Paweł Rajba
pawel@cs.uni.wroc.pl
http://pawel.ii.uni.wroc.pl/

JavaScript

Agenda

- Wprowadzenie
- Osadzanie JavaScript
- Podstawy języka, typy danych
- Przegląd obiektów
- Konwersja typów
- Zmienne i stałe, predefiniowane stałe
- Operatory, instrukcje sterujące, wyjątki
- Funkcje, obiekty
- Interakcja z przeglądarką i DOM

Wprowadzenie

- Obiektowy język skryptowy
- Posiada podstawowe typy danych, wyrażenia i struktury kontrolne
- Osadzony w np. przeglądarce, może mieć dostęp do dodatkowych elementów (np. DOM)
- Wykonywany
  - Po stronie klienta (np. przeglądarka)
  - Po stronie serwera (np. node.js)
- Drugie życie dzięki AJAX i Web 2.0

Wprowadzenie

- Realizuje standard ECMA-262
  - Specyfikacja: http://www.ecma-international.org/publications/standards/Ecma-262.htm
  - ECMA jest również standardem ISO-16262
- Najnowsza wersja stabilna: 7 (czerwiec 2016)
  - Historia na stronach Wikipedii: https://en.wikipedia.org/wiki/ECMAScript
- Wsparcie w przeglądarkach
  - http://kangax.github.io/compat-table/es6/
- Wersje JavaScript w Mozilla
  - Najnowsza 1.8.5 https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript

Wprowadzenie

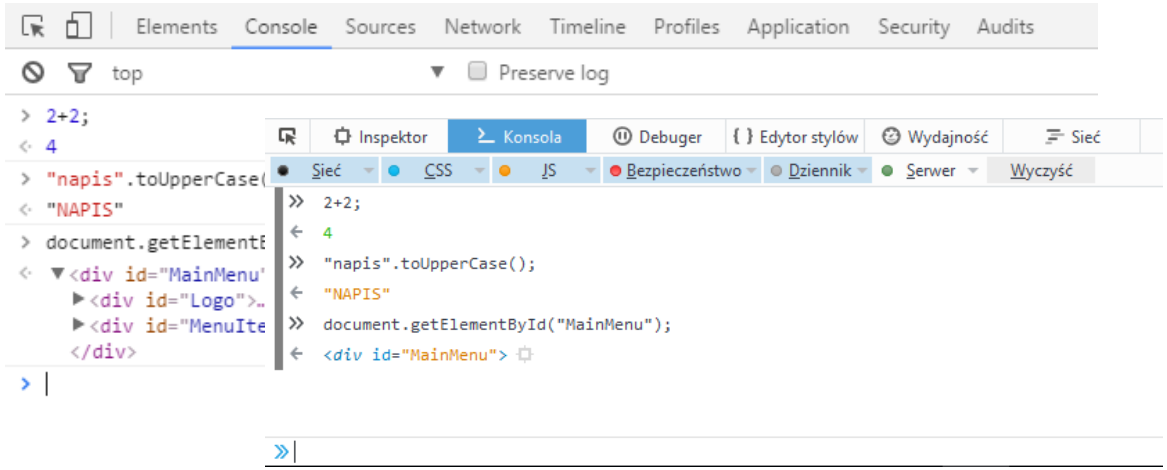
- Silniki JavaScript (JS Engines)
  - V8: Google Chrome, node.js
  - SpiderMonkey: Firefox
  - Chakra: Microsoft Edge
  - Carakan: Opera
- Powyższe silniki są kompilowane (JIT)
  - Mnóstwo porównań, które szybsze: http://developers.redhat.com/blog/2016/05/31/javascript-engine-performance-comparison-v8-charkra-chakra-core/
- Są też interpretery
  - Znacznie wolniejsze
  - Teraz już raczej nieużywane produkcyjnie

Wprowadzenie

- Benchmarki
  - Kraken Benchmark
    - http://krakenbenchmark.mozilla.org/
  - JetStream
    - http://browserbench.org/JetStream/
  - FishIE Tank
    - http://www.fishgl.com/
  - Test Drive
    - https://developer.microsoft.com/en-us/microsoft-edge/testdrive/?o=1
  - GUIMark 2
    - http://www.craftymind.com/guimark2/
  - WebGL Aquarium:
    - http://webglsamples.org/

