1.
OOP przypomnienie

Function execution context
this keyword *

```
function hello() {
    console.log(this.name + " mówi cześć!");
}

var person = {name: "Zenek", hello: hello}

hello() // undefined

person.hello() // Zenek mówi cześć!

var helloZenek = hello.bind(person);

hello.bind(person)() // Zenek mówi cześć!
HelloZenek() // Zenek mówi cześć!
```

Object creation
Object literal *

The simplest way to create object in JS is object literal.

```
var cat = {
   name: 'Fluffy',
   age: 1,
   sound: 'Meeeeow!',
   makeSound: function() {
      console.log(this.sound);
   },
   speak: function() {
      console.log('Sorry cats can’t speak');
   }
};
```

Object creation
Factory functions *

Factory function is a function that creates something using arguments that are passed to function. Usually it creates an object:

```
function catFactory(name) {
   return {
      name: name,
      sound: 'Meeeeow!',
      speak: function(){
         console.log('Sorry cats can’t speak');
      }
   }
}
```

Task 1

“

Make a factory function that produces a cat object with name property that will passed through function parameter.

Make several cats and assign them to some variables.

Object creation
Object.assign()

The **Object.assign()** method is used to copy the values of all own properties from one or more source objects to a target object. It will return the target object.

```
const obj1 = { a: 1, b: 2, c: 3 }

const obj2 = Object.assign({c: 4, d: 5}, obj1)

console.log(obj2) // { a: 1, b: 2, c: 4, d: 5}
```

Inheritance
What is prototype in JS?

When it comes to inheritance, JavaScript only has one construct: objects. Each object has a private property which holds a link to another object called its prototype. That prototype object has a prototype of its own, and so on until an object is reached with null as its prototype. By definition, null has no prototype, and acts as the final link in this prototype chain.

Nearly all objects in JavaScript are instances of Object which sits on the top of a prototype chain.

Inheritance
Constructor functions *

```
function Cat(name, age){
    this.name = name
    this.age = age
    this.sound = “Meeeeow!”,
    this.makeSound = function(){
        console.log(this.sound)
    }
    this.speak = function(){
        console.log(‘Sorry cats can’t speak’)
    }
}
```

Inheritance

Constructor functions *

```
function Cat(name, age){
    this.name = name
    this.age = age
    this.sound = 'Meeeeow!'
}

Cat.prototype.makeSound = function(){
    console.log(this.sound);
}
Cat.prototype.speak = function(){
    console.log('Sorry cats can’t speak');
}
```

Inheritance
Constructor functions with IIFE *

```
var Cat = (function() {
  function Cat(name) {
    this.sound = "Meow";
  }

  Cat.prototype.makeSound = function makeSound() {
    console.log(this.sound);
  };

  return Cat;
})();
```

Object creation sumup
Object literal

```
var cat = {
   name: 'Fluffy',
   age: 1,
   sound: 'Meeeeow!',
   makeSound: function() {
      console.log(this.sound);
   },
   speak: function() {
      console.log('Sorry cats can’t speak');
   }
};
```

Object creation sumup
Constructor & factory

```
function Cat(name){
     this.name = name;
     this.sound = ‘meow’;
     this.speak = function(){
          console.log(this.sound);
     };
}

Var cat1 = new Cat(‘puszek’);
```

```
function catFactory(name) {
   return {
      name: name,
      sound: ‘meow’,
      speak: function(){
         console.log(this.sound);
      }
   }
}

var cat2 = catFactory(‘puszek’);
```

Task 2

Make a constructor function that makes cats object, with name, sound properties, and speak, makeSound methods, from scratch.