Web Development Training

Accelerated

About me

Damian Sosnowski

working as a UI Architect @GFT Poland in Poznań

JavaScript developer for 5 - 6 years.

I've worked with multiple frameworks like: jQuery, Backbone, ExtJS, Sencha Touch, Angular...

About me

Now working mostly in Angular (2.x - 4.x), creating enterprise SPA.

Head of the Digital Family (UX and UI development)

Trainings

- Multiple internal trainings at GFT
- External trainings for other companies and communities
- Trainings and workshops performed for GFT Clients, including Barclays: London, Prague, Kiev
- Lectures on the University in Poznań

Your turn!

Tell me about yourself.

Let's start with a...

test! :)

https://github.com/sosnowski/trainings

JavaScript Quiz

./tasks/basics/quiz

Run test.html to check results, edit quiz.js to provide the solution

Map Array

./tasks/basics/map_array

Run test.html to check results, edit code.js to provide the solution

Event Emitter

./tasks/basics/event_emitter

Run test.html to check results, edit ee.js to provide the solution

BMI Counter

./tasks/basics/dom_bmi

Run index.html to check results, edit app.js to provide the solution

JavaScript

Language of the web

JavaScript:

- It's not Java! ;)
- Dynamic, scripting language
- No strict typing
- First-class functions
- Object Oriented (but users prototypes)

Variables

```
//Variables declaration

var a = 2;
var s = "hello!";

var b = 3, c = 'test', d;

gl = 'this is global variable'; //no "var"! Always creates global variable
```

Scope

```
var a = 2; // global
function test () {
    var b = 3; // local
    var a = 4; //local
};

console.log(a); // 2
```

Functional scope

```
function test () {
    if (true) {
        var b = 3;
    }
    var a = 2;
    console.log(a); // 2
    console.log(b); // 3
};
```

```
function test () {
    var a, b;
    if (true) {
        b = 3;
    }
    a = 2;
    console.log(a); // 2
    console.log(b); // 3
};
```

Ecma Script 6 / TypeScript

```
let a = 2; //block scope!

function test () {
    if (true) {
        let b = 3;
    }
    let a = 2;
    console.log(a); // 2
    console.log(b); // undefined
};
```

Types in JavaScript

```
var a = 12; //Number
var b = 1.234; //Number
var c = "some string"; //String
var d = true; //Boolean
var e = null; //null
var f; // undefined
var g = undefined;
var h = 2 * "duck"; //NaN
```

```
typeof 12; // "number"
typeof "12"; // "string"
typeof "smth"; // "string"
typeof true; // "boolean"
typeof undefined; // "undefined"
```

