Lekcja 7

Temat: Funkcje1 – własne funkcje użytkownika

Definiowanie funkcji

Funkcje są nieocenionym narzędziem, które pozwala w wydajny sposób wielokrotnie wykonywać powtarzalne operacje. Załóżmy, że chcesz utworzyć stronę z galerią fotografii, która ma zawierać 50 miniatur. Kiedy użytkownik kliknie jeden z małych obrazków, program ma przyciemnić tło strony oraz wyświetlić tytuł i większą wersję wybranego zdjęcia. Za każdym razem, kiedy internauta wybierze rysunek, proces ten się powtarza. Dlatego na stronie z 50 miniaturami skrypt musi wykonać serię tych samych operacji 50 razy. Na szczęście nie trzeba pisać 50 wersji prawie takiego samego kodu, aby utworzyć galerię fotografii. W zamian wystarczy napisać funkcję z wszystkimi operacjami i uruchamiać ją przy każdym kliknięciu miniatury. **Kod funkcji wystarczy napisać raz, a następnie można wielokrotnie ją uruchamiać.**

Funkcje są to wydzielone bloki kodu przeznaczone do wykonywania określonych zadań. Można o nich myśleć jako o mini programach wykorzystywanych do stworzenia większego, tzn. tworzymy jeden duży pogram, który składa się z małych podprogramów wykonujących jakieś konkretne zadanie.

Schematyczna definicja funkcji ma postać:

```
function nazwa_funkcji()
{
  // kod funkcji
}
```

Nazwa funkcji, podobnie jak inne identyfikatory (np. nazwy zmiennych), może składać się z liter (również znaków narodowych), cyfr oraz znaków podkreślenia. Wolno stosować zarówno wielkie, jak i małe litery, które są rozróżniane, co oznacza, że przykładowe nazwy: **funkcja** i **Funkcja** są traktowane jako różne. **Nazwa funkcji nie może zaczynać się od cyfry**. Nazwy funkcji często zawierają czasownik, na przykład obliczPodatek, pobierzWysokoscEkranu,zaktualizujStrone lub wygasRysunek.

Programiści zazwyczaj umieszczają funkcje na początku skryptu lub w skrypcie zewnętrznym, aby można było używać ich w dalszym kodzie. Instrukcje funkcji nie są wykonywane w miejscu jej utworzenia. Przeglądarka zapisuje je w pamięci, a programista może wywołać funkcję w dowolnym momencie, kiedy chce wykonać określone operacje.

Funkcje są jedyną konstrukcją języka JS, które mogą zostać **wywołane**. Aby wykonać instrukcje znajdujące się wewnątrz funkcji (pomiędzy znakami nawiasu klamrowego), należy ją wywołać. **Wywołanie polega na umieszczeniu w kodzie skryptu nazwy funkcji wraz z występującymi po niej znakami nawiasu okrągłego.**

Przykład 1. Przygotuj stronę html wykorzystujące poniższy kod.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="utf-8">
    <title>Funkcje - l11p1</title>
    <style>
    body{
    background-color: #004477;
    color: #fff;
    font-size: 25px;
    }
    </style>
</head>
<body>
    <script>
    // definicja funkcji bez parametrów
    function dodaj()
        {
            var a= parseInt(prompt("Podaj liczbę a:"));
            var b= parseInt(prompt("Podaj liczbę b:"));
            var s=a+b;
            alert("suma liczb wynosi:"+s);
        }
    // wywołanie funkcji
    dodaj();
    // ponowne wywołanie
    dodaj();
    </script>
</body>
</html>
```

Parametry

Funkcjom można przekazywać parametry, czyli wartości (dane), które mogą wpływać na ich zachowanie lub też być przez nie przetwarzane. Listę parametrów oddzielonych od siebie przecinkami należy umieścić w nawiasie okrągłym za nazwą funkcji:

```
function nazwa_funkcji(parametr1,parametr2,... ,parametrN)
{
  // instrukcje wnętrza funkcji
}
```

W języku technicznym proces przesyłania informacji do funkcji jest nazywany „przekazywaniem argumentów”. Argumenty są wartościami przekazywanymi do funkcji i odpowiadają parametrom definiowanym podczas jej tworzenia.

Przykład 2. Przygotuj stronę html wykorzystującą poniższy kod.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="utf-8">
    <title>Funkcje - l11p2</title>
    <style>
    body{
    background-color: #004477;
    color: #fff;
    font-size: 25px;
    }
    </style>
</head>
<body>
    <script>
    // definicja funkcji z parametrami x i y
    function dodaj(x,y)
        {
            var s=x+y;
            document.write("suma liczb wynosi:"+s);
        }
    // wczytanie danych
    var a= parseInt(prompt("Podaj liczbę a:"));
    var b= parseInt(prompt("Podaj liczbę b:"));
    //  wywołanie funkcji z argumentami a i b
    dodaj(a,b);
    </script>
</body>
</html>
```

Przykład 3. Przygotuj stronę html wykorzystującą poniższy kod.

```
<body>
    <script>
    // definicja funkcji z parametrem
    function wyswietl(x)
        {
            document.write("<h3>"+x+"</h3>");
        }
    //  kilkukrotne wywołanie funkcji z argumentem a
    for (var i=1; i<=4;i++){
    var a= prompt("Co wyświetlić?");
    wyswietl(a);
    }
    </script>
</body>
```

Zwracanie wartości

Przyjmowanie argumentów nie jest jedyną cechą funkcji — mogą one również **zwracać różne wartości**. Czynność ta jest wykonywana za pomocą instrukcji `return`. Jeśli wystąpi ona wewnątrz funkcji, ta jest przerywana i zwraca wartość występującą po return. Konstrukcja wygląda następująco:

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

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

Aby użyć zwróconej wartości, zwykle należy zapisać ją w zmiennej.

