# ES6

## ES6
## ECMAScript 2015

- It’s a standardized specification for JS
- It provides an unified way to write code
- ES6 (ECMAScript 2015) introduced in June 2015
- provides a lot of big changes into JS syntax

## ES6

## let - block scope variable

```
for (var i = 1; i <= 3; i++){
    console.log(i)
}

console.log('i = ', i)

// 1
// 2
// 3

// 4
```

```
for (let i = 1; i <= 3; i++){
    console.log(i)
}

console.log('i = ', i)

// 1
// 2
// 3

// ReferenceError: i is not
// defined
```

## ES6

## let - redeclaration

```
if(true) {
    var name = ‘Marry’
    var name = ‘John
}

console.log(name)

// John
```

```
if(true) {
    let name = ‘Marry’
    let name = ‘John
}

console.log(name)

// SyntaxError: Identifier
// 'name' has already been
// declared
```

# ES6
## let - hoisting

```
function f() {
    x = 2
    var x
    console.log(x)
}

f()

// 2
```

```
function f() {
    x = 2
    let x
    console.log(x)
}

f()

// ReferenceError: x is not
// defined
```

## *Task 1*

*Try let examples in console*

** Create for loop with setTimeout that console.log 1,2,3,4,5 after 3 seconds*

## ES6
## const

```
const max = 10

max = 15

// TypeError: Assignment
// to constant variable
```

```
const prop = {
    max: 10
}

prop.max = 15
console.log(prop.max) // 15

prop = {}

// TypeError: Assignment
// to constant variable
```

## ES6
## const

Additionally the value **CAN’T** be reassigned.

But we can change object’s (array is an object) contents that is assigned by reference to **const.**

## ES6
## let & const

**let** and **const** are NOT hoisted!

## *Task 2*

*Try const examples in console*

** Create for loop with setTimeout & const that console.log 1,2,3,4,5 after 3 seconds*

## ES6
## template strings

Before ES6 creating complex strings with dynamic data was very unpleasant (string concatenation).

- Template strings allow us to inject variable values into a string using special syntax
- Additionally with template strings we can create multiline strings with ease

To create template string use backtick ` instead of ‘ or “ !

## ES6
## template strings

ES5

```
var name = “Maciek”

var message = “Hello ” + name

console.log(message)

// Hello Maciek
```

ES6

```
const name = 'Chris'

const  message = `Hello $
{name}`

console.log(message)

// Hello Chris
```

## ES6
## template strings

ES5

```
var multilineMessage =
'first line\n' +
'second line\n' +
'third line'

// first line
// second line
// third line
```

ES6

```
const multilineMessage =
`first line
second line
third line`

// first line
// second line
// third line
```

## ES6
## template strings

Inside template string we can use **ANY** JavaScript expression - everything thats produce value!

## *Task 3*

“ *Make variables one with your name, second with sentence & template string, use name in template string and try to console.log*

** Make an object literal with your name and favouriteColor.*
*Console log string like this:*
*“I’m **Your Name**. I like **green**! ” bold items should be from object.*

## ES6
## arrow functions

```
var up = function(param) {
    return param.toUpperCase()
}

console.log(up('abcd'))

// ABCD
```

```
const up1 = (param) => {
    return param.toUpperCase()
}

const up2 = (param) =>
param.toUpperCase()

const up3 = param =>
param.toUpperCase()

console.log(up1('abcd'))
// ABCD
```

# ES6

## arrow functions - shorten syntax

```
() => {
    /*..*/
    return ''
}
(x) => {
    /*..*/
    return x * x
}
(x, y) => {
    /*..*/
    return x * y
}
```

```
(x) => {
    return x * x
}

x => {
    return x * x
}

x => x * x
```

```
x => {
    return {
        a: x,
        b: x * x
    }
}
x => (
    {
        a: x,
        b: x*x
    }
)
```

## ES6
## arrow functions

```
function Animal(sound) {
        this.sound = sound
        this.makeSound = function() {
                console.log(this.sound)
        }
}

const cat = new Animal('meouw')
cat.makeSound() // meouw

const makeSound = cat.makeSound
makeSound() // undefined
```

```
function ArrowAnimal(sound) {
        this.sound = sound
        this.makeSound = () => (
                console.log(this.sound)
        )
}

const arrowCat = new ArrowAnimal('meouw')
arrowCat.makeSound() // meouw

const arrowMakeSound = arrowCat.makeSound
arrowMakeSound() // meouw
```

