



npm i moduły ES


Michał Michalczuk

Michał Michalczuk

michalczukm.xyz

- ✚ IT Trainer @ infoShare Academy
- ✚ Full-Stack Software Developer
- ✚ Senior JavaScript Developer
@ SparteZ / Atlassian



 nodejs



nodejs?

- środowisko wykonawcze JavaScript
- server-side
- oparte o V8 (silnik JavaScript, ten co w Chrome)
- cross-platform
- open-source
- od 2009



nodejs vs browser



vs



nodejs vs browser



Cookies,
localStorage,
browser stuff



po co nam *nodejs* na front-end'zie?

- budowanie projektów
 - bundlowanie kodu
 - minifikowanie kodu
 - etc
- serwery developerskie
- narzędzia developerskie
 - linter, analiza kodu etc.
- zarządzanie zależnościami



nvm

Node Version Manager

Aby wygodnie zarządzać wersją nodejs na naszej maszynie.

github.com/nvm-sh/nvm

nvm

Node Version Manager

Aby wygodnie zarządzać wersją nodejs na naszej maszynie.

github.com/nvm-sh/nvm

instalacja: github.com/nvm-sh/nvm#installation-and-update

```
# zainstaluj Long Term Support
```

```
nvm install --lts
```

```
# zainstaluj konkretną wersję
```

```
nvm install 12.13.0
```

```
# ustaw aktualną wersję (musi być zainstalowana)
```

```
nvm use 12.13.0
```

```
# twoje wersje
```

```
nvm list
```

Ćwiczenie 1.

1. Zainstaluj nvm

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.1/install.sh | bash
```

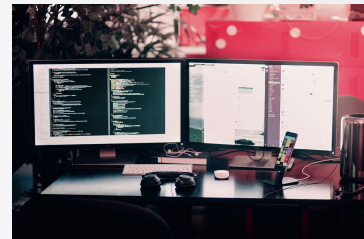
```
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.nvm" ||  
printf %s "${XDG_CONFIG_HOME}/nvm")" [ -s "$NVM_DIR/nvm.sh" ] && \  
"$NVM_DIR/nvm.sh" # This loads nvm
```

2. Zainstaluj nowy LTS nodejs:

```
nvm install --lts
```

3. Sprawdź czy masz poprawną wersję

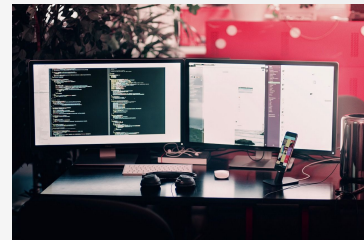
```
node -v
```



Ćwiczenie 2.

1. Stwórz nowy plik .js `my-file.js`
2. Wyświetl w nim na konsoli ciąg znaków od 1 do 10 (1,2,3,...10)
3. Uruchom ten plik w nodejs z konsoli

```
node my-file.js
```





npm



npm: Node Package Manager

- manager pakietów nodejs
- pakiet = kod którego możemy użyć
- zależność = pakiet którego używamy
- npm potrafi
 - zainstalować zależności wg listy w package.json
 - publikować pakiety
 - wersjonować pakiety
- nie trzymamy zależności w naszym repo
- każdy projekt jest pakietem (nie musimy go publikować)



```
# stwórz pakiet
```

```
npm init
```

```
# stwórz pakiet z domyślnymi wartościami
```

```
npm init -y
```

```
# pojawia się nowy plik
```

```
# package.json
```

```
{  
  "name": "my-project",  
  "version": "1.0.0",  
  "main": "index.js",  
  "author": "Michał Michałczuk <michalczukm@gmail.com> (https://michalczukm.xyz)",  
  "license": "MIT",  
}
```


npm: instalacja zależności

- znajdź zależność:
 - npmjs.com
 - często projekty mają wszystko w readme na github'ie
- zainstaluj zależność
 - wejdź do folderu z projektem
 - `npm install nazwa-zaleznosci`
 - nowy wpis pojawi się w *package.json*