Wprowadzenie

- Narzędzia
  - Konsole w:
    - Chrome
    - Firefox
  - Online
    - Z bardziej popularnych:
      - http://codepen.io/
      - https://jsfiddle.net/
      - https://jsbin.com/
    - Zestawienie:
      - https://www.sitepoint.com/7-code-playgrounds/



Wprowadzenie

- DEMO
  - Odpalamy JSBin
  - Wklejamy
    - HTML

```
<div id="test"></div>
<button id="button">BUTTON</button>
```

    - CSS

```
#test { font-family: Arial,sans-serif; }
```

    - JS

```
document.getElementById('button').onclick =
    function() {
                document.getElementById('test').innerHTML = '<b>ALA MA KOTA</b>';
   }
```

Osadzanie JavaScript

- Wewnątrz dokumentu HTML
  - ```
    <SCRIPT language="javascript" type="text/javascript">
    <!-
    ...tutaj umieszczamy skrypt...
    //-->
    </SCRIPT>
    ```
- Dołączenie zewnętrznego pliku
  - ```
    <SCRIPT language="javascript" type="text/javascript"
    src="skrypt.js"></SCRIPT>
    ```
- Wewnątrz znaczników:
  - ```
    <A href="javascript:void(0)">Tu klikać</A>
    ```
    ```

Osadzanie JavaScript

- Gdzie umieszczać?
  - W nagłówku
  - W treści dokumentu (początek? koniec?)
- Zdarzenia
  - window.onload
  - $(document).ready()

  http://stackoverflow.com/questions/3698200/window-onload-vs-document-ready
- Unobtrusive JavaScript
  - Analogia do CSS
- Bundling and minification
  - https://jscompress.com/

Na dobry początek ☺

... i wyjaśnienie:
http://www.2ality.com/2012/01/object-plus-object.html

Podstawy

- Komentarze
  - Tak jak w innych językach podobnych do języka C

```
// komentarz jednowierszowy
/* komentarz blokowy */
```

Typy danych

- Proste (primitives)
  - Boolean: true, false
  - null vs. undefined
  - Number
  - String
    - W apostrofach lub cudzysłowach
    - Znaki specjalne (np. \n, \t, \", \\)
  - Symbol
    - https://hacks.mozilla.org/2015/06/es6-in-depth-symbols/
- Obiekty: Object

Typy danych

- Sprawdzanie typów
  - typeof *zmienna*
  - *zmienna* instanceof *typ*
- Rekomendacja
  - Use instanceof
    - for custom types and complex built in types
  - Use typeof
    - for simple built in types
- Dobre wyjaśnienie
  - http://stackoverflow.com/questions/899574/which-is-best-to-use-typeof-or-instanceof

```
var ClassFirst = function () {};
var ClassSecond = function () {};
var instance = new ClassFirst();
typeof instance; // object
typeof instance == 'ClassFirst'; //false
instance instanceof Object; //true
instance instanceof ClassFirst; //true
instance instanceof ClassSecond; //false
```

Przegląd obiektów

- Array
- Map
- Boolean
- Date
- Math
- String
- RegExp

Przegląd obiektów: Array

- Tworzenie tablic:
  - konstruktor Array

```
t = new Array()
t = new Array(10)
t = new Array("A", " B", "C")
```

  - literał

```
t = ["Zebra", "Ryjówka", "Tygrys"]
```

- Właściwości
  - length – liczba elementów w tablicy

Przegląd obiektów: Array

- Wybrane metody
  - concat(t1, t2, …, tN)
  - join(separator)
  - reverse()
  - sort([funkcja_porownujaca])
  - pop()
    - usuwa i zwraca ostatni element tablicy
  - push(elem1,…,elemN)
    - wstawia elementy na koniec tablicy i zwraca nowy rozmiar

Przegląd obiektów: Map

