni jego pierwszą literę na dużą.

Przykład 3.55

```
var tx1 = 'komputer';
document.write(tx1.duzaLitera());
```

W wyniku wykonania skryptu wyświetli się tekst `Komputer`.

3.7.2. Obiekt Array

Tablice służą do przechowywania wielu zmiennych. W języku JavaScript do pracy z tablicami można używać wbudowanego obiektu `Array`. Ma on metody do manipulowania tablicami zmiennych.

Aby utworzyć nową tablicę, należy zadeklarować obiekt `Array` w postaci:

```
var nazwaTablicy = new Array();
```

lub

```
var nazwaTablicy = [];
```

Jeżeli w nawiasach zostanie podana liczba `n`, to zostanie utworzona tablica zawierająca `n` pustych elementów.

Przykład 3.56

```
var tab1 = new Array(10);
var tab2 = [15];
```

W pierwszym przypadku uzyskamy tablicę zawierającą 10 pustych elementów, w drugim powstanie tablica zawierająca jeden element o wartości 15.

Tablicę można również tworzyć, wstawiając do niej konkretne wartości.

Przykład 3.57

```
var tab3 = new Array('Anna', 'Adam', 'Piotr', 'Ewa');
var tab4 = ['Paweł', 'Marcin', 'Ela'];
```

Żeby uzyskać dostęp do elementów tablicy, należy podać numer indeksu danego elementu. Elementy są indeksowane od zera.

Przykład 3.58

```
document.write(tab3[2]);
```

W wyniku wyświetli się wartość `Piotr`.

Aby dodać nową wartość do tablicy, należy przypisać tę wartość do odpowiedniego indeksu tablicy.

Przykład 3.59

```
var tab5 = ['kot', 'pies', 'koń'];
tab5[3] = 'mysz';
tab5[4] = 'chomik';
document.write(tab5[0] + ' i ' + tab5[3]);
```

W wyniku wyświetli się tekst `kot i mysz`.

Dzięki właściwości `length` można określić, z ilu elementów składa się tablica. Jest to bardzo przydatna właściwość, szczególnie gdy chcemy utworzyć pętlę odczytującą wszystkie elementy tablicy.

Przykład 3.60

```
var imie = ['Anna', 'Adam', 'Piotr', 'Ewa', 'Paweł', 'Marcin', 'Ela'];
for (var i = 0; i < imie.length; i++) {
  document.write(imie[i] + "<br>");
}
```

W wyniku zostaną wyświetlone po kolei wszystkie elementy tablicy.

Zadanie 3.1

Zmień zapis skryptu podanego w przykładzie 3.60, tak aby wyświetlone zostały elementy tabeli od ostatniego do pierwszego. Pamiętaj, że indeksowanie elementów rozpoczyna się od zera.

Tablice wielowymiarowe

W języku JavaScript można również tworzyć tablice wielowymiarowe. Wtedy element tablicy jest opisywany za pomocą indeksu określającego jego położenie w wierszu i kolumnie.

Przykład 3.61

```
var Osoby = [];
Osoby[0] = ['Anna', 'Nowak'];
Osoby[1] = ['Adam', 'Kowal'];
Osoby[2] = ['Piotr', 'Ogórek'];
Osoby[3] = ['Ewa', 'Lisowska'];
document.write('imię: ' + Osoby[0][0] + ', nazwisko: ' + Osoby[0][1]
+ "<br>");
document.write('imię: ' + Osoby[1][0] + ', nazwisko: ' + Osoby[1][1]
+ "<br>");
document.write('imię: ' + Osoby[2][0] + ', nazwisko: ' + Osoby[2][1]
+ "<br>");
document.write('imię: ' + Osoby[3][0] + ', nazwisko: ' + Osoby[3][1]);
```

Łączenie elementów tablicy

Za pomocą metody `join()` można łączyć elementy tablicy w jeden ciąg znaków. W metodzie tej można opcjonalnie podać parametr, który określi znak oddzielający kolejne elementy tablicy. Jeżeli nie zostanie podana wartość tego parametru, domyślnym znakiem będzie przecinek.

Przykład 3.62

```
var Tablica = new Array('Anna', 'Adam', 'Piotr');
document.write(Tablica.join() + "<br>");
document.write(Tablica.join(" - ") + "<br>");
```

Odwracanie kolejności elementów tablicy

Za pomocą metody `reverse()` można odwrócić kolejność elementów tablicy.

Przykład 3.63

```
var Tablica = new Array('Anna', 'Adam', 'Piotr');
document.write(Tablica.join() + "<br>");
Tablica.reverse()
document.write(Tablica.join() + "<br>");
```

Sortowanie

Do sortowania elementów tablicy służy metoda `sort()`.

Przykład 3.64

```
var Tablica = new Array('Paweł', 'Anna', 'Maria', 'Adam', 'Piotr');
Tablica.sort();
document.write(Tablica.join());
```

Przykład 3.65

```
var Tablica = new Array(3000, 4567, 12459, 406745);
Tablica.sort();
document.write(Tablica.join());
```

Domyślnie tablica jest sortowana leksykograficznie. Powoduje to, że liczba `12459` będzie mniejsza od `4567`, ponieważ w liczbie `12459` cyfra na pierwszej pozycji jest mniejsza. Aby to zmienić, można sortować tablicę według własnych kryteriów. Należy skorzystać z dodatkowego parametru metody `sort()`. Parametrem będzie własna funkcja sortująca. Tworząc taką funkcję, należy pamiętać o trzech zasadach:

- jeżeli `funkcja(a, b)` zwróci wartość mniejszą od `0`, to wartości `a` zostanie nadany indeks mniejszy od indeksu przyznanego wartości `b`,
- jeżeli `funkcja(a, b)` zwróci wartość równą `0`, to wartości indeksów pozostaną bez zmian,
- jeżeli `funkcja(a, b)` zwróci wartość większą od `0`, to wartości `a` zostanie nadany indeks większy od indeksu przyznanego wartości `b`.

Przykład 3.66

```
function porownaj(a, b) {
  return a - b;
}
var Tablica = new Array(27, 100, 10, 450, 1654, 320);
document.write('Bez sortowania: ' + Tablica.join());
document.write("<br>" + 'Sortowanie domyślne: ');
Tablica.sort();
```

```
document.write(Tablica.join());
document.write("<br>" + 'Sortowanie poprawne: ');
Tablica.sort(porownaj);
document.write(Tablica.join());
```

W podanym przykładzie została zdefiniowana funkcja `porownaj(a, b)`, która zwróci wartość mniejszą od zera, równą zero lub większą od zera. W zależności od zwróconej wartości elementy tablicy zostaną poprawnie uporządkowane od wartości najmniejszej do największej.

Zadanie 3.2

Utwórz tablicę, której elementy będą zawierały cyfry i litery. Zdefiniuj funkcję, która pozwoli uporządkować elementy tablicy w ten sposób, że będą w niej umieszczone najpierw litery w kolejności alfabetycznej, a następnie cyfry w kolejności od wartości najmniejszej do największej.

3.7.3. Obiekt Date

Kolejnym obiektem języka JavaScript jest obiekt specjalny `Date`, który służy do przechowywania wartości daty i czasu. Za jego pomocą można odczytać wartość daty i czasu, można też rozłożyć te wartości na części, odczytując oddzielnie dzień, miesiąc, rok itp. Można również te części niezależnie modyfikować.

Aby odczytać bieżącą datę i czas, należy utworzyć obiekt `Date` bez parametrów.

Przykład 3.67

```
var data = new Date();
```

Można utworzyć obiekt z określoną liczbą parametrów. Tych parametrów może być od dwóch do siedmiu (rok, miesiąc, dzień, godzina, minuty, sekundy, milisekundy). Taki obiekt będzie zawierał ściśle określoną wartość daty i godziny.

Przykład 3.68

```
var data = new Date(2019, 2, 27);
```

W języku JavaScript wartości daty i czasu są przechowywane w formacie **timestamp**, czyli jako liczba milisekund, które upłynęły od północy 1 stycznia 1970 roku.

Do konwersji obiektu `Date` na tekst służy kilka funkcji:

- `toString()` — zwraca datę, czas oraz informacje o strefie czasowej w języku angielskim,
- `toLocaleString()` — zwraca datę i czas dla bieżących ustawień regionalnych,
- `toUTCString()` — zwraca datę, czas oraz informacje o strefie czasowej dla formatu UTC (*Universal Coordinated Time*),

- `toGMTString()` — działa jak funkcja `toUTCString()`,
- `toDateString()` — zwraca tylko datę w języku angielskim,
- `toLocaleDateString()` — zwraca tylko datę dla bieżących ustawień regionalnych,
- `toTimeString()` — zwraca tylko czas w języku angielskim,
- `toLocaleTimeString()` — zwraca tylko czas dla bieżących ustawień regionalnych.

Jednym z przykładów wykorzystania języka JavaScript na stronach WWW jest wyświetlanie daty i czasu jako elementu strony internetowej.

Przykład 3.69

```
<!DOCTYPE html>
<html>
<head>
<title>JavaScript - Data i czas</title>
<meta charset="UTF-8">
<body>
<h2>Wyświetlam bieżącą datę i czas</h2>
<p>
<script>
var data_n = new Date();
var data_l = data_n.toString();
var data_u = data_n.toGMTString();
var data_r = data_n.toLocaleString();
document.write("<b>Czas lokalny:</b> " + data_l + "<br>");
document.write("<b>Czas uniwersalny:</b> " + data_u + "<br>");
document.write("<b>Czas regionalny:</b> " + data_r + "<br>");
</script>
</p>
</body>
</html>
```

Utworzonej zmiennej o nazwie `data_n` przypisany został obiekt `Date`. Zmienne `data_l`, `data_u` i `data_r` zawierają tekstową postać czasu lokalnego, uniwersalnego i regionalnego odpowiadającego wartości przechowywanej w zmiennej `data_n`. Do wyświetlenia wartości otrzymanych zmiennych została użyta funkcja `document.write`. W rezultacie na stronie wyświetlą się informacje o czasie lokalnym, uniwersalnym i regionalnym (rysunek 3.7).