Be careful

```
typeof null; // .... "object"
typeof NaN; // "number"
```

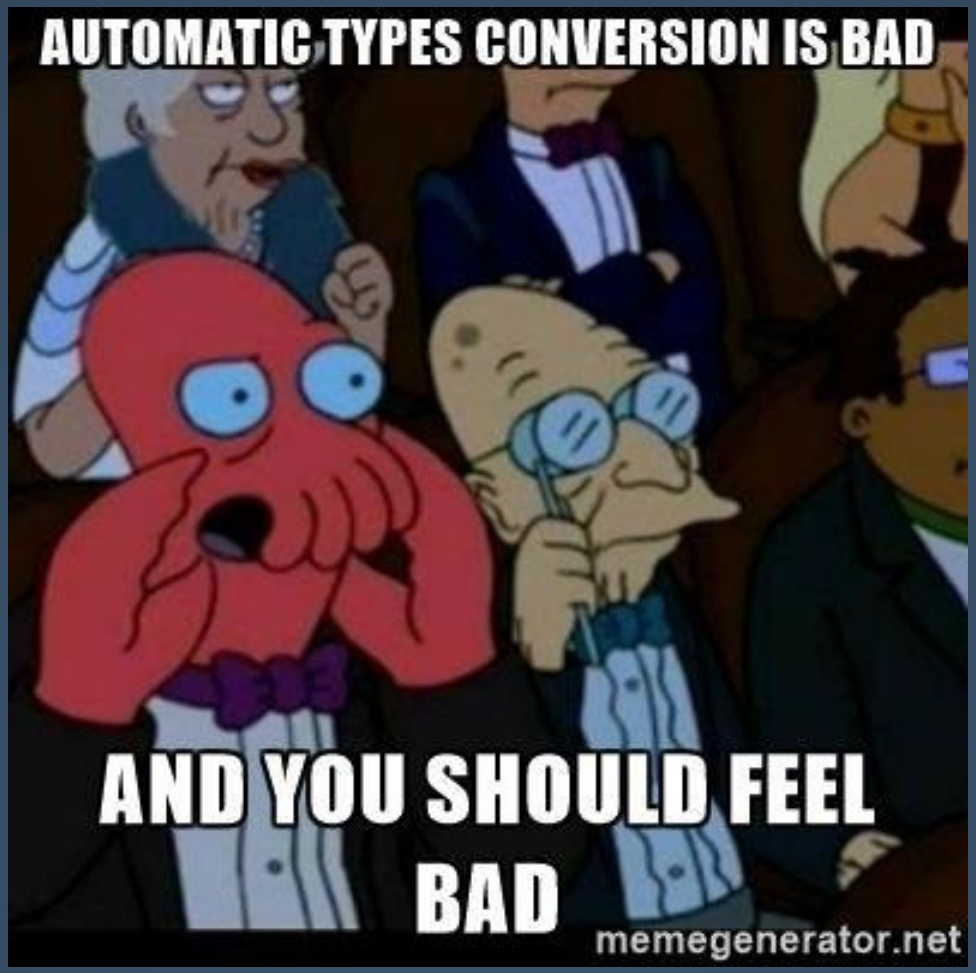
Operators

```
a = 2 + 2;
a = 2 - 2;
a = 2 * 2;
a = 2 / 2;
a += 2;
a -= 2;


//Be careful with strings

a = "test" + "smt"; //testsmth
a = "test" + 2; "test2"
a = "3" - 2;
a = "duck" - 5;
a = + "3" + 2;
```

Operators

```
2 == 2; // true
"test" == "test"; //true

"2" == 2; // true
0 == ""; // true
0 == false; // true
```

```
2 === 2; // true
"2" === 2; // false
0 === false; // false
false === false; // false
```


AUTOMATIC TYPES CONVERSION IS BAD
AND YOU SHOULD FEEL BAD
memegenerator.net

```
0 == +[]; // true
```

Logical operators

```
//for boolean values
false || true || false // true
true && false // false
true && true // true
```

```
//can be used with other values as well

false || 0 || "" || "hello" || 23 // "hello"
{} && "false" && 0 // 0
```

```
!true; //false
!!true; //true

!"something"; // true
!!0; //false
```

Falsy values

```
false, 0, null, undefined, "", NaN;
```

Arrays

```
var arr = [];
var arr2 [1, 2, "tet", null, []];
arr.push("test");

arr.length; // 1;
```

Objects

```
var obj = {};
var obj2 = {
    key: 'value',
    smth: [],
    method: function () {}
};
obj.foo = 'bar';
```

```
obj2.key; // value;
obj2['key'] // value;

var key = 'smth';
obj2[key]; // []
```

Functions

Declaring a function

```
function funcName() {
    //function declaration
}

var func = function () {
    //function expression
}

(function () {
    //IIFE
}())
```

```
funcName();
func();
```

Returning values

```
var sum = function (arg1, arg2) {
  return arg1 + arg2;
};

sum(5, 4); // 9
```

Arguments

```
var sumAll = function () {
  var sum = 0;
  for (var i = 0; i < arguments.length; i++) {
      sum += arguments[i];
  }
  return sum;
};
```

Functions in ES6

```
var sum = (arg1, arg2) => {
    return arg1 + arg2;
};
```

Arguments in ES6

```
let sumAll = (...args) => {
    return args.reduce((start, number) => {
        return start + number;
    }, 0);
};

let res = sumAll(1, 2, 3, 4, 5);
console.log(res);
```