```
$ npm install lodash
```

```
# zawartość package.json
```

```
{  
  "name": "my-project",  
  "version": "1.0.0",  
  "main": "index.js",  
  "author": "Michał Michałczuk <michalczukm@gmail.com> (https://michalczukm.xyz)",  
  "license": "MIT",  
  "dependencies": {  
    "lodash": "^4.17.15"  
  }  
}
```

“dependencies” zawiera
wszystkie zależności, których
używamy w **kodzie
produkcyjnym**

npm: skrypty

- komendy które możemy wykonać
- format **npm run nazwa_komendy**
- Standard:
 - npm start
 - npm test
- Własne polecenia
 - npm run nazwa_komendy



```
$ npm install lodash
```

```
# zawartość package.json
```

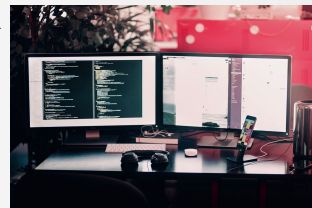
```
{  
  "name": "my-project",  
  "version": "1.0.0",  
  "main": "index.js",  
  "scripts": {  
    "start": "node index.js",  
    "custom-script": "node some-file.js"  
  },  
  "author": "Michał Michalczuk <michalczukm@gmail.com> (https://michalczukm.xyz)",  
  "license": "MIT",  
  "dependencies": {  
    "lodash": "^4.17.15"  
  }  
}
```

Do uruchomienia skryptu
“standardowego” - “start”,
wystarczy:
npm start

Aby uruchomić customowy
skrypt “custom-script” zwołaj:
npm run custom-script

Ćwiczenie 3.

1. Stwórz nowy pakiet (*npm init*)
2. Stwórz nowy plik .js `my-file.js`
 - Wyświetl w nim na konsoli ciąg znaków od 1 do 10 (1,2,3,...10)
3. Stwórz nowy plik .js `index.js`
 - Wyświetl w nim na konsoli "Hello!"
4. Uzupełnij plik `package.json` aby poniższe działało:
 - **npm start** powinno wykonać plik `index.js`
