PGS SOFTWARE

PGS SOFTWARE

Akademia Developera – edycja FrontDev

Klasyczny Javascript

JERZY DĘDOR **FRONT-END DEVELOPER**

Co poznamy dziś?

- zmienne i funkcje
- wybrane właściwości języka
- zakres zmiennych, domknięcia, hoisting
- przepływ kontroli
- typy danych
- operatory
- this
- obiektowość i dziedziczenie prototypowe

Dlaczego Javascript?

- (Oczywiście) Frontend
- Backend - Node.js
- Embedded - Node.js
- Programowanie mobilne - React Native, Ionic, etc.
- Performance V8

Czy Javascript jest prosty?

Spoiler: nie.

Dlaczego "klasyczny" Javascript?

- Stare projekty
- To nadal jest standard
- Transpilacja - teraz, kompatybilność wsteczna - zawsze(?)

Zmienne

```
var variableExample = "Łańcuch znaków";
```

- Deklaracja za pomocą "var"
- (Zalecane) Znacząca nazwa
- (Zalecane) Angielski w nazewnictwie

Funkcje

```
function example(argument1, argument2) {
    return argument1 + argument2;
}
var example2 = function(argument1, argument2){
    return argument1 + argument2;
}
exampleObject.methodProperty = function(){...}
```

- Brak możliwości przeciążania nazwy funkcji
- tablica arguments
- funkcje, funkcje anonimowe, metody

Wybrane właściwości języka - dynamiczne typowanie

Nie określamy typu zmiennej przy deklaracji, typ może się zmieniać w trakcie działania programu.

```
var zmienna = "Łańcuch znaków";
zmienna = 5;
zmienna = true;
```

Wygodne, ale niebezpieczne

Wybrane właściwości języka - słabe typowanie

Typ zmiennej jest zmieniany automatycznie, zależnie od kontekstu.

```
5 == "5" // true,
```

Dużo bardziej niebezpieczne niż wygodne.
Przykłady "dziwnych" zachowań Javascriptu od Kyle'a Simpsona
Więcej przykładów

Wybrane właściwości języka - funkcja jest "typem pierwszoklasowym"

- Funkcja może być zapisana do zmiennej
- Funkcja może być przekazana jako argument
- Funkcja może być zwrócona jako rezultat wykonania innej funkcji
- Funkcja może mieć właściwości i metody

Wybrane właściwości języka - jednowątkowość

W dawnych przeglądarkach - jeden wątek *dla wszystkich zakładek*.
Obecnie - jeden wątek dla *obsługi zdarzeń* jednej karty.
Niezależnie od głównego wątku działają Web Workers.

Zasięg widoczności zmiennych i hoisting

```
//remember this? :)
var a = 3;
function exampleFunction(b) {
    a = 4;
    if(b){
        var a = 5;
    }
}
exampleFunction(false);
```

- Zasięgiem zmiennych deklarowanych za pomocą "var" jest funkcja (nie blok!)
- Poza funkcją zasięg jest globalny (nie plikowy!) *)
- Deklaracja zmiennych w dowolnym miejscu funkcji obowiązuje w całej funkcji (hoisting)

Domknięcia (closures)

```
var counter = (function(){
    var closureVariable = 0;
    return function inner(){
        return ++closureVariable;
    }
})()
```

- Domknięcie - funkcja razem z kontekstem jej wykonania (zmienne)
- Funkcja wewnętrzna "zamraża" zmienne z funkcji zewnętrznej
- Domknięcia tworzy się za pomocą funkcji samowywołujących

BONUS: Wzorce projektowe - Module

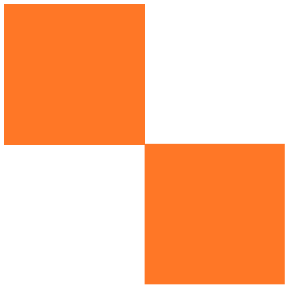
```
(function () {
    // ...
}());
```

- Wzorce projektowe nie muszą być skomplikowane :)
- Moduł służy ochronie globalnej przestrzeni nazw (izolacja kodu)

BONUS: Wzorce projektowe - Revealing Module

```
var ExampleModule = (function(window) {
    var internalVariables = {};
    function ExampleModule() {
        this.exampleMethod = function () {console.log('example method')
    return ExampleModule;
})(window);
```

- Wzorce projektowe nie muszą być skomplikowane :)
- Moduł odsłaniający pozwala tworzyć "prywatne" zmienne i metody

Przepływ sterowania

- If
- Switch
- for, for ... in
- while, do ... while

If

```
if (condition) console.log('condition is met');

if (condition) { console.log('condition is met'); }
    else { console.log('condition is not met'); }

if (condition) { console.log('condition is met'); }
    else if (anotherCondition) { console.log('anotherCondition is not m
    else { console.log('no condition is met'; }
```

Alternatywa: ternary operator

Switch

```
switch (someValue) {
    case "firstPossibility":
        doSomething();
        break;
    case "secondPossibility":
        doSomethingElse();
        break;
    default:
        doSomethingCompletelyDifferent();
}
```