Rysunek 3.7. Wyświetlenie na stronie informacji o dacie i czasie

Ćwiczenie 3.15

Zdefiniuj funkcję, która będzie wyświetlała datę i czas po polsku, na przykład tak jak na rysunku 3.8.

Rysunek 3.8. Data i czas wyświetlane na stronie internetowej

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Data</title>
</head>
<body>
<h2>Wyświetlam bieżącą datę i czas po polsku</h2>
<p>
<script>
function data_czas() {
  var miesiace = ["stycznia", "lutego", "marca", "kwietnia",  "maja",
"czerwca", "lipca", "sierpnia", "września", "października", "listopada",
"grudnia"];
  var dni = ["Niedziela", "Poniedziałek", "Wtorek", "Środa", "Czwartek",
"Piątek", "Sobota"];
  var data = new Date();
```

```
  var rok = data.getFullYear();
  var mies = data.getMonth();
  var nr_dzien = data.getDate();
  var dzien = data.getDay();
  var godz = data.getHours();
  var min = data.getMinutes();
  var sek = data.getSeconds();

  if (min < 10) {
    min = "0" + min;
  }

  if (sek < 10) {
    sek = "0" + sek;
  }

  var p_data_czas = dni[dzien] + ", " + nr_
dzien + " " + miesiace[mies] + " " + rok +
" roku<br>" + "Godzina: " + godz + ":" + min + ":" + sek;
  document.write(p_data_czas);
}
document.write("<b>Dzisiaj jest: <br>");
data_czas();
document.write("</b>");

</script>
</p>
</body>
</html>
```

Zadanie 3.3

Utwórz stronę internetową, na której data i czas będą wyświetlane w postaci kartki z kalendarza.

3.8. Obiekty DOM

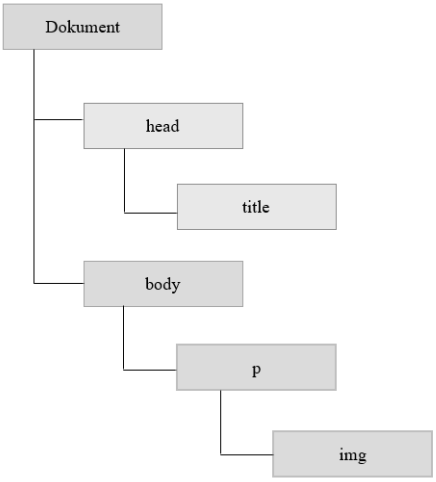
DOM (ang. *Document Object Model*) to sposób reprezentacji złożonych dokumentów XML i HTML w postaci modelu obiektowego.

Gdy kod strony jest wczytywany do przeglądarki, przeglądarka zamienia ciąg znaków na stronę internetową. Informacje na temat interpretacji kodu HTML przeglądarka przechowuje w elementach będących obiektami (są to na przykład informacje o tym, które elementy przedstawić w postaci nagłówków, paragrafów itp.). Obiekty te tworzą obiektowy model dokumentu.

DOM opisuje hierarchię obiektów na stronie oraz udostępnia metody i właściwości, które umożliwiają manipulowanie tymi obiektami. W tej hierarchii na samej górze znajduje się okno przeglądarki, czyli obiekt `window`. Zawiera ono wszystkie inne obiekty, funkcje i właściwości strony. W oknie znajduje się obiekt `document`, czyli otwarta strona internetowa. W obiekcie `document` znajdują się obiekty strony. W skryptach definiujemy różne działania związane z istniejącymi obiektami, czyli przez skrypty manipulujemy obiektami strony internetowej. Dzięki skryptom można wczytać nową stronę do przeglądarki, zmienić elementy dokumentu, otwierać okna lub modyfikować tekst na stronie. Dzięki DOM język JavaScript staje się narzędziem tworzenia dynamicznych stron internetowych.

Przykład 3.70

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Tytuł strony</title>
</head>
<body>
<p>Moje góry
<img src="obraz.jpg" alt="obraz">
</p>
</body>
</html>
```



Rysunek 3.9.
Hierarchia obiektów przykładowej strony internetowej

Podany w przykładzie dokument można rozrysować w postaci drzewa (rysunek 3.9). Na samej górze jest dokument HTML, niżej znajdują się węzły (`nodes`).

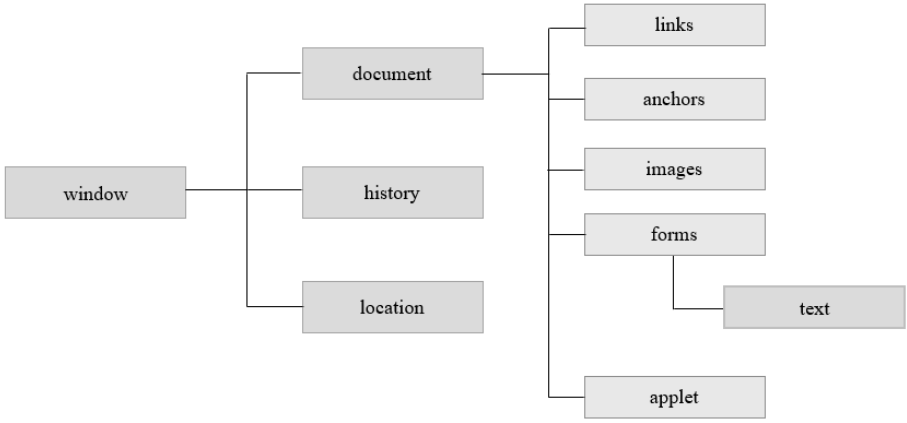
3.8.1. Hierarchia obiektów DOM

Odwołując się do obiektu, należy używać nazw obiektów nadrzędnych oddzielonych kropkami, po których następuje nazwa wybranego obiektu.

Przykład 3.71

```
window.document.links
```

Wycinek hierarchii DOM z najważniejszymi obiektami strony internetowej został pokazany na rysunku 3.10.



Rysunek 3.10. Hierarchia obiektów DOM

3.8.2. Obiekty przeglądarki

Dla każdej strony internetowej zdefiniowane są następujące obiekty:

- `window`,
- `navigator`,
- `document`,
- `history`,
- `location`.

Obiekt window

Obiektem nadrzędnym dla wszystkich obiektów jest obiekt `window`, który reprezentuje okno przeglądarki. W danej chwili może istnieć wiele obiektów `window`. Każdy z nich reprezentuje otwarte okno przeglądarki. `window` jest najważniejszym obiektem w hierarchii, dlatego odwołanie do jego właściwości lub metod nie wymaga podania nazwy obiektu. Tworzony jest automatycznie podczas otwierania okna przeglądarki. Ma wiele właściwości przydatnych podczas tworzenia strony.

Do otwierania nowego okna używamy metody `open()`. Parametry metody to adres URL otwieranej strony oraz nazwa wewnętrzna okna (nie należy mylić jej z nazwą wyświetlaną przez przeglądarkę zdefiniowaną metatagiem `<title></title>`):

```
window.open('http://helion.pl', 'Wydawnictwo');
```

Do określania rozmiaru okna mogą być używane właściwości:

- `window.innerHeight` — wysokość okna przeglądarki,
- `window.innerWidth` — szerokość okna przeglądarki.

Do zamknięcia okna wykorzystywana jest metoda `close()`.

Do sterowania wykonywaniem kodu służą dwie metody:

- `window.setTimeout()`,
- `window.setInterval()`.

Metoda window.setTimeout()

Metoda `setTimeout` przyjmuje dwa parametry. Pierwszy parametr jest ciągiem znaków lub referencją do funkcji, która ma się wykonać tylko raz. Drugi parametr to czas, po którym ma zostać wywołany kod podany jako pierwszy parametr.

```
window.setTimeout(kod, opóźnienie);
```

```
window.setTimeout(funkcja, opóźnienie);
```

Przykład 3.72

```
window.setTimeout(function() {
  alert("Uwaga!!!");
}, 1000);
```

Przykład 3.73

```
setTimeout("showtime()",1000);
```

Metoda window.setInterval()

Wywołanie metody `setInterval()` daje podobny efekt do wywołania metody `setTimeout()`. Różnica polega na tym, że w metodzie `setInterval()` podany kod będzie wykonywany wielokrotnie (w teorii do nieskończoności), co określony interwał czasowy.

```
window.setInterval(kod, opóźnienie);
```

```
window.setInterval(funkcja, opóźnienie);
```

Przykład 3.74

```
window.setInterval("mojCzas()", 1000);
```

Metoda window.clearInterval()

Do zakończenia wykonywanej co określony czas akcji zadanej w metodzie `setInterval()` służy metoda `clearInterval(int)`. Jako parametru tej metody należy użyć identyfikatora zwróconego przez metodę `setInterval()`.

Przykład 3.75

```
<!DOCTYPE html>
<html>
<head>
<title>Metody obiektu Window</title>
<meta charset="UTF-8">
<script>
var licz = 0;
var Id = window.setInterval(
  function() {
    alert('działa');
    licz++;
    if (licz >= 5) {
      window.clearInterval(Id);
    }
  }, 3000);
</script>
</head>
<body>
</body>
</html>
```

W podanym przykładzie jako kod do wykonania w metodzie `setInterval()` została zadeklarowana funkcja `alert()`, która wyświetli w oknie dialogowym komunikat `działa`. Komunikat ten powinien być wyświetlany w nieskończoność, ale instrukcja `if` sprawdza stan zmiennej `licz` i po jego pięciokrotnym wyświetleniu zostanie wywołana metoda `clearInterval()`, która spowoduje zakończenie wykonywania akcji zadanej w metodzie `setInterval()`.

navigator

Pozwala na dostęp do informacji dotyczących przeglądarki. Służy do ustalenia wersji przeglądarki, jaką posługuje się użytkownik. Właściwości tego obiektu mogą być tylko odczytywane. Najczęściej korzysta się z niego do sprawdzenia, czy przeglądarka jest odpowiednia do zastosowanej wersji języka JavaScript.