```
var sayings = new Map();
sayings.set("dog", "woof");
sayings.set("cat", "meow");
sayings.set("elephant", "toot");
sayings.size; // 3
sayings.get("fox"); // undefined
sayings.has("bird"); // false
sayings.delete("dog");
sayings.has("dog"); // false

for (var [key, value] of sayings) {
  console.log(key + " goes " + value);
}
// "cat goes meow"
// "elephant goes toot"

sayings.clear();
sayings.size; // 0
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Keyed_collections

Przegląd obiektów: Boolean

- Konstruktor
  - Boolean(wartość)
  - Nie mylić literałów true i false z obiektami
  - Przykład

```
x = new Boolean(false);
if(x) // warunek jest prawdziwy
x = false;
if(x) // warunek jest fałszywy
```

- Metody
  - toString()
    - zwraca wartość przechowaną w obiekcie

Przegląd obiektów: Date

- Konstruktor
  - `new Date()`
  - `new Date(milliseconds)`
  - `new Date(dateString)`
  - `new Date(yr_num, mo_num, day_num [, hr_num, min_num, sec_num, ms_num])`
- Uwagi:
  - numer roku powinien być 4 cyfrowy
  - miesiące: 0=styczeń – 11=grudzień
  - dni tygodnia: 0=niedziela – 6=sobota
  - milisekundy podajemy od 1.1.1970, 00:00:00

Przegląd obiektów: Date

- Uwaga z formatem roku:
  - ```
    d = new Date(04,02,20)
    alert(d.getYear()) // 4
    alert(d.getFullYear()) // 1904
    alert(d.getMonth()) // 2
    ```
  - ```
    d = new Date(2004,07,14)
    alert(d.getFullYear()) // 2004
    alert(d.getYear()) // 104
    alert(Date.parse(2004,07,14) // 1092434400000
    alert(d.getTime()) // 1092434400000
    ```
  - ```
    d = new Date(104,02,20)
    alert(d.getYear()) // -1796
    alert(d.getFullYear()) // 104
    ```
    ```

Przegląd obiektów: Math

- Stałe
  - E – stała Eulera, liczba e, podstawa log naturalnego ok. 2.718
  - LN2 – logarytm naturalny z 2, ok. 0.693
  - LN10 – logarytm naturalny z 10, ok. 2.302
  - LOG2E – logarytm dwójkowy z E ok. 1.442
  - LOG10E – logarytm dziesiętny z E ok. 0.434
  - PI – liczba pi, ok. 3.14159
  - SQRT1_2 – pierwiastek kwadratowy z ½; to samo co 1 przez pierwiastek kwadratowy z 2, ok. 0.707
  - SQRT2 - pierwiastek kwadratowy z 2, ok. 1.414

Przegląd obiektów: Math

- Wybrane metody
  - abs(x)
  - acos(x)
  - asin(x)
  - atan(x)
  - cos(x)
  - exp(x)
  - ceil(x)
  - floor(x)
  - log(x)
  - max(x,y)
  - min(x,y)
  - pow(x,y)
  - random()
  - round(x)
  - sin(x)
  - sqrt(x)
  - tan(x)

Przegląd obiektów: String

- Właściwości
  - length
- Metody
  - charAt(index), charCodeAt(index)
    - Uwaga: numeracja od 0
  - concat(s1, s2, …, sN)
  - fromCharCode(k1, k2, …,kN)
    - `String.fromCharCode(65,66,67) // "ABC"`
  - indexOf(szukany[, od]), lastIndexOf(szukany[, od])
  - split([separator][, limit])
  - substr(start[, długość]), substring(start, end)

Przegląd obiektów: RegExp

- Literał
  - re = /wzorzec/flagi
  - re = /ab+c/i
- Konstruktor
  - re = new RegExp("wzorzec"[, "flagi"])
  - re = new RegExp("\\w+")
  - re = /\w+/

Przegląd obiektów: RegExp

- Właściwości
  - global
    - czy tylko pierwsze wystąpiene, czy wszystkie dopasowania
    - flaga g
  - ignoreCase
    - flaga i
  - multiline
    - rozpatruje każdy wiersz osobno
    - flaga m
  - lastIndex
    - miejsce od którego będzie kolejne wyszukiwanie
    - ma sens z opcją global

Przegląd obiektów: RegExp

- Metody
  - exec(napis)
    - sprawdza, czy napis dopasowuje się do wzorca i zwraca tablicę
  - test(napis)
    - to samo co exec, tylko zwraca true lub false

Przegląd obiektów: RegExp

- Przykłady
  - re = new RegExp("ala+","ig");
  - re.exec("ala ma kota ala")
  - re.lastIndex
  - re.exec("ala ma kota ala")
  - re.lastIndex
  - /a{5}/ig.exec("aaaaaaaa")
  - /a{5,10}/ig.exec("aaaaaaaa")

Przegląd obiektów: RegExp

- Przykłady
  - re = /ab*/ig;
  - t = re.exec("abbadona");
  - alert(t);
  - /^Kasia/.exec("Ala\nKasia")
  - /^Kasia/m.exec("Ala\nKasia")
  - myRe=/d(b+)(d)/ig;
    myArray = myRe.exec("cdbBdbsbz");
    // ["dbBd", "bB", "d"]

Konwersja typów

- JavaScript jest językiem typowanym dynamicznie
- Nie deklarujemy typów, a w razie potrzeby dokonywane są odpowiednie konwersje
- Przykład
  - `var zmienna = 69;`
  - `zmienna = "nowa wartość" // nie będzie błędu`
  - `x = "x = "+40; // zwraca "x = 40"`
  - `y = "69"-9; // zwraca 60`
  - `z = "69"+9; // zwraca 699`

Zmienne i stałe

- Deklaracje zmiennych
  - przez przypisanie wartości: x=5
  - znaczenie var vs. let
    - var: zasięg do końca bieżącej funkcji
    - let: zasięg do końca najbliższego bloku
      - http://stackoverflow.com/questions/762011/let-keyword-vs-var-keyword-in-javascript
- Deklaracja stałych
  - stała nie może zmieniać wartości lub być przedeklarowana
  - Przykład:
    - `const wroclaw = "071";`

Zmienne i stałe

- Mechanizm „hoisting”
  - Dostęp do zmiennych przed ich deklaracją
  - Nie dotyczy, gdy użyto let
  - Wartość undefined

