Java Script

Wprowadzenie

- Programowanie jako sztuka
- Skrypty *server-side* a *client-side*
- *Java* a *JavaScript*

Co to jest zmienna?

var **x** = “*Andrzej Duda*”

Prezydentem Polski jest **x**. *Napisz do* **x**.

Co to jest funkcja?

```
function emerytura() {
        var wiekEmerytalny = 65;
        var wiekObecny = 23;
        var iloscLat = wiekEmerytalny - wiekObecny;
        document.write(“Do emerytury zostało Ci” + iloscLat + “ lata”);
}
emerytura();
```

Co to są obiekty?

Obiekt = predefiniowane i wbudowane funkcje, zmienne, atrybuty, np.:

document.write("*Aktualizacja:* " + **document.lastModified**);

Obiektem jest tutaj “*document*” i ma on własność “*lastModified*” oraz metodę “*write*”.

Pierwszy skrypt

```
<html>

<head></head>

<body>

<h1>Java Script</h1>

<p>Oto mój pierwszy skrypt:</p>

<script>

var x = “Udało się! Skrypt został wykonany!”;

document.write(x);

</script>


</body>

</html>
```

JavaScript – zaawansowany

Instrukcje warunkowe

```
if ((x=="A") && (y!=="B") && (x!=="C")) {
        instrukcja1;
}
else if ((x=="Z") || (x=="W") || (x=="R")) {
        instrukcja2;
}
else {
        instrukcja3;
}
```

Zmienne

- W języku angielskim, **zakres** obowiązywania zmiennych określany jest mianem „*scope*” (*global scope / local scope / current scope*).
- Deklaracja „*var*” tworzy zmienne w bieżącym zakresie (*current scope*), czyli obowiązuje wewnątrz {} oraz wgłąb. Opuszczenie deklaracji „*var*”, domyślnie tworzy zmienną w obiekcie „*window*” (*global scope*). Czyli, poza funkcjami, poprzedzanie zmiennych deklaracją „*var*” jest często bez znaczenia. W obu przypadkach będą to zmienne globalne (obowiązujące w obiekcie „*window*”, który jest „*global scope*”). Chyba że stosujesz się do najnowszej specyfikacji języka (deklaracja „*use strict*”) - wtedy musi być „*var*”.
- Wewnątrz **funkcji**, zmienne poprzedzone „*var*” mają zasięg lokalny (obowiązują wewnątrz {} oraz wgłąb). Bez deklaracji „*var*” - mają zasięg globalny (i nadpisują tę samą zmienną zadeklarowaną wcześniej globalnie). Wewnątrz funkcji, stosowanie „*var*” pozwala „ukryć” zmienne globalne mające tę samą nazwę.
- Domyślnie należy zawsze używać „*var*”, chyba że mamy ważny powód, aby robić inaczej. Lokalne zmienne zawsze działają szybciej niż globalne.
- Niedawno wprowadzono zmienne poprzedzane słowem „*let*”. Ich zakres jest mniejszy niż „*var*” (zakres stosuje się do bloku lub wiersza, nie wychodzi poza {}, ale też nie wchodzi wgłąb).
- Można usuwać z pamięci globalnej tylko zmienne nie utworzone za pomocą „*var*”, gdyż one mogą mieć flagę „*DontDelete*”.
- Zmienne utworzone w funkcji są usuwane po wykonaniu funkcji.

Funkcje

Dwa sposoby tworzenia funkcji:

```
        // Tzw. statement:
        function nazwaFunkcji() {
        }
```

lub

```
        // Tzw. expression:
        var nazwaFunkcji = function() {
        }
```

lub

```
        // Też expression:
        let nazwaFunkcji = function() {
        }
```

We wszystkich przypadkach wywołanie funkcji jest takie samo:

nazwaFunkcji();

Funkcję, która nie ma nazwy można wywołać od razu przy deklaracji (jednocześnie ją deklarujemy i wywołujemy):

(**function ()** {
    jakaśInstrukcja;
}**()**);

Nowy standard ES6 wprowadził także możliwość deklaracji **strzałkowej** (skraca formę). Po strzałce występuje to, co funkcja wykonuje i to co zwraca (słowo „return" pomija się, nawiasy klamrowe też można pominąć, jeśli występuje w nich tylko jedna instrukcja):

(**()** =>{
    jakaśInstrukcja;
}) **()**;

lub z przypisaną zmienną (to jedyny sposób, aby funkcji strzałkowej nadać nazwę):

let **nazwaZmiennej** = **()** =>{
    jakaśInstrukcja;
};
nazwaZmiennej(); *// Wywołanie. Uwaga: Czyli zmienna staje się nazwą funkcji i jest traktowana jako funkcja!*

Funkcja strzałkowa z argumentami:

let **nazwaZmiennejCzyliFunkcji** = **(a, b)** => a+b; *// Gdy funkcja otrzyma argumenty „a" i „b", wykona operację (instrukcję) „a+b" i ją zwróci (return);*

Warto wiedzieć:

- Funkcja podrzędna strzałkowa jest w stanie dziedziczyć argumenty funkcji nadrzędnej. W tradycyjnym zapisie funkcji jest to niemożliwe.
- Funkcja strzałkowa nie może być konstruktorem ani nie może tworzyć prototypów.
- Jeśli funkcja strzałkowa posiada tylko jedną instrukcję, można pominąć klamry {}.

Przekazywanie wartości przez użytkownika

Sposób 1:

```
var bokKwadratu = prompt("Podaj bok kwadratu: ");
```

Sposób 2:

```
<input name="bokKwadratu" id="bokKwadratu">
<script>
function obliczPoleKwadratu() {
    wpisanyBokKwadratu = document.getElementById("bokKwadratu").value;
    alert("Pole powierzchni kwadratu to: " + wynik**2);
}
</script>
<button onclick="obliczPoleKwadratu()">Oblicz...</button>
```

Konstruktory

Konstruktor to funkcja, która przechowuje strukturę obiektu (jego właściwości).

```
let nazwaZmiennejFunkcjiKontrukcyjnej = function(argument1) {
        this.Wlasciwosc1 = argument1; // Ten (jakiś) obiekt będzie miał własność nr 1 (jeśli chcemy go użyć wewnątrz tej funkcji, to zamiasy „this” powinniśmy dać „self”.
};
```

Tworzymy nowy obiekt oparty o konstruktor (o jego nazwę zmiennej):

```
let nazwaZmiennejObiektu = new nazwaZmiennejFunkcjiKontrukcyjnej(‘jakiś Tekst’);
document.write(nazwaZmiennejObiektu.Wlasciwosc1); // Wypisujemy właściwości nowego obiektu, czyli ‘jakiś Tekst’
```

JavaScript a Android

Uwaga! Niektóre przeglądarki w systemie Android (np. *Google Chrom / Chromium*) nie akceptują metod **<audio>**, np. *onplay*, *onended*, *autoplay*, itp. Jest to celowa polityka, której celem jest oddanie decyzji o uruchamianiu dźwięków w ręce użytkownika (a także o pobieraniu danych w tle). Nie będą więc działały skrypty umieszczone w sekcji *<body>* oraz odwołania do zewnętrznych skryptów (tzw. *inline script*) bez względu na to, czy te odwołania zrobimy w sekcji *<head>* czy też w sekcji *<body>*:

<script src=”jakiśSkrypt.js”></script>

Sprawę ratuje umieszczenie jawnego skryptu (nie odwołania!) w sekcji *<head>*.

Należy zauważyć, że *Firefox Mobile* nie ma takich ograniczeń.

Ostatnia aktualizacja: 21 czerwca 2023.