Uwaga: Słowo kluczowe return powinno być ostatnią instrukcją funkcji, bo zaraz po tym, gdy tylko interpreter JavaScript je napotka, funkcja zostanie zakończona. Żaden kod umieszczony w funkcji po tym słowie kluczowym nigdy nie zostanie wykonany.

Przykład 4. Przygotuj stronę html wykorzystującą poniższy kod. Zapisz przykład jako *l11p4.html.*

```
<body>
    <script>
    // definicja funkcji z parametrami, zwracającej wartość
    function dodaj(x,y)
        {
            var s=x+y;
            return s;
            // return przerywa wykonywanie funkcji i zwraca wartość s
        }
    // wczytanie danych
    var a= parseInt(prompt("Podaj liczbę a:"));
    var b= parseInt(prompt("Podaj liczbę b:"));
    /*  wywołanie funkcji z argumentami a i b
    i zapamiętanie wyniku zwróconego przez return w zmiennej*/
    var wynik =dodaj(a,b);
    document.write( " suma liczb wynosi: "+wynik);
    </script>
```

Sposoby tworzenia funkcji:

1. **Deklaracja funkcji - Słowo kluczowe, nazwa, parametry**

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

2. **Wyrażenie funkcyjne – wykorzystanie funkcji anonimowych**
   Definiujemy zmienną i do tej zmiennej przypisujemy funkcję anonimową(bez nazwy)

```
var zmienna =function(argumenty)
{
  // instrukcje wnętrza funkcji
  return wartość;
}
```

3. **Konstruktor funkcji**

```
var zmienna = new Function('argumenty', '// instrukcje
wnętrza funkcji', 'return wartość;');
```
- nie polecany do tworzenia funkcji, choć możliwy

4. **Operator FAT ARROW (=>) - Najnowszy sposób, wprowadzony w ES**

```
let zmienna = (argumenty)=>
{
  // instrukcje wnętrza funkcji
  return wartość;
 }
```

Przykład 5. Przygotuj stronę html wykorzystującą poniższy kod.

```
<body>
    <script>
    // definicja funkcji anonimowej z parametrami, zwracającej wartość
    var userName=function(name,surname){
        console.log("nazwa użytkownika to "+name+ " "+ surname);
        return name+ " "+surname;
    }
    //wywołanie funkcji
    userName('Gabriela','Ledachowicz');
    </script>
```

Funkcja strzałkowa jest jedną z funkcji wprowadzonych w wersji ES6 JavaScript. Pozwala tworzyć funkcje w czystszy sposób w porównaniu do zwykłych funkcji.

Na przykład funkcję

```
// function expression
let x = function(x, y) {
   return x * y;
}
```

można zapisać jako za pomocą funkcji strzałkowej:

```
// using arrow functions
let x = (x, y) => x * y;
```

Składnia funkcji strzałek

Składnia funkcji strzałki to:

```
let myFunction = (arg1, arg2, ...argN) => {
    statement(s)
}
```

Tutaj,

- `myFunction` to nazwa funkcji
- `arg1, arg2, ...argN` są argumentami funkcji
- `statement(s)` jest ciałem funkcji

Jeśli ciało ma pojedynczą instrukcję lub wyrażenie, możesz napisać funkcję strzałkową jako:

```
let myFunction = (arg1, arg2, ...argN) => expression
```

Przykład 1: Funkcja strzałkowa bez argumentu

Jeśli funkcja nie przyjmuje żadnego argumentu, powinieneś użyć pustych nawiasów. Na przykład,

```
let greet = () => console.log('Hello');
greet(); // Hello
```

Przykład 2: Funkcja strzałkowa z jednym argumentem

Jeśli funkcja ma tylko jeden argument, możesz pominąć nawiasy. Na przykład,

```
let greet = x => console.log(x);
greet('Hello'); // Hello
```

Przykład 3: Funkcja strzałkowa jako wyrażenie

Możesz także dynamicznie tworzyć funkcję i używać jej jako wyrażenia. Na przykład,

```
let age = 5;

let welcome = (age < 18) ?
  () => console.log('Baby') :
  () => console.log('Adult');

welcome(); // Baby
```

Przykład 4: Wielowierszowe funkcje strzałek

Jeśli treść funkcji zawiera wiele instrukcji, musisz umieścić je w nawiasach klamrowych {}. Na przykład,

```
let sum = (a, b) => {
    let result = a + b;
    return result;
}

let result1 = sum(5,7);
console.log(result1); // 12
```