Przykład 3.76

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Jaka przeglądarka</title>
</head>
<body>
<script>
document.write("Masz przeglądarkę: ");
document.write(navigator.appName + "<br>");
document.write("Język przeglądarki: ");
document.write(navigator.language + "<br>");
document.write("Platforma: ");
document.write(navigator.platform);
</script>
</body>
</html>
```

Właściwość `appName` zawiera nazwę przeglądarki, `language` — język przeglądarki, a właściwość `platform` — nazwę platformy.

Obiekt document

Obiekt `document` reprezentuje stronę internetową (zawiera informacje o bieżącym dokumencie HTML). Jest on potomkiem obiektu `window`. Poprzez właściwości obiektu `document` mamy wpływ na elementy strony (na przykład kolor tła, kolor czcionki). Jego metody umożliwiają wyświetlenie na przykład tekstu w oknie przeglądarki.

Za pomocą polecenia `window.document` można odwołać się do bieżącego dokumentu. Można to zrobić również przy użyciu polecenia `document`. Odwołanie nastąpi do bieżącego dokumentu w bieżącym oknie. Jeżeli zostało otwartych kilka okien, to aby określić, do którego dokumentu powinno nastąpić odwołanie, należy podać nazwę okna i nazwę dokumentu.

Informacje o bieżącym dokumencie otrzymamy, gdy odwołamy się do właściwości i metod obiektu `document`.

- `document.URL` — zwraca adres URL dokumentu jako ciąg tekstu,
- `document.title` — zwraca tytuł strony zdefiniowany w znaczniku `<title>`,
- `document.lastModified` — zwraca datę ostatniej modyfikacji strony,

- `document.bgColor` — określa kolor tła dokumentu ustawianego atrybutem `bgcolor` znacznika `<body>`,
- `document.fgColor` — określa kolor pierwszego planu dokumentu ustawianego atrybutem `text` znacznika `<body>`,
- `document.linkColor` — określa kolor łącza w dokumencie ustawianego atrybutem `link`,
- `document.alinkColor` — określa kolor łącza w dokumencie ustawianego atrybutem `alink`,
- `document.vlinkColor` — określa kolor łącza w dokumencie ustawianego atrybutem `vlink`,
- `document.cookie` — ustawia lub odczytuje `cookie` dla dokumentu.

Do odwołania się do elementu strony służą metody `getElementById()` oraz `getElementByTagName()`. Metoda `getElementById()` używana jest, gdy element, do którego się odwołujemy, ma atrybut `id`, natomiast metodę `getElementByTagName()` wykorzystujemy do pobrania kolekcji zawierającej elementy danego typu.

Przykład 3.77

```
<p id="tekst1">Skrypty języka JavaScript</p>
<p>Dokument HTML</p>
<h2>Model dokumentu DOM</h2>
```

Odwołanie w skrypcie do tak zdefiniowanych elementów może mieć postać:

- `document.getElementById("tekst1")` — odwołanie do akapitu z atrybutem `id="tekst1"`,
- `document.getElementByTagName("p")` — odwołanie do kolekcji akapitów,
- `document.getElementByTagName("p")[0]` — odwołanie do pierwszego akapitu w kolekcji.

Gdy używa się metody `getElementById()`, należy pamiętać, że jest to metoda obiektu `document`, dlatego dostęp do niej jest możliwy tylko za pomocą tego obiektu. Odwołanie do elementu strony będzie możliwe tylko wtedy, gdy temu elementowi zostanie nadany atrybut `id`.

Przykład 3.78

```
<!DOCTYPE html>
<html>
<head>
<title>Przyciski</title>
<meta charset="UTF-8">
</head>
```

```
<body>
<input type="button" id="klik" value="Kliknij!">
<script>
var b = document.getElementById("klik");
document.write("<br>Tekst z przycisku: " + b.value);
</script>
</body>
</html>
```

W podanym przykładzie za pomocą metody `getElementById()` została pobrana i zapisana w zmiennej `b` wartość elementu o `id="klik"`. Następnie za pomocą metody `write()` wartość zmiennej została wyświetlona.

Możliwość odwołania się do elementu strony jest wykorzystywana do zmiany jej zawartości. Zawartość elementu strony można odczytać i zmienić, używając właściwości `innerHTML`. Właściwość tę ma każdy element strony internetowej. Określa ona wartość przypisaną elementowi.

Przykład 3.79

```
<!DOCTYPE html>
<html>
<head>
<title>Tekst</title>
<meta charset="UTF-8">
</head>
<body>
<h2>Zmień tekst</h2>
<script>
function zmien_tekst()
{
  document.getElementById('blok').innerHTML = 'Jesień mimozami się
zaczyna';
}
</script>
<p>Pory roku: <b id="blok">Lato, lato, lato czeka...</b></p>
<input type="button" onclick="zmien_tekst()" value="Zmień tekst"/>
</body>
</html>
```

Rysunek 3.11. Możliwość zmiany tekstu wyświetlanego na stronie

Po kliknięciu przycisku nastąpi zmiana wyświetlanego tekstu (rysunek 3.11).

Ćwiczenie 3.16

W przykładzie 3.69 zostało zdefiniowane wyświetlanie daty i czasu. Zmiana czasu następowała po odświeżeniu strony. Zmodyfikuj powstały kod, tak aby czas wyświetlany na stronie był zawsze aktualny.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Mój zegar</title>
<meta charset="UTF-8">
</head>
<body>
<h2>Bieżący czas</h2>
<div id="zegar"></div>

<script>
var timerID = null;
var timerRunning = false;
function stopclock() {
  if (timerRunning) {
    clearTimeout(timerID);
  }
  timerRunning = false;
}

function startclock() {
  stopclock();
  showtime();
}

function showtime() {
  var data_n = new Date();
  var godz = data_n.getHours();
  var min = data_n.getMinutes();
  var sek = data_n.getSeconds();
```

```
    var czas = "" + godz;
    czas += ((min < 10) ? ":0" : ":") + min;
    czas += ((sek < 10) ? ":0" : ":") + sek;

    document.getElementById('zegar').innerHTML = czas;
    timerID = setTimeout("showtime()", 1000);
    timerRunning = true;
}
</script>
<script>
startclock();
</script>
</body>
</html>
```

Metoda `getElementById()` odwołuje się do wcześniej zdefiniowanego identyfikatora `zegar`. W przykładzie za pomocą właściwości `innerHTML` elementowi o identyfikatorze `zegar` przypisana została wartość zmiennej `czas`, która zawiera bieżący czas. Operator warunkowy `((min < 10) ? ":0" : ":") + min;))` ma trzy argumenty: `test`, po którym występuje znak `?` oraz `wyrażenie1` i `wyrażenie2` oddzielone znakiem `:`. Jeżeli wartość `test` to `true`, to wynikiem jest `wyrażenie1`, w przeciwnym razie wynikiem jest `wyrażenie2`. Wynik interpretacji kodu został pokazany na rysunku 3.12.

Rysunek 3.12. Wyświetlanie na stronie aktualnego czasu

Metodą obiektu `document` jest również metoda `document.write()`, która wyświetla na stronie internetowej w oknie dokumentu podany tekst.

Zadanie 3.4

Zmodyfikuj kod pokazany w przykładzie 3.79 w ten sposób, aby kolejne kliknięcia przycisku *Zmień tekst* powodowały wyświetlanie następnych tekstów, na przykład „Hu, hu, ha, nasza zima zła" i „Wiosna, wiosna wonna i radosna". Po wyświetleniu podanych tekstów powinien nastąpić powrót do wyświetlania tekstu pierwszego.

Zadanie 3.5

Wykorzystując język HTML, arkusze CSS oraz skrypty z poprzednich przykładów, utwórz kod, który na stronie internetowej wyświetli bieżący czas w postaci zegara cyfrowego.

Zadanie 3.6

Utwórz i dodaj do dokumentu HTML skrypt, który wyświetli datę ostatniej modyfikacji tego dokumentu.

Obiekt history

Drugim obiektem potomnym w stosunku do obiektu `window` jest obiekt `history`. Zawiera on informację o stronach odwiedzanych w bieżącej sesji. Ma zdefiniowane metody, które pozwalają na przejście do wcześniej odwiedzanych stron.

- `history.go()` — otwiera określony adres URL z listy historii; w nawiasach należy podać liczbę dodatnią lub ujemną, określającą, o ile do przodu lub do tyłu należy przemieścić się, aby otworzyć określony adres, na przykład `history.go(3)`,
- `history.back()` — otwiera poprzedni adres URL z listy historii,
- `history.forward()` — otwiera następny adres URL z listy historii, jeżeli taki istnieje.

Obiekt `history` ma właściwość `history.length`, która zawiera informację o długości listy historii.

Ćwiczenie 3.17

Wykorzystaj metody `back()` i `forward()`, do utworzenia skryptu, który wyświetli na stronie przyciski *Wstecz* oraz *Dalej*, umożliwiające poruszanie się w przeglądarce po odwiedzanych stronach, jak pokazano na rysunku 3.13.