 - **npm run my-script** powinno wykonać plik `my-file.js`



npm: gdzie są pliki?

- **node_modules**

- Wszystkie zależności które zainstalujemy **lokalnie** czyli w naszym projekcie, są tam przechowywane.
- Nie trzymamy tego katalogu na repo.
 - nie nasz kod
 - ciężki

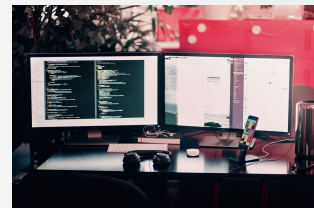


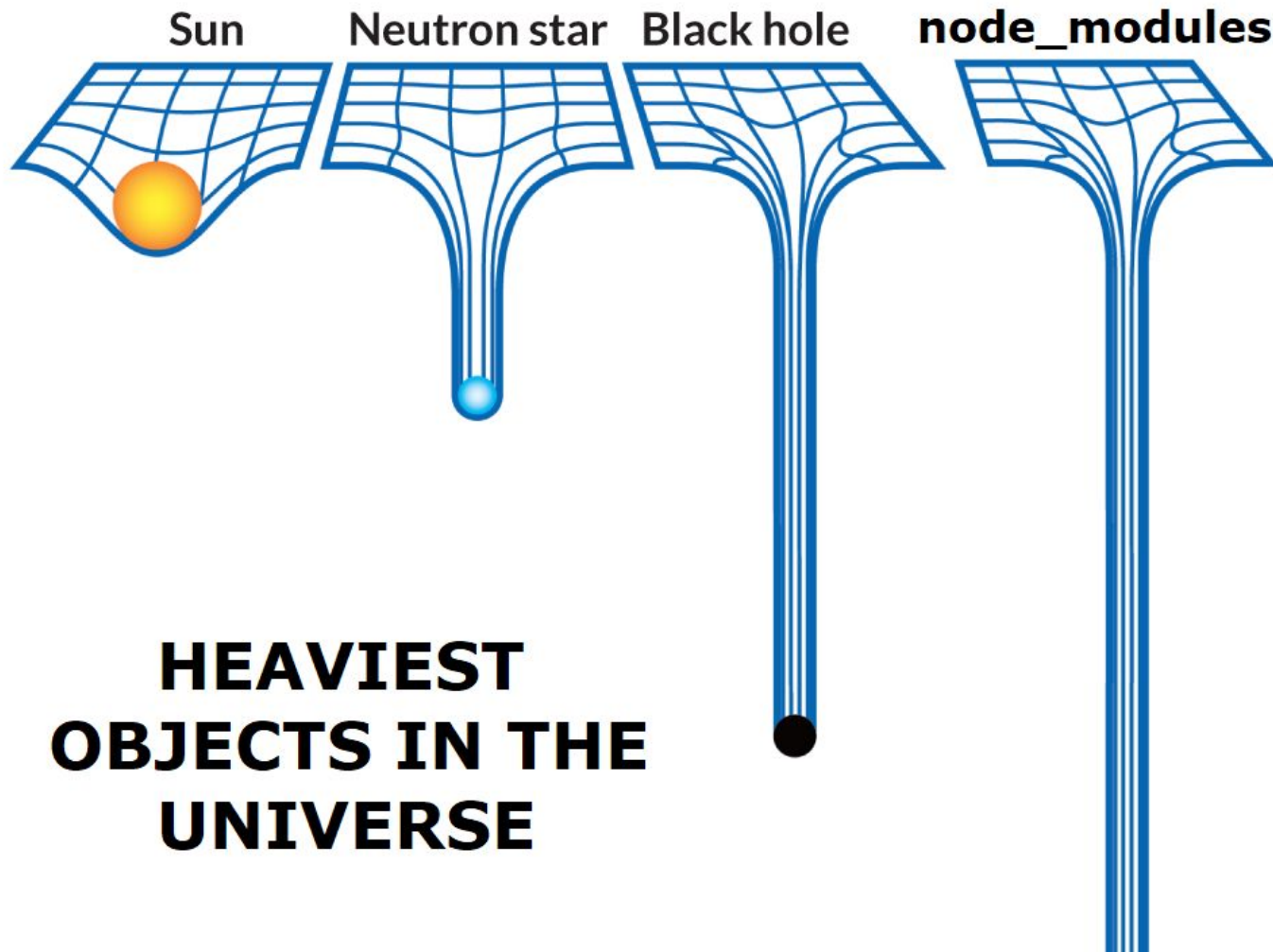
- **npm install**

- instaluje lokalnie pakiety (wrzuca do **node_modules**) czytając **`dependencies`** z naszego **`package.json`**

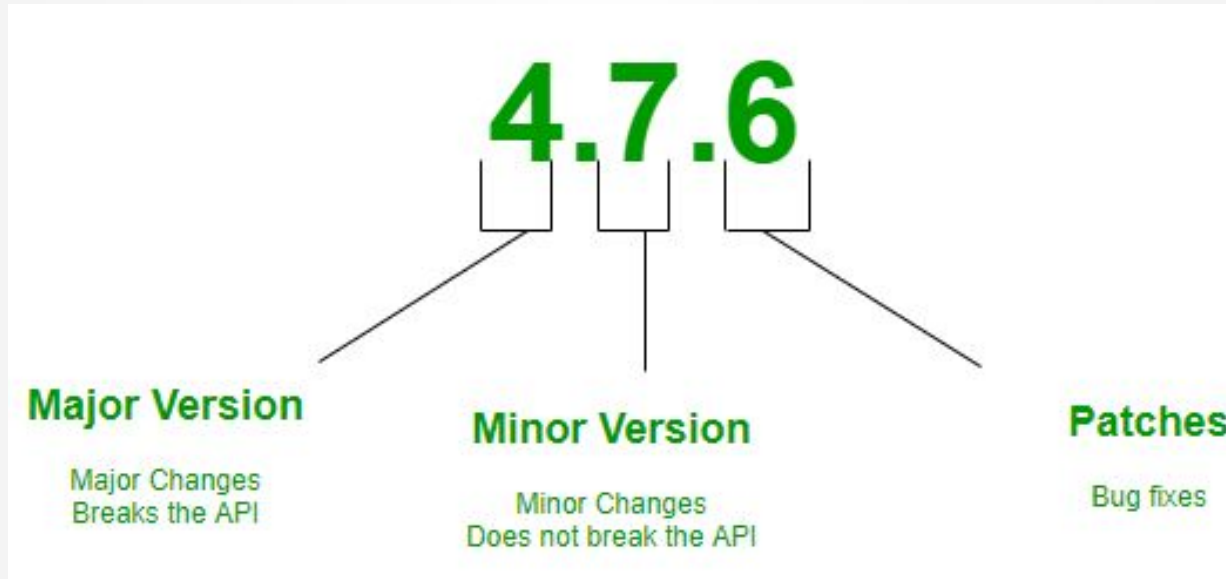
Ćwiczenie 4.

1. Stwórz nowy pakiet (*npm init*)
2. Zainstaluj pakie **lodash** (npmjs.com/package/lodash)
3. Stwórz nowy plik .html `index.html`
 - Wyświetl w nim na konsoli ciąg znaków od 1 do 10 (1,2,3,...10), tym razem za pomocą lodash'a (lodash.com/docs/4.17.15#range)
4. **Pamiętaj** o dodaniu zależności w html jako `<script src="..."`!





semantic versioning



semantic versioning (semver)

Pakiety w npm są wersjonowane wg semver

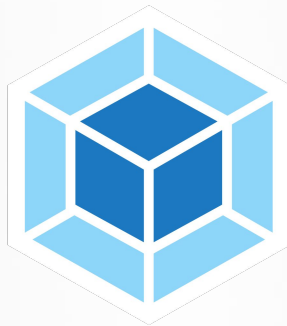
Sprawdź!

<https://semver.npmjs.com/>





webpack

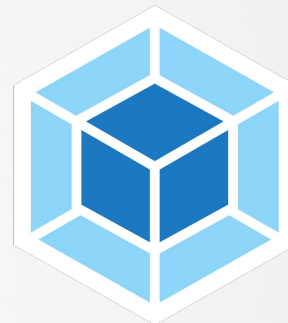


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <script src="./node_modules/lodash/lodash.min.js"></script>
  <script src="./1.js"></script>
  <script src="./2.js"></script>
  <script src="./another.js"></script>
  <script src="./index.js"></script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <script src="./bundle.js"></script>
</body>
</html>
```