```
/**
 * Example 1
 */
console.log(x === undefined); // true
var x = 3;

/**
 * Example 2
 */
// will return a value of undefined
var myvar = "my value";

(function() {
  console.log(myvar); // undefined
  var myvar = "local value";
})();
```

=

```
/**
 * Example 1
 */
var x;
console.log(x === undefined); // true
x = 3;

/**
 * Example 2
 */
var myvar = "my value";

(function() {
  var myvar;
  console.log(myvar); // undefined
  myvar = "local value";
})();
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_Types#Variable_hoisting

Predefiniowane stałe

- Infinity – stała reprezentująca nieskończoność
  - Infinity jest większa od każdej liczby
  - -Infinity jest mniejsza od każdej liczby
  - Infinity zachowuje się operacjach matematycznych podobnie do nieskończoności
- var wartosc = Infinity;
  - `alert(isFinite(wartosc));`
  - `alert(isFinite(23444));`
- NaN – nie-liczba

Operatory

- Przypisania: =, +=, -=, /=, %=
  ```
  x = 7; x += 4; x %= 10;
  ```
- Porównania: ==, ===, !==, !=, <=, <, >, >=
  ```
  4=='4'; 3==='3'; 3!=="3"; 3<10;
  ```
- Arytmetyczne: ++, --, %
  ```
  x++; --x; x%4;
  ```
- Bitowe: &, |, ^, ~, <<, >>, >>>
  ```
  15 & 9 // 9; 15 ^ 9 // 6; 9 << 2 //
  36;
  ```
- Logiczne: &&, ||, !
  ```
  true && false; !false
  ```

Operatory

- Operator łączenia napisów: +
  `"Paweł "+"Rajba"==="Paweł Rajba"`
- Operator warunkowy: ?:
  `status = (wiek>=18) ? "pełnoletni" : "dziecko";`
- Operator przecinek - stosowany głównie w for
  `for (var i=0, j=9; i<=9; i++, j--) { ... }`
- Operator in – sprawdza, czy obiekt ma szukaną własność
  `auta=new Array("Volvo", "Audi", "Mercedes);`
  `0 in auta; 4 in auta; PI in Math;`
- Operator void – wymusza obliczenie wyrażenia bez zwracania wartości
  `<a href="javascript:void(0)">Click</A>`