Rysunek 3.13. Przyciski Wstecz i Dalej z podpiętymi metodami back() i forward()

Rozwiązanie

```
<!DOCTYPE html>
<head>
<title>Przyciski</title>
<meta charset="UTF-8">
</head>
<body>
<h1>Rewolucja przemysłowa 4.0</h1><br>
<input type="button" onclick="history.back()" value="Wstecz">
<input type="button" onclick="history.forward()" value="Dalej">
</body>
</html>
```

Ćwiczenie 3.18

Modyfikując ćwiczenie 3.17, dodaj za pomocą znaczników HTML kod umożliwiający otwieranie kilku kolejnych stron internetowych. Wykorzystaj przyciski *Wstecz* i *Dalej* do poruszania się między otwartymi stronami.

Obiekt location

Trzecim obiektem potomnym w stosunku do obiektu `window` jest obiekt `location`. Zawiera on informację o bieżącym adresie dokumentu otwartego w oknie. Za pomocą właściwości tego obiektu można uzyskać pełną informację o adresie URL, a także dostęp do jego fragmentów.

- `location.href` — zawiera cały adres URL,
- `location.protocol` — zawiera protokół,
- `location.hostname` — zawiera nazwę hosta,
- `location.port` — zawiera numer portu,
- `location.pathname` — zawiera nazwę pliku ze ścieżką,
- `location.search` — zawiera zapytanie, jeżeli znajduje się ono w adresie,
- `location.hash` — zawiera nazwę kotwicy, jeżeli kotwica występuje w adresie.

Ogólna postać lokalizatora URL to:

```
protocol://host:port/path#fragment?query
```

Właściwość `protocol` to łańcuch znakowy określający protokół, zgodnie z którym należy nawiązać łączność z podanym serwerem (na przykład *http*, *ftp*, *file*), `host` to łańcuch znakowy określający nazwę serwera z bieżącego adresu, `port` to łańcuch znakowy odpowiadający portowi, przez który należy połączyć się z serwerem, `path` to łańcuch znakowy określający ścieżkę dostępu do dokumentu na serwerze, `fragment` to łańcuch znakowy odpowiadający nazwie zakotwiczenia (przypisanie tu jakiejś wartości spowoduje przewinięcie dokumentu do wskazanego punktu), `query` to człon lokalizatora URL.

Aby zmienić adres strony wyświetlanej w oknie, wystarczy zmienić lokalizator URL.

Przykład 3.80

```
window.location.href = "http://www.helion.pl"

window.location = "https://www.google.pl/search?q=warszawa+lotnisko"
```

UWAGA

Właściwość `location.href` zawiera ten sam adres co właściwość `document.URL`. Jednak właściwości `document.URL` nie można modyfikować. W celu otwarcia nowej strony należy posługiwać się właściwością `location.href`.

Obiekt `location` ma dwie metody:

- `location.reload()` — odświeża (ponownie wczytuje) bieżący dokument. Jeżeli dodany zostanie parametr `true`, odświeżanie odbędzie się niezależnie od tego, czy dokument uległ zmianie, czy nie.
- `location.replace()` — zastępuje bieżący adres URL nowym.

Zadanie 3.7

Dla dokumentu HTML utwórz skrypt, który wyświetli nazwę pliku oraz ścieżkę dostępu do bieżącej strony internetowej.

Obiekt link

Obiektem potomnym w stosunku do obiektu `document` jest obiekt `link`. Zawiera on informację o łączu do określonego adresu. Obiekty `link` są zapisane w tablicy `links`. W dokumencie może wystąpić wiele obiektów `link`. Każdy z nich jest zapisany jako oddzielny element tablicy.

Właściwość tablicy `document.links.length` określa liczbę linków na stronie.

Każdy obiekt `link` zapisany w tablicy ma listę właściwości określających adres URL. Są to właściwości takie same jak dla obiektu `location`. Można się do nich odwoływać, podając numer w tablicy i nazwę właściwości.

Przykład 3.81

```
var link1 = links[0].href;
```

Obiekt anchor

Kolejnym obiektem potomnym w stosunku do obiektu `document` jest obiekt `anchor`. Reprezentuje on kotwicę w bieżącym dokumencie.

WSKAZÓWKA

Kotwica określa zdefiniowaną lokalizację w dokumencie HTML, do której można się przenieść.

Kotwice są zapisywane w tablicy o nazwie `anchors`. Każdy jej element jest obiektem `anchor`.

Właściwość tablicy `document.anchors.length` określa liczbę elementów kotwicy na stronie.

Obiekt form

Obiektem potomnym w stosunku do obiektu `document` jest również obiekt `form`. Zawiera on informacje dotyczące formularzy występujących w dokumencie HTML.

Obiekty `form` są zapisane w tablicy `forms`. Ponieważ w dokumencie może wystąpić wiele formularzy, każdy z nich jest zapisany jako oddzielny element tablicy.

Do wybranego formularza można odwoływać się przez:

indeks, wpisując w kodzie polecenie:

```
document.forms[0]
```

lub przez nazwę, wpisując w kodzie polecenie:

```
document.forms['Form1']
```

Inną metodą odwołania się do formularza jest wykorzystanie metody `getElement-ById()`, na przykład:

```
document.getElementById('form1')
```

Przykład 3.82

```
<body>
<form id="form1" name="Form1" action=" " method="post">
...
</form>
<script>
document.forms[0];
// lub
document.forms['Form1'];
// lub
document.getElementById('form1');
...
</script>
</body>
```

Jeżeli został zastosowany atrybut `name` (jak w przykładzie 3.82), to do formularza można odwołać się również w ten sposób: `document.Form1`.

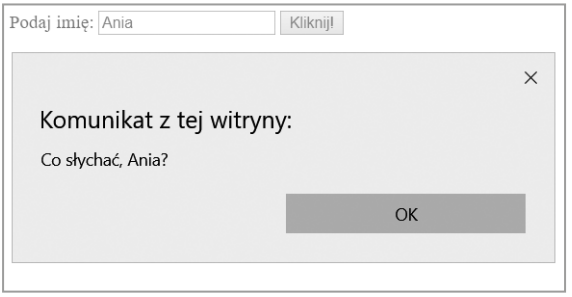
Każdy element formularza jest obiektem, więc ma właściwości. Jedną z nich jest właściwość `value`, która przechowuje bieżącą wartość elementu.

Przykład 3.83

```
<!DOCTYPE html>
<html>
<head>
<title>Co słychać</title>
```

```
<meta charset="UTF-8">
</head>
<body>
<form id="form1" name="Form1" action=" " method="post">
Podaj imię: <input type="text" name="imie" id="imie"/>
<button onclick="witaj()">Kliknij!</button>
</form>
<script type="text/javascript">
function witaj() {
  var imie = document.forms['Form1'].imie.value;
  alert('Co słychać, ' + imie + '?');
}
</script>
</body>
</html>
```

Podany kod definiuje formularz z polem `imie` i przyciskiem *Kliknij!*. Gdy dla przycisku wystąpi zdarzenie `onclick` (kliknięcie przycisku), zostanie uruchomiona funkcja `witaj()`, która pobierze wartość elementu `imie` (`var imie = document.forms['Form1'].imie.value;`) i wyświetli ją w oknie z komunikatem (rysunek 3.14).



Rysunek 3.14.
Pobrane z formularza imię zostało wyświetlone w oknie z komunikatem

Elementy formularza tworzą tablicę. Dostęp do nich jest możliwy przez odwołanie się do kolejnych elementów tej tablicy (`elements[i]`). Podobnie jak do całego formularza, do jego elementów można odwoływać się przez indeks lub przez nazwę:

```
document.forms['Form1'].elements[0]
document.forms['Form1'].elements['imie']
document.forms['Form1'].imie
```

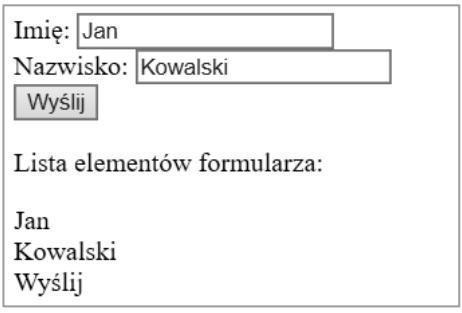
albo za pomocą metody `getElementById()`:

```
document.getElementById('imie').value
```

Przykład 3.84

```
<!DOCTYPE html>
<html>
<head>
<title>Lista formularza</title>
<meta charset="UTF-8">
</head>
<body>
<form id="form1" action=" " method="post">
Imię: <input type="text" name="imie" value="Jan"/><br>
Nazwisko: <input type="text" name="nazwisko" value="Kowalski" /><br>
<input type="submit" value="Wyślij">
</form>
<p>Lista elementów formularza:</p>
<script>
var x = document.getElementById("form1");
for (var i = 0; i < x.length; i++) {
  document.write(x.elements[i].value);
  document.write("<br>");
}
</script>
</body>
</html>
```

Wynikiem wykonania kodu będzie wyświetlenie formularza z polami *Imię:* i *Nazwisko:* oraz przycisku *Wyślij* wraz z listą elementów formularza (rysunek 3.15).



Rysunek 3.15. Wyświetlenie na stronie formularza wraz z listą jego elementów

3.9. Obsługa zdarzeń

Tworzone w języku JavaScript skrypty mogą służyć do obsługi zdarzeń (ang. *event handler*). Są to tak zwane procedury obsługi zdarzeń. Skrypty takie definiują zachowanie się przeglądarki w przypadku wystąpienia określonego zdarzenia. Większość zdarzeń wywoływana jest przez działania użytkownika (na przykład kliknięcie przyciskiem myszy). Gdy wystąpi takie zdarzenie, przeglądarka przechodzi do wykonania skryptu związanego z zaistniałym zdarzeniem. Występują również zdarzenia, które nie są inicjowane przez użytkownika (na przykład zakończenie wczytywania strony).