webpack

- To narzędzie do budowania bundli
 - **bundle** = plik stworzony z połączenia powiązanych rzeczy
- Konfiguruje się go przez reguły i loadery
 - podajemy im regex jakie pliki (o jakich nazwach) ma analizować
 - podajemy funkcje, która potrafi zbudować bundla, lub przetworzyć zawartość



```
// webpack.config.js

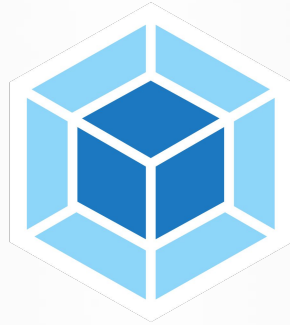
const path = require('path');

module.exports = {
  entry: './index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          { loader: 'style-loader' },
          { loader: 'css-loader' }
        ],
      },
    ],
  },
}
```

```
// package.json

{
  "name": "my-package",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "webpack --mode development --watch",
    "build": "webpack --mode production"
  },
  "dependencies": {
    "css-loader": "^3.2.0",
    "lodash": "^4.17.15"
  },
  "devDependencies": {
    "style-loader": "^1.0.0",
    "webpack": "^4.41.2",
    "webpack-cli": "^3.3.10",
    "webpack-dev-server": "^3.9.0"
  }
}
```

webpack



webpack.js.org



babel



babel

- Transpiler do JavaScript'u
 - np. ES6 -> ES5
 - piszemy kod w nowym standardzie -
uruchamiamy na starych przeglądarkach
- Możemy go użyć w loaderze do webpack'a
- Na żywo: babeljs.io/repl