Instrukcje sterujące

- Warunkowe: if, switch
- Pętle
  - `for (var i=0; i<10; ++i) { ... }`
  - `do { ... } while (warunek)`
  - `while (warunek) { ... }`
  - `for .. in`

```
tablica = [ ,"Ala","Basia","Małgosia"];
delete tablica[2];
for ( zm in tablica ) {
  alert( tablica[zm] );
}
```

Instrukcje sterujące

- Instrukcja with
  - ```
    var a, x, y;
    var r=10
    with (Math) {
      a = PI * r * r;
      x = r * cos(PI);
      y = r * sin(PI/2);
    }
    ```
    ```

Wyjątki

- Obsługa wyjątków: throw i try..catch

```
function getMonthName(mo) {
  mo = mo - 1; // Adjust month number for array index (1 = Jan, 12 = Dec)
  var months = ["Jan","Feb","Mar","Apr","May","Jun","Jul",
                "Aug","Sep","Oct","Nov","Dec"];
  if (months[mo]) {
    return months[mo];
  } else {
    throw "InvalidMonthNo"; //throw keyword is used here
  }
}

try { // statements to try
  monthName = getMonthName(myMonth); // function could throw exception
}
catch (e) {
  monthName = "unknown";
  logMyErrors(e); // pass exception object to error handler -> your own function
}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling

Funkcje

- Każda funkcja jest obiektem
- Nie ma przeciążania funkcji
- Jest wspierana rekurencja
- Funkcje również obowiązuje hoisting

Funkcje

- Deklaracja
  - `function square(x) { return x*x; }`
  - `var squareF = function (x) { return x*x; };`
  - ```
    function map(f,a) {
      var result=new Array;
      for (var i = 0; i != a.length; i++)
      result[i] = f(a[i]);
      return result;
    }
    ```
- Wywołanie
  - `square(2)` → `4`
  - `squareF(2)` → `4`
  - `(function(x) {return x*x;})(argument);`
  - ```
    map(function(x) {return x*x*x}, [0, 1, 2, 5, 10]);
    // zwróci [0, 1, 8, 125, 1000].
    ```

Funkcje

- Arrow functions
  - Czyli lambda wyrażenia
- Przykład:
  - ```
    function multiply(multiplier, ...theArgs) {
      return theArgs.map(x => multiplier * x);
    }
    var arr = multiply(2, 1, 2, 3);
    ```

```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression
        // equivalent to:  => { return expression; }

// Parentheses are optional when there's only one parameter:
(singleParam) => { statements }
singleParam => { statements }

// A function with no parameters requires parentheses:
() => { statements }
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
```

Funkcje

- Parametry
  - Dodatkowe parametry są ignorowane
    - add(a,b) { … } add(1,2,3) → 3 (OK!)
  - Dla brakujących przypisywane jest undefined
  - Obiekty przekazywane przez referencję, typy proste przez wartość
  - Zmienna arguments
    - Reprezentuje argumenty, typu object
    - Jest trochę niezależna od listy argumentów

```
function add() { document.write(arguments.length); }
add(2,3);
```

  - Rest parameters
    - Pozwala zebranie dodatkowych parametrów

```
function multiply(multiplier, ...theArgs) {
  return theArgs.map(x => multiplier * x);
}
var arr = multiply(2, 1, 2, 3);
```

Funkcje

- Możliwe jest zagnieżdżanie funkcji
- Closure: wewnętrzna funkcja ma dostęp do parametrów funkcji zewnętrznej
- Do odpalenia w JSBin

```
var a = "1";
var outer = function() {
  var b = "2";
  var inner = function()
  {
    document.write(a+" "+b);
  }
  inner();
}
outer();
```

```
$.get('ajax/test.html', function (data) {
    $(resultSelector).html(data);
    alert('Load was performed.');
});
```

Funkcje

- Wybrane predefiniowane funkcje
  - eval(wyrażenie_lub_instrukcje)
  - isFinite(liczba)
  - parseFloat(napis)
  - parseInt(napis)
  - isNaN(napis)

Strict mode

- Ustawia inny tryb wykonywania JavaScript
- Kod może się wykonywać szybciej
- Pewne „ciche” błędy będą powodować wyjątki
- Lista różnic
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode/Transitioning_to_strict_mode
- Włączenie