Każdy skrypt opisujący zdarzenie skojarzony jest z określonym obiektem strony internetowej. Żeby takie zdarzenie mogło zostać obsłużone, element HTML musi być dostępny dla skryptu. Jedną z metod, by to zapewnić, jest wstawianie skryptu na końcu strony przed znacznikiem `</body>`. Inną metodą jest dodanie atrybutu `defer` do znacznika `<script>`, co spowoduje, że gdy przeglądarka napotka skrypt, będzie on wczytywany w tle razem ze stroną internetową, a uruchomiony zostanie po załadowaniu całego dokumentu. Dobrym rozwiązaniem jest użycie zdarzenia `DOMContentLoaded` — spowoduje to, że kod, który odwołuje się do elementów w kodzie HTML, zostanie wykonany po ich wczytaniu.

Przykład 3.85

```
document.addEventListener("DOMContentLoaded", function() {
  console.log("DOM został wczytany");
});
```

Metoda `addEventListener()` przypisuje do elementu zdarzenie opisane daną funkcją.

Przykłady zdarzeń to:

- kliknięcie przez użytkownika myszą,
- przesunięcie myszy nad element,
- wczytanie strony internetowej,
- wczytanie obrazu,
- zmiana zawartości pola wprowadzania,
- naciśnięcie klawisza.

W kodzie HTML w znaczniku opisującym obiekt musi znaleźć się specyfikacja skryptu opisującego zdarzenie.

Przykładowo zdarzenie `onmouseover` zachodzi wówczas, gdy wskaźnik myszy pojawi się nad obiektem (na przykład przyciskiem). Zdarzenie to może zostać opisane w następujący sposób w kodzie HTML:

```
<img src="przycisk.gif" onmouseover="zmien();">
```

Obsługa zdarzenia została opisana jako wartość atrybutu (`zmien`), a sam atrybut otrzymał nazwę, która jest nazwą zdarzenia (`onmouseover`). W tym wypadku obsługa zdarzenia została zaprojektowana w postaci funkcji, czyli po wywołaniu funkcji zdarzenie zostanie obsłużone.

Nazwy zdarzeń w języku JavaScript zwykle zaczynają się od słowa `on`. Kliknięcie będzie mieć nazwę `onclick`, najechanie myszą `onmouseover` itd.

Jest kilka sposobów obsługi zdarzeń. Aby obsłużyć zdarzenie, nie zawsze trzeba tworzyć funkcję.

Obsługa zdarzeń w kodzie HTML

Jeżeli kod, który ma zostać wywołany, jest pojedynczą instrukcją, można wpisać skrypt bezpośrednio w znaczniku.

Przykład 3.86

```
<!DOCTYPE html>
<html>
<head>
<title>Okno ALERT</title>
<meta charset="UTF-8">
</head>
<body>
<input type="button" value="Pokaż" onclick="alert('Witaj!');"/>
</body>
</html>
```

Po kliknięciu przycisku wyświetli się okno z komunikatem `Witaj!`.

Nie jest to najlepsza metoda obsługi zdarzenia, ponieważ miesza się w niej język HTML z językiem JavaScript.

Obsługa zdarzenia jako właściwości obiektu

W tej metodzie musi zostać zdefiniowana funkcja obsługująca zdarzenie. Następnie funkcję należy przypisać do danego elementu jako jego właściwość. Funkcja powinna zostać przypisana po nazwie bez nawiasów, ponieważ nie zostanie ona wywołana.

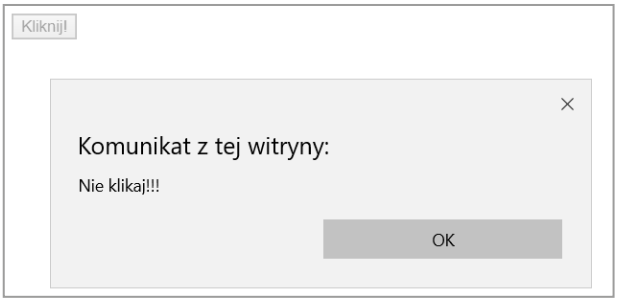
Przykład 3.87

```
<!DOCTYPE html>
<html>
<head>
<title>Klik</title>
```

```
<meta charset="UTF-8">
</head>
<body>
<input type="button" id="klik" value="Kliknij!">
<script>
function pisz() {
  alert('Nie klikaj!!!');
}
document.getElementById("klik").onclick = pisz;
</script>
</body>
</html>
```

W podanym przykładzie został utworzony przycisk `<input type="button" id="klik" value="Kliknij!">` z identyfikatorem `klik` oraz została zdefiniowana funkcja `pisz()`. Następnie funkcja `pisz` została przypisana do elementu `klik` jako jego właściwość `document.getElementById("klik").onclick = pisz;`.

Wynik interpretacji kodu został pokazany na rysunku 3.16.



Rysunek 3.16. Wynik obsługi zdarzenia onclick po kliknięciu przycisku Kliknij!

3.9.1. Zdarzenia myszy

Mysz jest podstawowym narzędziem wykorzystywanym do poruszania się po stronach internetowych, dlatego obsługa zdarzeń związanych z myszą jest jedną z najczęściej stosowanych funkcjonalności.

Do zdarzeń wywołanych działaniem myszy należą:

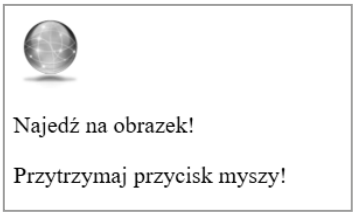
- `onclick` — występuje po kliknięciu myszą,
- `ondblclick` — występuje po dwukrotnym kliknięciu myszą,
- `onmousedown` — występuje, gdy przycisk myszy zostanie wciśnięty,

- `onmouseup` — występuje, gdy przycisk myszy zostanie zwolniony,
- `onmouseover` — występuje, gdy kursor myszy zostanie umieszczony na elemencie,
- `onmousemove` — występuje, gdy kursor myszy zostanie przesunięty wewnątrz elementu,
- `onmouseout` — występuje, gdy kursor myszy zostanie przesunięty poza element.

Przykład 3.88

```
<!DOCTYPE html>
<html>
<head>
<title>Zdarzenie myszy</title>
<meta charset="UTF-8">
</head>
<body>
<a href="http://www.helion.pl" onmouseover='zmiana.src="Ikona3.png"'
onmouseout='zmiana.src="Ikona1.png"' onmousedown='zmiana.src="Ikona4.png"'>
<img src="Ikona1.png" name="zmiana" border="0" width="50" height="50"></a>
<p>Najedź na obrazek!</p>
<p>Przytrzymaj przycisk myszy!</p>

</body>
</html>
```

W przykładzie zostały wykorzystane zdarzenia myszy `onmouseover`, `onmouseout` i `onmousedown`. W efekcie po ustawieniu myszy na pierwszym rysunku ulegnie on zmianie. Wciśnięcie przycisku myszy spowoduje kolejną zmianę rysunku, a po przesunięciu myszy poza rysunek zostanie przywrócona pierwsza grafika. Kliknięcie rysunku spowoduje przejście do strony *helion.pl* (rysunek 3.17).



Rysunek 3.17.
Obsługa zdarzeń myszy

Przykład 3.89

```
<!DOCTYPE html>
<html>
<head>
<script>
function ik2() {
  document.getElementById('ikona').src = "Ikona2.png";
}
```

```
function ik1() {
  document.getElementById('ikona').src = "Ikona1.png";
}
</script>
</head>
<body>
<img id="ikona" onmousedown="ik2()" onmouseup="ik1()" src="Ikona1.png"
width="100" height="100" />
<p>Kliknij i przytrzymaj!</p>
</body>
</html>
```

W przykładzie zostały wykorzystane zdarzenia myszy `onmousedown` i `onmouseup`. W celu odwołania się do elementu `id="ikona"` posłużono się metodą `getElementById()`. Po wciśnięciu przycisku myszy nastąpi zmiana grafiki. Po jego zwolnieniu nastąpi powrót do poprzedniej grafiki.

3.9.2. Zdarzenia klawiatury

Zdarzenia związane z klawiaturą są rzadko stosowane. Wykorzystują je tylko aplikacje typu edytor tekstu lub czytnik poczty. Są to trzy zdarzenia:

- `onkeypress` — klawisz został naciśnięty i zwolniony,
- `onkeydown` — klawisz został wciśnięty, ale nie został zwolniony,
- `onkeyup` — klawisz został zwolniony.

3.9.3. Zdarzenia formularza

Z formularzem związane są dwa zdarzenia:

- `onsubmit` — zdarzenie jest generowane, gdy użytkownik wysyła formularz (po kliknięciu przycisku typu `submit`). Zdarzenie to może zostać wykorzystane do sprawdzenia, czy formularz został poprawnie wypełniony.
- `onreset` — zdarzenie jest generowane, gdy formularz jest czyszczony z zawartości (po kliknięciu przycisku typu `reset`).

3.9.4. Zdarzenia elementów formularza

Oprócz zdarzeń generowanych przez formularz mogą wystąpić zdarzenia generowane przez elementy formularza. Są to:

- `onfocus` — element formularza został zaznaczony, czyli wpisywane dane będą trafiały do tego elementu,

- `onblur` — element stracił zaznaczenie,
- `onselect` — element został wybrany (wybrany element nie zawsze otrzymuje zaznaczenie),
- `onchange` — zawartość elementu formularza uległa zmianie.

3.9.5. Zdarzenia dokumentu

Do zdarzeń związanych z dokumentem należą:

- `onload` — strona została załadowana,
- `onunload` — strona jest zamykana.

3.10. Wykorzystanie skryptów na stronie internetowej

3.10.1. Kolejność wykonywania skryptów

W kodzie HTML może być umieszczonych wiele różnych skryptów. Mogą to być skrypty zapisane między znacznikami `<script>` i `</script>`, skrypty zapisane w zewnętrznych plikach lub procedury obsługi zdarzeń. Skrypty będą wykonywane w ściśle określonej kolejności.