BABEL

```
// webpack.config.js

// webpack.config.js
const path = require('path');

module.exports = {
  entry: './index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader'
        }
      },
      {...},
    ]
  }
}
```

```
// package.json
{
  "name": "my-package",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "webpack --mode development --watch",
    "build": "webpack --mode production"
  },
  "dependencies": {
    "css-loader": "^3.2.0",
    "lodash": "^4.17.15"
  },
  "devDependencies": {
    "@babel/core": "^7.7.2",
    "@babel/preset-env": "^7.7.1",
    "babel-loader": "^8.0.6",
    "style-loader": "^1.0.0",
    "webpack": "^4.41.2",
    "webpack-cli": "^3.3.10",
    "webpack-dev-server": "^3.9.0"
  }
}
```

```
// webpack.config.js

// webpack.config.js
const path = require('path');

module.exports = {
  entry: './index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader'
        }
      },
      {...},
    ]
  }
}
```

```
// package.json
{
  "name": "my-package",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "webpack --mode development --watch",
    "build": "webpack --mode production"
  },
  "dependencies": {
    "css-loader": "^3.2.0",
    "lodash": "^4.17.15"
  },
  "devDependencies": {
    "@babel/core": "^7.7.2",
    "@babel/preset-env": "^7.7.1",
    "babel-loader": "^8.0.6",
    "style-loader": "^1.0.0",
    "webpack": "^4.41.2",
    "webpack-cli": "^3.3.10",
    "webpack-dev-server": "^3.9.0"
  }
}
```



ECMAScript

JS

**TC
39**

ES

ECMAScript

- specyfikacja języka skryptowego, określana przez Ecma International w odpowiednim standardzie
- Została stworzona aby ustandaryzować JavaScript
- Każdy silnik JavaScript = inna implementacja JavaScript, stąd mocna potrzeba standaryzacji
- Aktualnie co roku jest wydawana nowa wersja
- tc39 - komitet techniczny ECMAScript



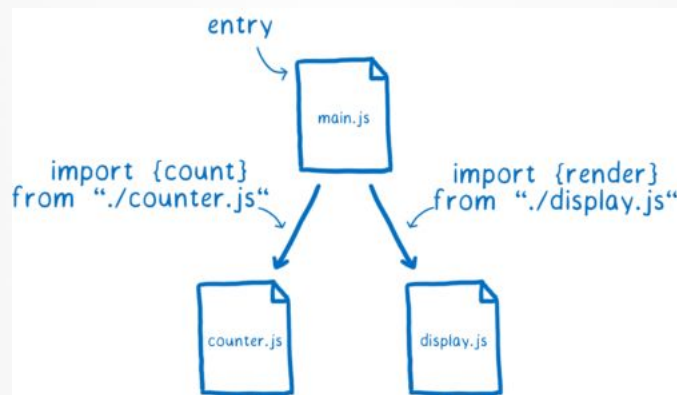
ES

5.1	June 2011		This edition 5.1 of the ECMAScript standard is fully aligned with third edition of the international standard ISO/IEC 16262:2011.	Pratap Lakshman, Allen Wirfs-Brock
6	June 2015 ^[11]	ECMAScript 2015 (ES2015)	See 6th Edition - ECMAScript 2015	Allen Wirfs-Brock
7	June 2016 ^[12]	ECMAScript 2016 (ES2016)	See 7th Edition - ECMAScript 2016	Brian Terlson
8	June 2017 ^[13]	ECMAScript 2017 (ES2017)	See 8th Edition - ECMAScript 2017	Brian Terlson
9	June 2018 ^[14]	ECMAScript 2018 (ES2018)	See 9th Edition - ECMAScript 2018	Brian Terlson
10	June 2019 ^[9]	ECMAScript 2019 (ES2019)	See 10th Edition - ECMAScript 2019	Brian Terlson, Bradley Farias, Jordan Harband