## ES6
## arrow functions - sum up

- if we have only one expression in function body we can omit return - value is returned by default
- if we want to return object from arrow function without return we need to wrap it in (), so JS can evaluate it as an expression not a block of code
- we can omit parameters brackets () when we have only one parameter
- arrow function CAN’T be constructor function
- arrow function CAN’T be named as normal function - it only can be assigned to variable
- arrow function have lexical scope this - it have this as it was at the moment of declaration not execution

## *Task 4*

“

*Make an object with property counter and method start.*

*Method start should increase counter every each second, use console log.*

## Task 4

“

```
const myCounter = {
  counter: 0,
  start: function() {
    ...
  }
}

MyCounter.start() // 1 2 3
```

## ES6
## destructuring

With destructuring we can get access to values nested in arrays of objects very easily.

## ES6
## destructuring - arrays

ES5

```
const arr = [10, 20, 30]

const a = arr[0]
const b = arr[1]
const c = arr[2]

console.log(a, b, c)

// 10 20 30
```

ES6

```
const arr = [10, 20, 30]

const [a, b, c] = arr

console.log(a, b, c)
```

## ES6
## destructuring – arrays

USING ONLY SPECIFIC INDEX:

```
const arr = [10, 20, 30]

const [,,a] = arr
console.log(a) // 30

const longArray = [1, 2, 3, 4, 5, 6]

const [,second, , fourth] = longArray
console.log(second, fourth) // 2 4
```

## ES6
## destructuring - objects

WITHOUT DESTRUCTURING

```
const user = {
    name: 'Bob',
    surname: 'Builder'
}

const name = user.name
const surname = user.surname

console.log(
`${name} ${surname}`
)
// Bob Builder
```

**WITH DESTRUCTURING**

```
const user = {
    name: 'Bob',
    surname: 'Builder'
}

const { name, surname } = user

console.log(
`${name} ${surname}`
)
```

# ES6
## destructuring – objects

NAMING PARAMETERS DIFFERENTLY

```
const user = {
    name: 'Bob',
    surname: 'Builder'
}

const {
    name: userName,
    surname: userLastName
} = user

console.log(`${userName} ${userLastName}`) // Bob Builder
```

## ES6
## spread operator - arrays

ES5

```
var arr = [10, 20, 30];
var element = 50

arr.push(element);



// [10, 20, 30, 50]
```

ES6

```
const arr = [10, 20, 30];
const element = 50;

const arr2 =
[...arr, element]
```

# ES6

## spread operator - objects

ES5

```
var obj1 = {
    name: ‘puszek’,
}

var obj2 =
Object.assign(obj1, {age:
20});


// {name: ‘puszek’, age: 20}
```

ES6

```
var obj1 = {
    name: ‘puszek’,
}

var obj2 =
{...obj1, age: 20};
```

## “ Task 5

*Play with destructuring arrays and objects in Babel editor on Babel official site.*

** Update Game with destruct & spread operator*

## ES6
## modules

Modules are way to combining parts of code that is written in different files **OR NPM MODULES!**

We can **import** and **export** variables from and into files and use them!

## ES6
## modules - named exports

You can export multiple variables, functions etc. from a module as **named exports.**

```
export const name1 = () => {/*...*/}
export const name2 = () => {/*...*/}
export const name3 = () => {/*...*/}
```

## ES6
## modules - named imports

```
import {name1, name2, name3} from 'moduleName'

import {anotherName} from ‘./path/to/module/yourFileName'

import {reallyReallyLongModuleExportName as shortName}
from ‘modName'

import * as yourName from 'yetAnotherModuleName'
```

## ES6
## modules - default export

There can be only one **default export** in a module.

```
export default function() {/*...*/}

export default () => {/*...*/}

export default class SomeClass {
  /*...*/
}
```

## ES6
## modules - named imports

```
import yourName from ‘moduleName’
```

Important part is that default export ISN’T named, so name that YOU provide after import keyword can be anything!

## ES6
## modules - default imports

```
import name from 'moduleName'

import name, {anotherName} from ‘yourFileName'
```

## *Task 7*

"

*Install by NPM package called moment (momentjs.com). Import it into your code and console.log current date formatted "MMM DD YY" ("Jan 10 18")*

## Task 8

“

*Make function that console logs current date in prev task format.*
*Export it and import in another file.*
*Invoke imported function.*