Closure

```
var times = function (t) {
   return function (number) {
     return t * number;
  };
};

var timesTwo = times(2);
timesTwo(5); // 10
timesTwo(4); //8


var addListener = function (eventName, listener) {

    //...
    collectionOfListeners.add(eventName, listener);
    //...

    return function () {
        collectionOfListeners.remove(eventName, listener);
    }
};

//register listener
var removeListener = addListener('click', someFunction);

//when you don't want to listen for events anymore, you can remove the listener
//Thanks to closue, you don't have to provide any arguments
removeListener();
```

Function context

```
var obj {
    method: function () {
        console.log(this);
    }
}
obj.method(); // obj
```

```
var func = function () {
     console.log(this);
};

func(); // window (global object);
```

Context is dynamic

```
obj.method(); // obj
var func = obj.method;

func(); // window
```

Every function has its own context

```
var obj = {
  method: function () {
    console.log(this);

    (function () {
        console.log(this);
    }());
  }
}
obj.method(); // obj; window;
```

Context can be modified

```
var obj = {}, func = function () {
    console.log(this);
};

func(); // window
func.apply(obj); //obj
func.call(obj); //obj

func(); //window
```

```
var func2 = func.bind(obj);

func2(); // obj
func2(); // obj

// but

func(); //global
```

Context in ES6

```
function Person(){
  this.age = 0;

  setInterval(() => {
    this.age++; // |this| properly refers to the person object
  }, 1000);
}

var p = new Person();
```

translated to ES5

```
function Person(){
  this.age = 0;

  setInterval((function () {
    this.age++; // |this| properly refers to the person object
  }).bind(this), 1000);
}

var p = new Person();
```

Object Oriented JavaScript

The story about prototypes

Objects in JS are simple

```
var obj = {};
obj.name = 'Jack';
```

```
obj.sayHello = function () {
    return 'Hi, ' + this.name;
};

obj.sayHello(); // Hi, Jack
```

```
var obj = {
    name: 'Jack',
    sayHello: function () {
        return 'Hi, ' + this.name;
    }
};
```

But inheritance is confusing

PROTOTYPES

Create object with a prototype

```
var Person = {
    sayHello: function () {
        return 'Hi ' + this.name + '!';
    }
};
```

```
var obj = Object.create(Person);
obj.name = 'Damian';

var obj2 = Object.create(Person);
obj.name = 'Zdzichu';

obj.sayHello(); // Hi Damian!
obj2.sayHello(); // Hi Zdzichu!
```

```
obj !== obj2
obj.sayHello === obj2.sayHello
```