```
// Whole-script strict mode syntax
"use strict";
var v = "Hi!  I'm a strict mode script!";
```

- Do poczytania
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode

Obiekty

- Obiektowość oparta o prototypy, a nie klasy
  - Prototyp to obiekt (dokładniej: jego schemat), na podstawie którego tworzymy inny obiekt
  - Typem bazowym dla wszystkich obiektów to Object
  - Porównanie

Class-based (Java)	Prototype-based (JavaScript)
Class and instance are distinct entities.	All objects can inherit from another object.
Define a class with a class definition; instantiate a class with constructor methods.	Define and create a set of objects with constructor functions.
Create a single object with the `new` operator.	Same.
Construct an object hierarchy by using class definitions to define subclasses of existing classes.	Construct an object hierarchy by assigning an object as the prototype associated with a constructor function.
Inherit properties by following the class chain.	Inherit properties by following the prototype chain.
Class definition specifies *all* properties of all instances of a class. Cannot add properties dynamically at run time.	Constructor function or prototype specifies an *initial set* of properties. Can add or remove properties dynamically to individual objects or to the entire set of objects.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model

Obiekty

- Właściwości typu object
  - Object.prototype
  - Object.prototype.constructor
- Metody typu object
  - Object.create(proto[, propertiesObj])
  - Object.keys(o), Object.values(o), Object.entries(o)
  - Object.prototype.hasOwnProperty(str)
  - Object.prototype.isPrototypeOf(obj)
  - Object.prototype.getPrototypeOf(obj)

Obiekty

- Obiekt możemy utworzyć na kilka sposobów (1/2)
  - obj = new Object;
    obj.x = 1; obj.y = 2;
  - function Point(nx, ny) { this.x = nx; this.y = ny; }
    var p = new Point(1, 2);
  - person = {
      imie : "Jan", nazwisko : "Kowalski",
      adres : { ulica: "Mala 7”, miasto: "Wroclaw" },
      stan : ["wolny", "niewolny"],
      pokaz : function() { alert(imie+" "+nazwisko); }
    };

Obiekty

- Obiekt możemy utworzyć na kilka sposobów (2/2)
  - Object.create

```
// Animal properties and method encapsulation
var Animal = {
  type: "Invertebrates", // Default value of properties
  displayType : function() {  // Method which will display type of Animal
    console.log(this.type);
  }
}

// Create new animal type called animal1
var animal1 = Object.create(Animal);
animal1.displayType(); // Output:Invertebrates

// Create new animal type called Fishes
var fish = Object.create(Animal);
fish.type = "Fishes";
fish.displayType(); // Output:Fishes
```

Obiekty

- Atrybuty właściwości
  - writable: zapisywalne (lub nie)
  - enumerable: dostępne podczas for..in lub Object.keys
  - configurable: zapisywalne i usuwalne
- Dostęp
  - Obiekt.property
  - Obiekt["property"]
- Iteracja
  - for (var i in obj) { res += i + ": " + obj[i] + "; "; }

Obiekty

- Tworzenie właściwości i gettery/settery

```
function Archiver() {
  var temperature = null;
  var archive = [];

  Object.defineProperty(this, 'temperature', {
    get: function() {
      console.log('get!');
      return temperature;
    },
    set: function(value) {
      temperature = value;
      archive.push({ val: temperature });
    }
  });

  this.getArchive = function() { return archive; };
}
```

```
var arc = new Archiver();
arc.temperature; // 'get!'
arc.temperature = 11;
arc.temperature = 13;
arc.getArchive(); // [{ val: 11 }, { val: 13 }]
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty

Obiekty

- Pola „statyczne” – na poziomie prototypu

```
Car.prototype.color = null;
car1.color = "black";
```

Obiekty

- Dziedziczenie

JavaScript

```javascript
function Employee() {
  this.name = "";
  this.dept = "general";
}
```