Scripting engine ⇅	Reference application(s) ⇅	Conformance ^[41]			
		ES5 ^[42] ⇅	ES6 ^[43] ⇅	ES7 ^[44] ⇅	Newer (2016+) ^{[44][45]} ⇅
Chakra	Microsoft Edge 18	100%	96%	100%	48%
SpiderMonkey	Firefox 67	100%	98%	100%	83%
Chrome V8	Google Chrome 75, Opera 62	100%	98%	100%	98%
JavaScriptCore (Nitro)	Safari 12.1	99%	99%	100%	87%

ECMAScript

moduł **ES**



Moduły w JS: historycznie

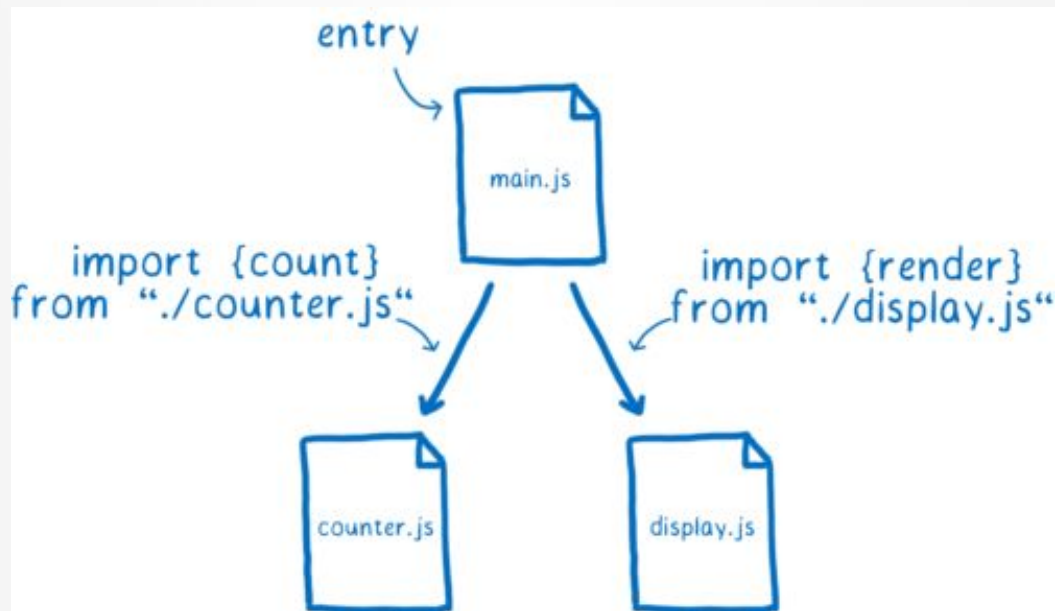
CommonJS

UMD

AMD

ES6 modules

Moduły ES



plik = moduł

export - wystaw do
zaimportowania

import - zaimportuj
moduł

nic nie jest dodawane do
global scope

Module ES: names exports

```
//plik foo.js  
  
export const MAX_CLIENTS = 10;  
  
export const getStuffDone = (stuff) => { /* ... */ };  
  
export class Stuff { }
```

```
//plik index.js  
  
import { getStuffDone } from './foo';  
  
getStuffDone('stuff to do');
```

Module ES: default export

```
// plik foo.js  
export default (stuff) => { /* ... */ };
```

```
// plik index.js  
import getStuffDone from './foo';  
  
getStuffDone('stuff to do');
```


Moduły ES: oba naraz

```
// plik foo.js  
  
export const MAX_CLIENTS = 10;  
  
export class Stuff { }  
  
export default (stuff) => { /* ... */ };
```

```
// plik index.js  
  
import getStuffDone, { MAX_CLIENTS, Stuff } from './foo';  
  
getStuffDone('stuff to do');  
  
new Stuff();
```

Module ES: named imports 1/2

```
// plik foo.js  
export const MAX_CLIENTS = 10;  
  
export const getStuffDone = (stuff) => { /* ... */ };  
  
export class Stuff { }
```