Skrypty `<script>`, znajdujące się w sekcji `<head>`, niezależnie od tego, czy są wbudowane w kod, czy też są importowane z zewnętrznych plików, wykonywane są w pierwszej kolejności. Skrypty te nie mogą tworzyć zawartości wyświetlanej na stronie, dlatego najczęściej służą do definiowania funkcji wykorzystywanych w innych skryptach.

Skrypty typu `<script>` znajdujące się w sekcji `<body>` wykonywane są w drugiej kolejności. Są one wykonywane podczas wczytywania i wyświetlania strony, w kolejności takiej, w jakiej zostały zdefiniowane w sekcji `<body>`.

Skrypty, które służą do obsługi zdarzeń (procedury zdarzeniowe), są uruchamiane w chwili wystąpienia zdarzenia, na przykład skrypt obsługujący zdarzenie `onLoad` wykonywany jest w momencie rozpoczęcia wczytywania strony. Ponieważ sekcja `<head>` jest wczytywana przed zaistnieniem jakiegokolwiek zdarzenia, funkcje, które zostały zdefiniowane w skryptach znajdujących się w tej sekcji, mogą być używane w procedurach zdarzeniowych.

3.10.2. Animowanie tekstu

Jednym z ciekawszych zastosowań języka JavaScript jest możliwość tworzenia na stronie internetowej animowanej grafiki. Mogą to być: obraz zmieniający się w chwili najechania na niego myszą, zmieniające się automatycznie obrazki, animowany baner, pokaz slajdów czy galeria zdjęć.

Baner (przesuwany tekst)

Animowanie tekstu w banerze może polegać na przemieszczaniu tekstu z początkowego położenia z prawej strony okna w kierunku jego lewej krawędzi.

Ćwiczenie 3.19

Wykonaj animację dowolnego tekstu znajdującego się w górnej części strony internetowej. Tekst powinien przemieszczać się od prawej strony okna do lewej, a gdy dotrze do lewej krawędzi, animacja ma zostać wykonana powtórnie.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Przepływający tekst</title>
<meta charset="UTF-8">
<style type="text/css">
#napis {
  position: absolute;
  background-color: #c0cfc0;
  width: 350px;
  float: left;
  font-size: 18pt;
  font-family: Arial;
}
</style>
<script>
function animacja() {
  var blok = document.getElementById('napis');
  if (parseInt(blok.style.left) < 50) {
    blok.style.left = "500px";
  } else {
    blok.style.left = (parseInt(blok.style.left) - 3) + "px";
  }
}
window.setInterval(animacja, 100);
</script>
```

```
</head>
<body>
<div id="napis" style="left: 500px;">JavaScript - język skryptowy :-)
</div>
</body>
</html>
```

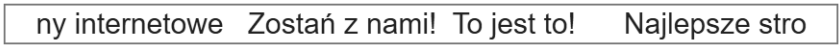
W podanym przykładzie w sekcji `<body>` został utworzony blok `div` ze zdefiniowanym identyfikatorem `"napis"`, który będzie animowany. Początkowe położenie bloku `<div>` zostało zdefiniowane z wykorzystaniem parametru `style="left: 500px;"`. Zadaniem skryptu jest przesuwanie bloku o 3 piksele co 0,1 sekundy. Funkcja `animacja()` sprawdza wartość właściwości `style.left` elementu `id="napis"` i jeżeli wynik jest mniejszy niż `50px`, element jest przenoszony do początkowej pozycji (`blok.style.left = "500px"`). Jeżeli wynik jest większy niż `50px`, to kontynuowane jest odejmowanie od aktualnej wartości `3px`. Metoda `window.setInterval()` wywołuje funkcję `animacja()` co 0,1 sekundy.

Baner (tekst pływający)

Innym sposobem animowania tekstów w banerach reklamowych jest wyświetlanie tekstu w ciągłej pętli.

Ćwiczenie 3.20

Umieść w bloku, w górnej części okna strony internetowej, długi tekst. Wykonaj jego animację, wzorując się na przepływie newsów na pasku w dolnej części ekranu telewizora. Tekst powinien przewijać się litera po literze w utworzonym bloku (rysunek 3.18).



Rysunek 3.18. Pasek z przewijanym tekstem

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Baner reklamowy</title>
<meta charset="UTF-8">
<style type="text/css">
#tekst {
  font-size: 18pt;
  font-family: Arial;
  text-align: center;
}
```

```
</style>
<script>
czas = 200;
znak_p = 1;
function przewin() {
  window.setTimeout(przewin, czas);
  var nap = document.napis.text.value;
  document.napis.text.value = nap.substring(znak_p)
+ nap.substring(0, znak_p);
}
przewin()
</script>
</head>
<body>
<form name="napis">
<input id="tekst" name="text" size=60 value="   Najlepsze strony
internetowe.   Zostań z nami!  To jest to!   ">
</form>
</body>
</html>
```

W podanym przykładzie została wykorzystana metoda `window.setTimeout()`, która wywołuje funkcję po określonej liczbie milisekund. Metoda ta ma dwa argumenty. Pierwszy z nich to nazwa uruchamianej funkcji, a drugi to parametr określający, po jakim czasie powinna zostać uruchomiona funkcja.

Metoda `substring()` zwraca fragment tekstu z łańcucha. Jej parametry określają położenie początku i końca tego tekstu. Pierwszy parametr musi wystąpić, drugi występuje opcjonalnie. Jeżeli ten ostatni nie zostanie podany, koniec fragmentu tekstu jest równoznaczny z końcem łańcucha.

3.10.3. Animowanie grafiki

Zmiana grafiki

Prostym sposobem animowania grafiki jest zmiana obrazka po ustawieniu na nim kursora myszy.

Ćwiczenie 3.21

Przygotuj dwa dowolne pliki ze zdjęciami. Umieść na stronie internetowej pierwsze zdjęcie. Napisz skrypt, który spowoduje, że po wskazaniu tego zdjęcia myszą zostanie ono zamienione na zdjęcie drugie. Po odsunięciu myszy na stronie ponownie powinno zostać wyświetlone zdjęcie pierwsze.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Zmiana obrazka</title>
</head>
<body>
<p>
<img src="obraz1.png" alt="" id="widok">
</p>
<script>
var obraz = document.getElementById('widok');
obraz.onmouseover = function() {
  this.src = 'obraz2.png';
}
obraz.onmouseout = function() {
  this.src = 'obraz1.png';
}
</script>
</body>
</html>
```

W podanym rozwiązaniu zostały wykorzystane dwa obrazki zapisane w plikach *obraz1.png* oraz *obraz2.png*. W wyniku wykonania skryptu po najechaniu kursorem myszy na pierwszy obrazek nastąpi zmiana atrybutu `src` na drugi obrazek. Po odsunięciu kursora myszy atrybut `src` ponownie przyjmie wartość pierwszego obrazka.

W wyniku wykonania polecenia `var obraz = document.getElementById('widok');` została utworzona zmienna `obraz` zawierająca odnośnik do elementu o `id="widok"`. Kolejnym działaniem jest obsługa zdarzenia `onmouseover` dla elementu `obraz`. Po wystąpieniu tego zdarzenia nastąpi zmiana atrybutu `src` obiektu (`this.src = 'obraz2.png'`). Po usunięciu kursora znad obrazka nastąpi ponowna zmiana atrybutu `src` (`this.src = 'obraz1.png'`). Słowo kluczowe `this` wskazuje na obiekt, który wywołał funkcję.

Animowany baner (slider)

Innym rodzajem animowanej grafiki jest baner ze zmieniającymi się automatycznie obrazkami.

Ćwiczenie 3.22

Utwórz stronę internetową z miejscem zarezerwowanym na slider. Przygotuj sześć zdjęć o takich samych rozmiarach, które będzie można umieścić w górnej części strony. Napisz skrypt, który automatycznie na przykład co 3000 milisekund będzie zmieniał zdjęcia wyświetlane w sliderze.

Rozwiązanie