Alternatywa: Obiekt i dynamicznie przypisywane metody

for, for ... in

```
for (var i = 0; i < 5; i++) {
    console.log("Iteration " + i);
    if (something) break;
    if (somethingElse) continue;
}

var a = {
    "key1" : "value1",
    "key2" : "value2" };

for (var b in a) {
    console.log('Key ' + b + 'value ' + a[b]);
}
```

while, do .. while

```
while (someCondition) {
    doSomething();
}

 do {
    doSomething();
} while (someCondition)
```

Typy danych

- String
- Number
- Null
- Undefined
- Boolean
- Object

String

- Primitive vs object
- Literały z użyciem ' lub "
- Pusty string jest 'falsy'
- Specyfikacja

Boolean

- Primitive vs object
- Literały true, false
- Wartości "falsy" : undefined, null, NaN, 0, "" (empty string), false
- Wartości "truthy": wszystkie pozostałe (w tym new Boolean(false)!!!)

Number

- Zarówno liczby całkowite (integer) jak i zmiennoprzecinkowe (float) używają tego samego typu
- Ostrożnie z liczbami dziesiętnymi!
- 01234 - octal, 0xff - hex
- parseInt, parseFloat
- NaN
- Obiekt Math

Null vs undefined

- Undefined - nie określono wartości
- Null - określono wartość jako "brak wartości"
- Undefined nie jest kompatybilny z JSON

```
null === undefined //false
null == undefined //true, ALE
null == "" //itd. -> problem
```

Typy danych - Obiekt

- Literał {}
- Może służyć za mapę
- Specyfikacja
- Przekazywany przez referencję (pozostałe typy - przez wartość)
- Prawie wszystko jest obiektem

Obiekt Array

- Literał []
- Specyfikacja

Operatory

- Arytmetyczne: +, -, *, / , %, ++, --
- Przypisanie: =, +=, -=, *=, /=, %=
- Łańcuchów znaków: +, +=
- Porównania: ==, ===, !=, !==, >, <. <=, >=
- Logiczne: &&, ||, !
- Ternary: condition ? statement1 : statement2
- Bitwise i pozostałe: może innym razem :)

Więcej o operatorach

Ciekawostka

This

This wskazuje na obiekt który jest aktualnym "właścicielem" wykonywanej funkcji.

W praktyce oznacza to, w zależności od kontekstu...

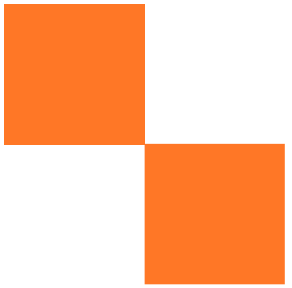
- kod poza funkcją -> obiekt window
- funkcja globalna -> obiekt window lub undefined (strict mode)
- Konstruktor -> tworzony obiekt
- Inline event handler -> element DOM
- Event handler -> źródło eventu
- metoda wywołana na obiekcie -> obiekt
- bind, call, apply...-> wskazany obiekt

This - bind, call, apply

- call, apply - jednorazowe przypisanie *this* i wywołanie
- bind - utworzenie nowej funkcji z przypisanym *this*

```
var obj = { ... }
var func = function(arg1,arg2) { ... }
var args = [...];

func.apply(obj, args);
func.call(obj, args[0], args[1], ...);
var func2 = func.bind(obj); //działa tylko raz!
```

Obiektowy Javascript (przed ES6)

- Konstruktor
- Prototyp
- Dziedziczenie

Obiektowy Javascript (przed ES6) - Konstruktor

```
function Greeter(name){
    this.name = name;
    this.greet = function() {
        console.log("Hi, I'm " + this.name;)}
    }
}
var john = new Greeter("John");
```

Codepen

Obiektowy Javascript (przed ES6) - Prototyp

- Przykłady
- Prototyp to "matryca" dla tworzenia obiektu
- Prototyp może być ustawiony przez konstruktor lub ręcznie
- Obiekt zawiera własności i metody swoje i wszystkich prototypów
- hasOwnProperty sprawdza, czy własność należy do obecnego obiektu czy do drzewa prototypów
- Można modyfikować wbudowane typy (np. String, Array) poprzez zmianę ich prototypów - NIE NALEŻY

Obiektowy Javascript (przed ES6) - Dziedziczenie

Istnieje wiele metod. "Class" w ES6 to tak naprawdę:

```
function Parent(){
    //some initialization
}
Parent.prototype.parentMethod = function() {...}
function Child(){
    Parent.call(this); //this is 'super' call
}
Child.prototype = Object.create(Parent.prototype); //setting inheritanc
Child.prototype.constructor = Child;//resetting constructor
```

Akademia Developera

Dziękuję za uwagę !

Odwiedź:

www.facebook.com/AkademiaDeveloperaRzeszow

JERZY DĘDOR • FRONT-END DEVELOPER