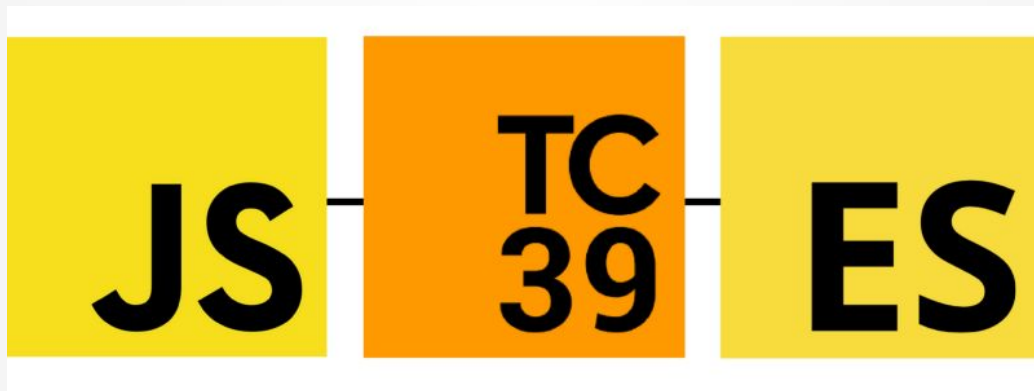
```
// plik index.js  
import { Stuff as LocalStuff } from './foo';  
  
new LocalStuff();
```

Module ES: named imports 2/2

```
// plik foo.js  
export const MAX_CLIENTS = 10;  
  
export const getStuffDone = (stuff) => { /* ... */ };  
  
export class Stuff { }
```

```
// plik index.js  
import * as foo from './foo';  
  
foo.getStuffDone();
```

Inne ważne rzeczy z nowych ES



ES6 (ECMAScript 2015)

Dużo zmian w stosunku do ES5

Najważniejsze:

- let, const
- template strings
- arrow functions
- class
- deconstructing
- spread / rest operator
- modules



ES

ES6 (ECMAScript 2015)

Dużo zmian w stosunku do ES5

Najważniejsze:

- **let, const**
- template strings
- arrow functions
- class
- deconstructing
- spread / rest operator
- modules



ES

let: zakres



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

```
console.log('i = ', i)
```

```
// 1  
// 2  
// 3  
// i = 4
```



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

```
console.log('i = ', i)
```

```
// 1  
// 2  
// 3  
// ReferenceError: i is not  
// defined
```

let: tylko jedna deklaracja



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

```
console.log(name)
```

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



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

```
console.log(name)
```

```
// John
```


let: brak hoistingu



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

```
f()
```

```
// 2
```



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

```
f()
```

```
// ReferenceError: Cannot access 'x'  
// before initialization
```

const

Działa tak samo jak **let**, z wyjątkiem:

- należy wykonać przypisanie do zmiennej podczas deklaracji
- nie można potem przypisać ponownie
- co nie oznacza, że wartość nie może się zmienić (jeśli jest obiektem)

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 (ECMAScript 2015)

Dużo zmian w stosunku do ES5

Najważniejsze:

- let, const
- **template strings**
- arrow functions
- class
- deconstructing
- spread / rest operator
- modules

A yellow square containing the letters 'ES' in a bold, black, sans-serif font.

template strings



// ES5

```
var name = "Mateusz"
```

```
var message = "Hello " + name
```

```
console.log(message)
```

// Hello Mateusz



// ES6

```
const name = 'Chris'
```

```
const message = `Hello ${name}`
```

```
console.log(message)
```

// Hello Chris

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 (ECMAScript 2015)

Dużo zmian w stosunku do ES5

Najważniejsze:

- let, const
- template strings
- **arrow functions**
- class
- deconstructing
- spread / rest operator
- modules



ES

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
```



arrow functions: skrócony zapis



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

arrow functions: lexical scope this