```
<!DOCTYPE html>
<html>
<head>
<title>Animowany baner</title>
<style type="text/css">
#baner {
  width: 900px;
  height: 150px;
}
</style>
<script>
window.onload = zmiana;
var nr = 0;
function zmiana() {
  var obrazy = ['z1.jpg', 'z2.jpg', 'z3.jpg', 'z4.jpg', 'z5.jpg',
'z6.jpg']; // dodawanie obrazków do tablicy
  nr++;
  if (nr == obrazy.length) {
    nr = 0;
  }
  document.getElementById('baner').src = obrazy[nr];
  setTimeout(zmiana, 3000);
}
</script>
</head>
<body>
<div>
<img src="z1.jpg" id="baner" alt="Reklama">
</div>
</body>
</html>
```

W podanym rozwiązaniu zastosowano tablicę do przechowywania nazw obrazów. Wielkość tablicy może być zmieniana w zależności od liczby wyświetlanych obrazów. Ich rozmiar został zdefiniowany za pomocą stylów.

Ponieważ próba odwołania się do elementów, które nie zostały jeszcze wczytane, mogłaby spowodować wystąpienie błędów i nieprawidłowe działanie skryptu, pierwsze polecenie skryptu `window.onload = zmiana;` obsługuje zdarzenie związane z zakończeniem wczytywania strony i wywołuje funkcję `zmiana()`.

Pojawianie się kolejnych zdjęć jest obsługiwane przy użyciu licznika czasu. Zmienna `nr` to licznik wyświetlanych obrazów. Jej początkowa wartość została ustawiona na `0`. Funkcja `zmiana()` zawiera definicję tablicy `obrazy`, która przechowuje nazwy plików z wyświetlanymi obrazami. Za pomocą kodu `if (nr == obrazy.length)` sprawdza się, czy wartość licznika obrazów (`nr`) jest równa liczbie elementów tablicy `obrazy`. Jeżeli tak, to licznik jest zerowany.

Obrazowi wyświetlanemu na stronie został przypisany identyfikator `id="baner"`. Kod `document.getElementById('baner').src = obrazy[nr]` pobiera adres obrazu z tablicy `obrazy` z pozycji określonej zmienną `nr`. Natomiast metoda `setTimeout()` wywołuje funkcję `zmiana()` co określoną liczbę milisekund.

3.11. Walidacja formularzy

Walidacja danych polega na sprawdzeniu, czy wszystkie pola formularza zostały wypełnione oraz czy wprowadzone wartości mają odpowiednią postać.

Jednym ze sposobów kontrolowania danych z formularzy jest stosowanie skryptów działających po stronie serwera (na przykład PHP). Innym jest kontrolowanie formularzy za pomocą skryptów działających po stronie przeglądarki.

Sprawdzanie poprawności wypełnienia formularza po stronie klienta ma kilka zalet. Po pierwsze, informacja o błędzie w wypełnieniu formularza jest zwracana natychmiast, bez konieczności oczekiwania na odpowiedź ze strony serwera. Po drugie, następuje zmniejszenie obciążenia serwera. W praktyce sprawdzanie poprawności wypełnienia formularza za pomocą skryptów JavaScript nie zwalnia z konieczności sprawdzania poprawności danych po stronie serwera (walidacja danych po stronie serwera zapewnia bezpieczne działanie całej aplikacji).

3.11.1. Sprawdzanie wypełnienia pól formularza

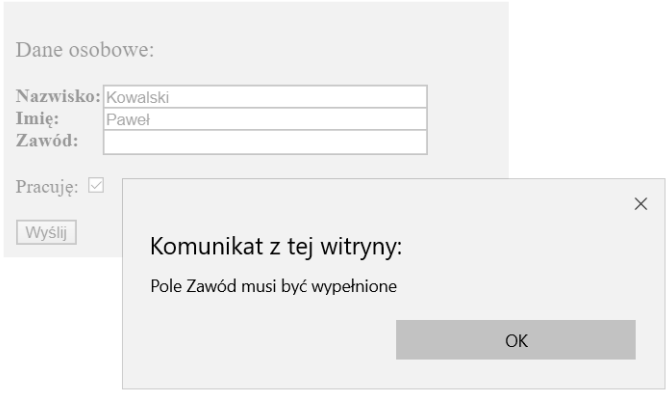
Podstawowym działaniem wykonywanym podczas walidacji formularza jest sprawdzenie, czy pola, które nie powinny pozostać puste, zostały przez użytkownika wypełnione. Jeżeli te pola są puste, formularz nie zostanie wysłany.

Przykład 3.90

```
<!DOCTYPE html>
<html>
<head>
<title>Sprawdź_formularz</title>
<meta charset="UTF-8">
<script>
function sprawdz(form) {
  if (form.nazw.value == '') {
    alert('Pole Nazwisko musi być wypełnione');
    form.nazw.focus();
    return false;
  }
  if (form.imie.value == '') {
    alert('Pole Imię musi być wypełnione');
    form.imie.focus();
    return false;
  }
  if (form.zawod.value == '') {
    alert('Pole Zawód musi być wypełnione');
    form.zawod.focus();
    return false;
  }
  return true;
}
</script>

<style type="text/css">
div {
  width: 400px;
  background-color: #dfdfdf;
  padding: 10px;
}
.pole {
  position:absolute;
  left:90px;
```

```
}
#ak1 {
  font-size: 14pt;
}
</style>
</head>
<body>
<div>
<form enctype="text/plain" action=" " method="post"
onsubmit="return sprawdz(this);">
<p id="ak1">Dane osobowe:</p>
<b>Nazwisko:</b>
<input class="pole" name="nazw" value="" size="40"><br>
<b>Imię:</b>
<input class="pole" name="imie" value="" size="40"><br>
<b>Zawód:</b>
<input class="pole" name="zawod" value="" size="40"><br><br>
<label> Pracuję: <input type="checkbox" name="opcje" maxlength="1">
</label><br><br>
<input type="submit" value="Wyślij"></form>
</div>
</body>
</html>
```

W podanym przykładzie funkcja `sprawdz()` zwraca wartość `true` lub `false`. Gdy zwróci `false` (pole nie zostało wypełnione), formularz nie zostanie wysłany (rysunek 3.19).



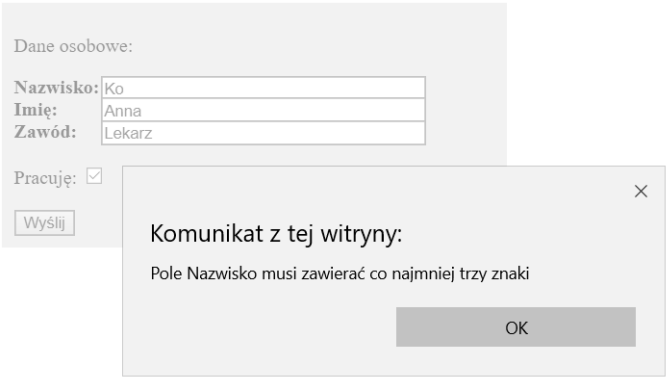
Rysunek 3.19. Walidacja formularza. Formularz nie zostanie wysłany

Inny sposób zdefiniowania funkcji `sprawdz()` z wykorzystaniem metody `getElementById()` został pokazany w kolejnym przykładzie (przykład 3.91). Za pomocą funkcji `sprawdz()` można nie tylko zweryfikować to, czy pole zostało wypełnione, ale również sprawdzić długość tekstu wprowadzonego do poszczególnych pól.

Przykład 3.91

```
function sprawdz(form) {
  if (document.getElementById('nazw').value.length < 3) {
    alert('Pole Nazwisko musi zawierać co najmniej trzy znaki');
    form.nazw.focus();
    return false;
  }
  if (document.getElementById('imie').value.length < 2) {
    alert('Pole Imię musi zawierać co najmniej dwa znaki');
    form.imie.focus();
    return false;
  }
  if (document.getElementById('zawod').value == '') {
    alert('Pole Zawód musi być wypełnione');
    form.zawod.focus();
    return false;
  }
  return true;
}
```

Wynik wykonania podanej w przykładzie funkcji został pokazany na rysunku 3.20.



Rysunek 3.20. Formularz nie zostanie wysłany, ponieważ ciąg znaków wpisany w polu Nazwisko jest za krótki

3.11.2. Sprawdzenie pól formularza po wypełnieniu

Sprawdzenie pól formularza po wypełnieniu stosuje się w celu poinformowania użytkownika o niepoprawnym wypełnieniu formularza. Najprostszym przypadkiem jest sprawdzanie, czy w polu pojawił się wymagany tekst.

Przykład 3.92

```
<!DOCTYPE html>
<html>
<head>
<title>Sprawdzenie po wpisaniu</title>
<meta charset="UTF-8">
<style type="text/css">
.pole {
  position: absolute;
  left: 90px;
}
</style>
<script>
function adres() {
  if (document.getElementById('mail').value != 'e-mail') {
    alert('Wpisz tekst: "e-mail"');
    return false;
  }
  return true;
}
</script>
</head>
<body bgcolor="#dfdfdf">
<form action="" method="post">
<b>Nazwisko:</b>
<input class="pole" name="nazw" id="nazw" value="" size="40"><br>
<b>e-mail:</b>
<input class="pole" name="mail" id="mail" value="" size="40"
onblur="return adres()"><br>
<b>Zawód:</b>
<input class="pole" name="zawod" id="zawod" value="" size="40"><br><br>
<input type="submit" value="Wyślij">   
</form>
</body>
</html>
```

W podanym przykładzie zastosowano obsługę zdarzenia `onblur` (utrata fokusu) dla elementu `e-mail`. Po jego opuszczeniu zostaje uruchomiona funkcja `adres()`, która sprawdza wartość przypisaną do elementu. Jeżeli jest niepoprawna, pojawi się komunikat `Wpisz tekst: "e-mail"`.

3.11.3. Wyrażenia regularne

Wyrażenia regularne to wzorce opisujące łańcuch symboli. Przy ich użyciu można sprawdzać i modyfikować teksty. Wyrażenia regularne są wykorzystywane do weryfikowania, czy ciągi znaków wprowadzonych do formularza są zgodne z wymaganymi wzorcami.

Język JavaScript ma wbudowane mechanizmy obsługi wyrażeń regularnych. Za obsługę tych wyrażeń odpowiada obiekt `RegExp(wzorzec, flaga)`. Można go utworzyć na dwa sposoby:

```
var nowe_wyr = new RegExp('^[0-9]+[a-z]+$');
var nowe_wyr = /^[0-9]+[a-z]+$/;
```

Podane przykłady są równoznaczne.

Definiowany `wzorzec` składa się z symboli (znaków specjalnych), które opisują wygląd określonego fragmentu tekstu. Tabela 3.6 zawiera opis wybranych symboli.

Tabela 3.6. Niektóre symbole wzorca

Symbol	Znaczenie	Przykład	Ciągi zgodne z podanym wzorcem
`^`	początek wzorca	`^pa`	pani, pan, parasol
`$`	koniec wzorca	`as$`	las, czas, kompas
`.`	dowolny pojedynczy znak	`.an.a`	banda, fanta, janka
`[...]`	dowolny z wymienionych znaków; w nawiasach można podać kolejne znaki lub wpisać zakres	`[a-z][b-t]naln[ey]`	finalny, tonalne
`[^...]`	dowolny z niewymienionych znaków	`kro[^st]`	krowa, kroki, kropy
`|`	jeden z ciągów rozdzielonych znakiem	`pierwszy|1`	pierwszy, 1
`{3}`	dokładnie trzy poprzedzające znaki lub elementy	`[0-9]{3}`	243, 178, 629
`{3,}`	co najmniej trzy poprzedzające znaki lub elementy	`[a-k]{3,}`	abecad, gafa, haha

Symbol	Znaczenie	Przykład	Ciągi zgodne z podanym wzorcem
`{2,5}`	od dwóch do pięciu poprzedzających znaków lub elementów	`[a-m]{2,4}`	mama, bal, da
`\.`	znak kropki	`[0-9]{3}\.[0-9]{2}`	421.23, 829.45

Jako drugi parametr obiektu `RegExp()` może wystąpić flaga, ale nie musi. Flaga działa na zdefiniowanym wzorcu (tabela 3.7).

Tabela 3.7. Niektóre symbole flagi

Znak flagi	Znaczenie
`i`	nie jest uwzględniana wielkość liter
`g`	zwracane są wszystkie pasujące fragmenty

W języku JavaScript dodatkowo zostały wprowadzone specjalne klasy znaków. Ich symbole mogą być wykorzystane przy definiowaniu wyrażeń regularnych (tabela 3.8).

Tabela 3.8. Klasy znaków

Klasa znaków	Znaczenie
`\s`	znak spacji, tabulacji lub nowego wiersza
`\S`	znak nie jest spacją, znakiem tabulacji lub znakiem nowego wiersza
`\w`	każdy znak, który jest literą, cyfrą lub znakiem _
`\W`	każdy znak, który nie jest literą, cyfrą ani znakiem _
`\d`	każdy znak, który jest cyfrą
`\D`	każdy znak, który nie jest cyfrą

Wzorzec kodu pocztowego

```
var w_kod = /^[0-9]{2}-[0-9]{3}$/;
```

Wzorce definiowane w języku JavaScript zaczynają się od znaku / i na nim się kończą. Zapis `[0-9]{2}-[0-9]{3}` oznacza, że najpierw powinny wystąpić dwie cyfry. Po nich musi wystąpić znak –, a po nim muszą być trzy cyfry (we wzorcu `[0-9]{3}`).

Inny zapis wzorca do weryfikacji kodu pocztowego:

```
var w_kod = /^[\d]{2}-[\d]{3}$/;
```

lub

```
var w_kod = new RegExp('^[0-9]-{2}[0-9]{3}$');
```

Wzorzec do weryfikacji imienia i nazwiska

```
var w_nazw = /^[a-zA-Z]{2,}\s+[a-zA-Z]{2,}$/;
```

Znak ^ oznacza, że wzorzec zaczyna się od początku tekstu. Zapis `[a-zA-Z]{2,}` mówi, że ciąg powinien zawierać przynajmniej dwie litery (imię). Zapis `\s+` oznacza, że dalej powinny być spacje lub tabulatory (przynajmniej jeden). Kolejny zapis `[a-zA-Z]{2,}` mówi, że następny ciąg to znowu przynajmniej dwie litery (nazwisko). Znak `$` oznacza zakończenie wzorca wraz z końcem tekstu.

Inny sposób zapisu wzorca do weryfikacji imienia i nazwiska:

```
var w_nazw = /^[\D]{2,}\s+[\D]{2,}$/;
```

lub:

```
var w_nazw = new RegExp('[a-zA-Z]{2,}\s[a-zA-Z]{2,}');
```

Wzorzec do weryfikacji adresu e-mail

```
var w_mail = /^[0-9a-zA-Z_.-]+@[0-9a-zA-Z.-]+\.[a-zA-Z]{2,3}$/;
```

Znak ^ oznacza, że wzorzec zaczyna się z początkiem tekstu. Zapis `[0-9a-zA-Z_.-]` mówi, że nazwa konta może składać się z dowolnych znaków z zakresu cyfr, liter, znaku podkreślenia _, kropki . i myślnika -. Potem powinien wystąpić znak `@`. Po tym znaku sprawdzana jest nazwa domeny, która może składać się z dowolnych znaków z zakresu cyfr, liter oraz znaków kropki . i znaku -. Zapis `\.` oznacza, że kolejnym znakiem musi być kropka, a zapis `[a-zA-Z]{2,3}` mówi, że po kropce musi wystąpić końcowa część nazwy domeny składająca się wyłącznie z liter i jej długość musi wynosić dwa lub trzy znaki. Znak `$` oznacza, że wzorzec ma się kończyć wraz z końcem tekstu.

Gdy zna się sposoby definiowania wzorców do weryfikacji danych zapisanych w formularzu, można przystąpić do napisania kodu, który będzie sprawdzał te dane na stronie internetowej.

Przykład 3.93

```
<!DOCTYPE html>
<html>
<head>
<title>Sprawdź według wzorca</title>
<meta charset="UTF-8">
<script>
function Spr_wzorzec()
{
    var form = document.getElementById('in_form'),
    wzory = {
      'nazwisko' : /^[a-zA-Z]{2,}\s+[a-zA-Z]{2,}$/i,
```

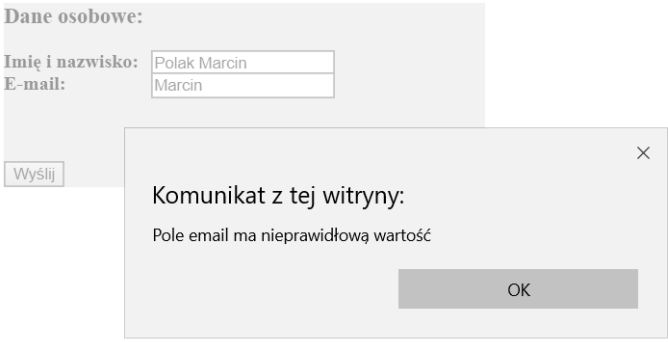
```
    'E-mail' : /^[0-9a-zA-Z_.-]+@[0-9a-zA-Z.-]+\.[a-zA-Z]{2,3}$/i,
  };
  for (var pole in wzory)
  {
    if (form[pole])
    {
      if (!wzory[pole].test(form[pole].value))
      {
         alert('Pole ' + pole + ' ma nieprawidłową wartość');
         form[pole].style.background = 'yellow';
         return false;
      }
      else
      {
         form[pole].style.background = '';
      }
    }
  }
  alert('Wszystkie pola wypełnione poprawnie!');
  return true;
}
</script>

<style type="text/css">
div {
width: 400px;
background-color: #dfdfdf;
}
p {
font-size: 14pt;
}
.pole {
    position:absolute;
    left:130px;
}
```

```
</style>


</head>
<body>
<div>
<form id="in_form" enctype="text/plain" action=""
method="post" onsubmit='return Spr_wzorzec();'>


<p>Dane osobowe:</p>
<b>Imię i nazwisko:</b>
<input class="pole" name="nazwisko" id="nazwisko" value=""><br>
<b>E-mail:</b>
<input class="pole" name="E-mail" id="E-mail" value=""><br><br>
<br><br>


<input type="submit" value="Wyślij"></form>
</div>
</body>
</html>
```

W podanym przykładzie formularz składa się z dwóch pól: *Imię i nazwisko:* oraz *E-mail:*. Po ich wypełnieniu następuje sprawdzenie za pomocą funkcji `spr_wzorzec()` poprawności wprowadzonych danych.

Funkcja `spr_wzorzec()` tworzy tablicę asocjacyjną `wzory`. W jej definicji zamiast nawiasów kwadratowych zostały zastosowane nawiasy klamrowe, a zamiast pojedynczych wartości podano pary: klucz i jego wartość (klucz jest oddzielony od wartości dwukropkiem). Kluczami są nazwy pól formularza, a wartościami obiekty wyrażeń regularnych. Dostęp do wszystkich elementów tablicy jest możliwy za pośrednictwem pętli `for (var pole in wzory)`. Jeżeli w formularzu istnieje element odpowiadający elementowi tablicy, to sprawdzane jest, czy jego aktualna wartość jest zgodna z wartością zapisanego w tablicy wyrażenia regularnego: `if (!wzory[pole].test(form[pole].value))`. Do tego celu została wykorzystana metoda `test()`, która sprawdza, czy wartość w polu formularza (`form[pole].value`) jest zgodna ze zdefiniowanym w tablicy wzorcem (`wzory[pole]`). Jeżeli nie, to wyświetli się komunikat o błędzie w danych i nastąpi powrót do wypełniania formularza. Gdy wszystkie pola zostaną wypełnione prawidłowo, wyświetli się komunikat o poprawności danych, a funkcja przyjmie wartość `true`.

Wynik interpretacji kodu z przykładu 3.93 został pokazany na rysunku 3.21.



Rysunek 3.21. Formularz nie zostanie wysłany, ponieważ pole zawierające adres e-mail jest wypełnione nieprawidłowo

3.11.4. Zerowanie pól formularza

Wypełnione pola formularza można wyczyścić za pomocą metody `reset()`, która powoduje wyzerowanie danych wprowadzonych do wszystkich pól formularza.

Przykład 3.94

```
<!DOCTYPE html>
<html>
<head>
<title>Zerowanie formularza</title>
<meta charset="UTF-8">
<script>
function zeruj() {
  document.getElementById("form1").reset();
}
</script>
<style type="text/css">
div {
  width: 400px;
  background-color: #dfdfdf;
  padding: 10px;
}
.pole {
  position: absolute;
  left: 130px;
}
```

```
</style>
</head>
<body>
<p>Wprowadź swoje dane:</p>
<div>
<form id="form1">
Imię: <input class="pole" type="text" name="imie"><br>
Nazwisko: <input class="pole" type="text" name="nazw"><br><br>
<input type="button" onclick="zeruj()" value="Wyczyść dane">
</form>
</div>
</body>
</html>
```