Java

```java
public class Employee {
   public String name = "";
   public String dept = "general";
}
```

JavaScript

```javascript
function Manager() {
  Employee.call(this);
  this.reports = [];
}
Manager.prototype = Object.create(Employee.prototype);

function WorkerBee() {
  Employee.call(this);
  this.projects = [];
}
WorkerBee.prototype = Object.create(Employee.prototype);
```

Java

```java
public class Manager extends Employee {
   public Employee[] reports = new Employee[0];
}



public class WorkerBee extends Employee {
   public String[] projects = new String[0];
}
```

Obiekty

- Dziedziczenie
  - Więcej przykładów i do poczytania:
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create
    - https://developer.mozilla.org/pl/docs/Web/JavaScript/Reference/Global_Objects/Function/call
      - Tutaj również inne zastosowania Function.prototype.call

Obiekty

- Ustawienie rozszerzalności obiektu
  - var o = Object.seal(obj)
- Można modyfikować istniejące property
- Nie można dodawać nowych

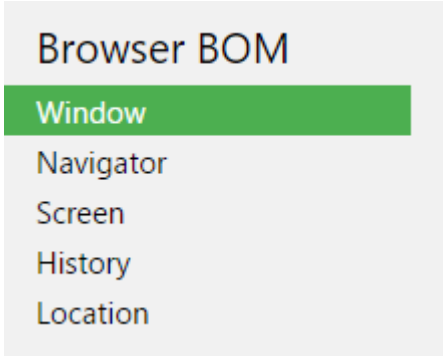
```
var o = Object.seal(obj);

o === obj; // true
Object.isSealed(obj); // === true
```

DEMO

- Prototypes.html
- Singleton.html
- ModulePattern.html
- RevealingModulePattern.html
- RevealingPrototypePattern.html

Interakcja z przeglądarką

- Główne obiekty
  - Window
  - Navigator
  - Screen
  - History
  - Location
- Przegląd szczegółów:
  - http://www.w3schools.com/jsref/obj_window.asp
    - … i pozostałe linki

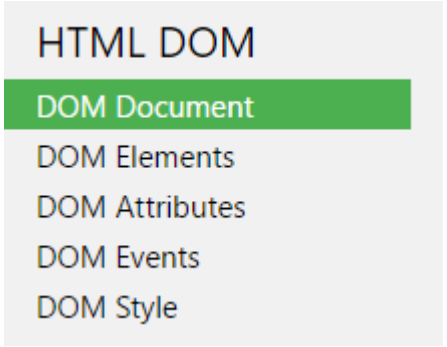


Interakcja z przeglądarką

- Wybrane metody Window:
  - setTimeout(wyrażenie/funkcja, milisekundy)
    - odracza wykonanie funkcji
  - clearTimeout(TimeoutID)
    - anuluje odroczenie i funkcja nie będzie wykonana
  - setInterval(wyrażenie/funkcja, milisekundy)
    - wykonuje wyrażenie co określoną liczbę milisekund
  - clearInterval(TimeoutID)
    - przerywa wykonywanie funkcji

Interakcja z DOM

- Główne kategorie
  - DOM Document
  - DOM Elements
  - DOM Attributes
  - DOM Events
  - DOM Style
- Przegląd szczegółów
  - http://www.w3schools.com/jsref/dom_obj_document.asp
    - … i pozostałe linki



Interakcja z DOM

- Warto zaznaczyć metody:
  - document.getElementById('id')
  - document.getElementsByClassName('classname')
  - document.getElementsByName('name')
  - document.getElementsByTagName('tagname')

DEMO

- okienko.html
- ramki.html
- dokument.html
- obrazki.html
- formularze.html
- coords.html
- cyfry.html
- dynamic.html

Do poczytania

- Mozilla Developer Network
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript
- JavaScript and HTML DOM Reference
  - http://www.w3schools.com/jsref/default.asp
- JavaScript Design Patterns
  - http://www.dofactory.com/javascript/design-patterns
- JavaScript Patterns
  - http://shichuan.github.io/javascript-patterns/
- 4 JavaScript Design Patterns You Should Know
  - https://scotch.io/bar-talk/4-javascript-design-patterns-you-should-know