```
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
```

arrow functions: podsumowanie

- arrow function mają leksykalny **this** - jest ustalany w **momencie deklaracji**, nie w momencie wykonania
- arrow function może mieć niejawny **return**, jeśli mamy tylko jedno wyrażenie
- możemy pominąć **()** przy parametrach, jeśli mamy tylko jeden
- arrow function nie mogą być konstruktorami
- arrow function nie mają property **arguments**
- jeśli chcemy zwrócić obiekt bez używania **return**, musimy go opakować w **{ ... }**

ES6 (ECMAScript 2015)

Dużo zmian w stosunku do ES5

Najważniejsze:

- let, const
- template strings
- arrow functions
- class
- **deconstructing**
- spread / rest operator
- modules

A yellow square containing the letters 'ES' in a bold, black, sans-serif font, representing ECMAScript.

deconstructing: obiekty

```


// 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}`
)
// Bob Builder

```

deconstructing: obiekty



```
// NAMING PARAMETERS DIFFERENTLY
```

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

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

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

deconstructing: tablice



// 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)
```

// 10 20 30

deconstructing: tablice



// 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 (ECMAScript 2015)

Dużo zmian w stosunku do ES5

Najważniejsze:

- let, const
- template strings
- arrow functions
- **class**
- deconstructing
- spread / rest operator
- modules



ES

class



```
class Animal {  
  // object constructor  
  // (executed when new Animal())  
  constructor(sound) {  
    this.sound = sound  
  }  
  // class method  
  makeSound() {  
    console.log(this.sound)  
  }  
}
```

class to tylko “*syntactic sugar*”, mechanizm działania to dalej **prototypy i funkcje**

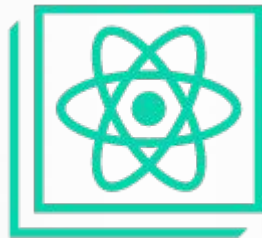
class: dziedziczenie



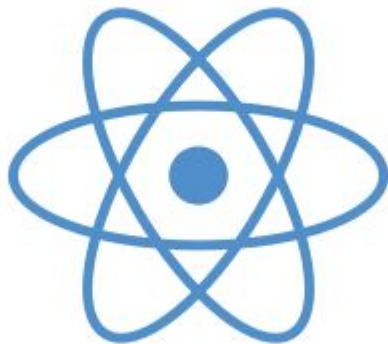
```
class Dog extends Animal {  
  constructor(breed) {  
    // executes Animal's constructor super('woof')  
    // you must do it first  
    super('woof')  
    this.breed = breed  
  }  
}
```

ES5 way - <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance>

 create-react-app



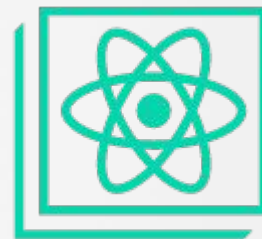
Create React App



Official. No Setup. Minimal.

create-react-app

- paczka npm
- tworzy całą strukturę projektu
- zapewnia gotowe ustawienia
 - babel
 - webpack
- oficjalny tool od facebook'a



create-react-app.dev

github.com/facebook/create-react-app

create-react-app

```
# instalacja CRA // wykonaj przed
```

```
npm i -g create-react-app
```

```
# użycie z npm init
```

```
npm init react-app <nazwa-apki>
```

```
# przez CRA
```

```
create-react-app <nazwa-apki>
```


create-react-app: npm scripts

```
// package.json
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

create-react-app: struktura aplikacji

Aplikacja przychodzi z konkretną zalecaną strukturą:

- **build** - tam wyląduje dystrybucja, nie chcemy wersjonować
- **node_modules** - jak zwykle, tutaj są nasze pakiety npm
- **public** - zasoby jak pliki .html, obrazki etc
- **src** - nasz kod który ma być ogarnięty przez webpack
- **package.json**
- **.gitignore** - gotowy plik .gitignore
- **README.md** - warto zajrzeć do tego readme, jest tam masa dokumentacji. A docelowo - powinna być to nasza własna

Dziękuję za uwagę

Michał Michalczuk

michalczukm.xyz



michalczukm



michalczukm@gmail.com