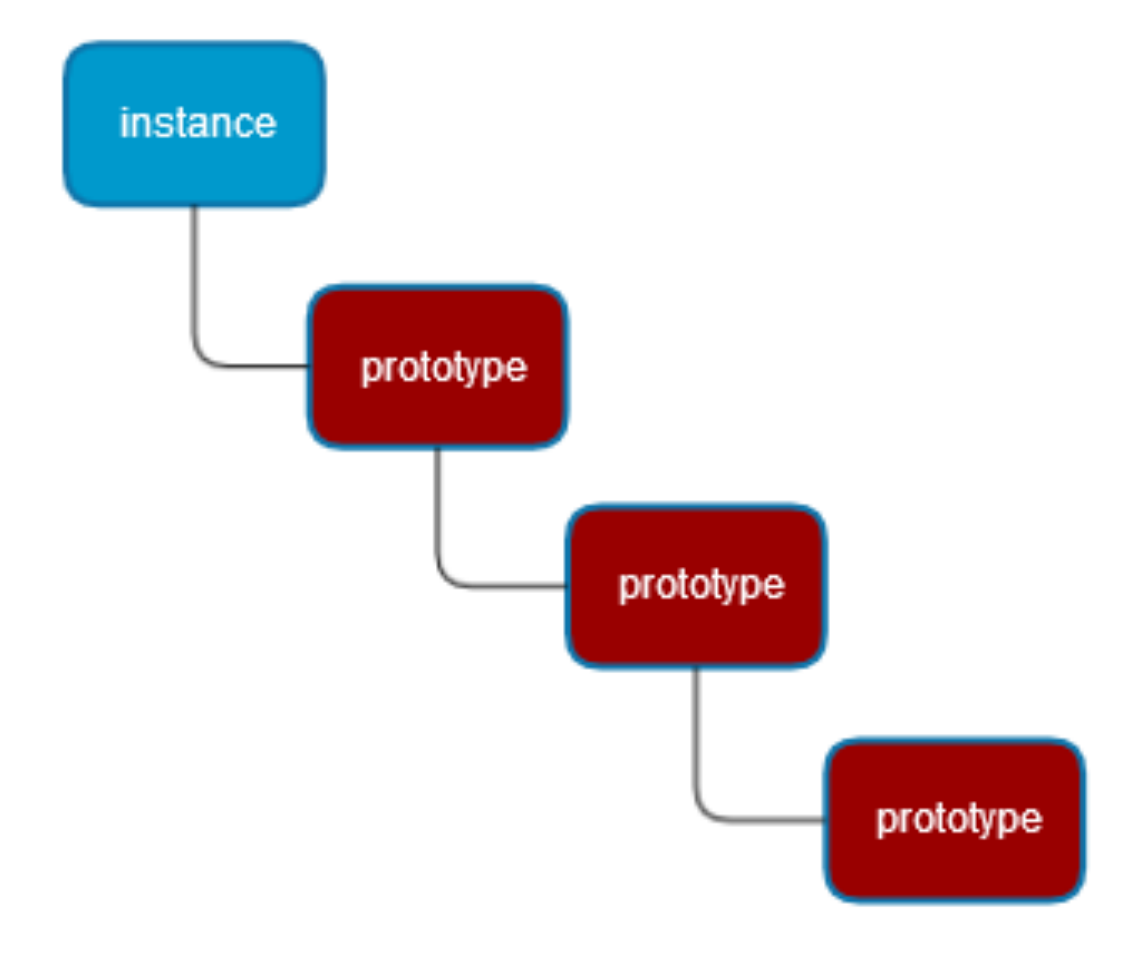
Inheritance - no classes

```
var Dog = {
    run: function () {
        return 'I run!';
    },
    bark: function () {
        return 'Hau! Hau!';
    }
};

var dog = Object.create(Dog);
dog.bark(); // Hau! Hau!
```

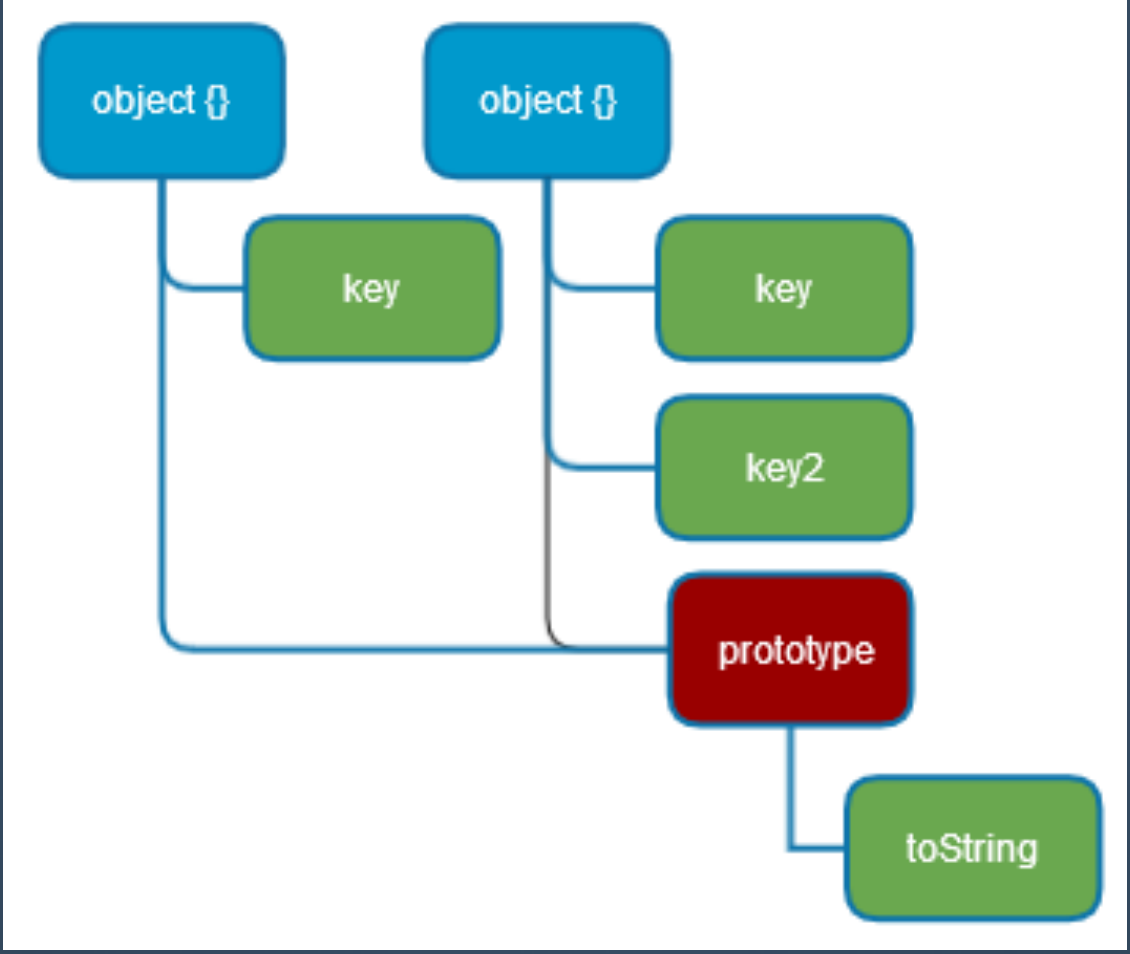
```
var SmallDogProto = Object.create(Dog);
SmallDogProto.bark = function () {
    return 'pi pi pi';
}
```

```
var smallDog = Object.create(SmallDogProto);
smallDog.run(); // I run!
smallDog.bark(); // Pi pi pi
```

Inheritance = prototype chain



Prototype and multiple instances



Where is prototype?

```
obj.prototype; // NO!
obj.__proto__;
Object.getPrototypeOf(obj);

obj.hasOwnProperty(prop)
```

Classes

```
var arr = new Array();
var obj = new Object();

var myInst = new MyClass();
```

Class in JavaScript is a function...

```
function MyClass() {

}

var inst = new MyClass();
```

Instance attributes

```
var Person = function (name) {
    this.myName = name;
    this.age = 18;

    this.doSmth = function () {}
}
```

```
var inst1 = new Person('Gucio'), inst2 = new Person('Zdzisio');

inst1.myName; // Gucio
inst1.doSmth();

inst1.doSmth !== inst2.doSmth; // true
```

Prototype to the rescue!

```
var Person = function (name) {
        this.myName = name;
}

Person.prototype.sayHello = function () {
        return 'Hi I am ' + this.myName + '.';
};
```

```
var dev1 = new Person('Damian');
var dev2 = new Person('Pawel');

dev1.sayHello === dev2.sayHello;

dev1.sayHello(); // Hi I am Damian.
dev2.sayHello(); // Hi I am Pawel.
```

If the class is just a function

How does it work? How the object is created with a proper prototype?

```
var inst = MyClass('Damian'); //no "new"!!
//No error!

inst === undefined;

window.myName; // 'Damian'
```

The "new" operator!

- If constructor ret**urns some** object, return i**t. If not, return t**he initially created ob**ject**
- Execute the constructor function in the context of the created object
- Take **prototype** property of the **contructor function** and make it a part of **prototype chain** of the object
- Create object