
PGS
SOFTWARE

PGS SOFTWARE

Akademia Developera – edycja FrontDev

ECMAScript 6, JavaScript build tools

JÓZEF TOKARSKI **FULL-STACK JAVA/WEB DEVELOPER**

Ewolucja Javascript'u

ECMAScript 3th Edition: grudzień 1999

ECMAScript 4th Edition (abandoned): 2007-2008

ECMAScript 5th Edition: grudzień 2009

ECMAScript 6th Edition: czerwiec 2015

ECMAScript 7th Edition: czerwiec 2016

ECMAScript 8th Edition: czerwiec 2017

ES.Next

ECMAScript 6th Edition - inne określenia

ECMAScript 2015

ECMAScript Harmony

ES6 - stałe

Słowo kluczowe `const`

```
const MIN_PASSWORD_LENGTH = 10;
```

obiekty są nadal mutowalne

```
const search = {
    term: 'zoo',
    region: 'PL'
};

search.term = "marina";
```

stałe to inaczej **niemutowalne referencje (lub wartości)**

```
function handleClick() {
    const searchInput = document
        .getElementById('search-main');
    const searchTerm = searchInput.value;
    const request = new XMLHttpRequest();
    ...
}
```

ES6 - domyślne wartości parametrów

domyślne wartości dla parametrów funkcji

```
function calc(x = 1, y = 20, z = 50) {
    console.log(x, y, z);
}

calc(5);

// 5 20 50
```

rest parameter

pozwala wywoływać funkcję ze zmienną liczbą argumentów

```
function grant(item, ...permissions) {
    console.log('Granted: ' + item +
        '(' + permissions.join(', ') + ')'
    );
}

grant('orders', 'create', 'read', 'update');

// Granted: orders(create, read, update)
```

spread operator

pozwala rozdystrybuować elementy kolekcji

```
var defaultPermissions = ['create', 'read'];
var myPermissions =
    [...defaultPermissions, 'update', 'delete'];

// ["create", "read", "update", "delete"]
```

Zmienne na poziomie bloku

- słowo kluczowe `let`
- istnieją tylko w obrębie bloku, nie są windowane

```
function foo() {
    let x = 5;
    if (true) {
        let x = "stringX";
        console.log(x);
    }
    console.log(x);
}
foo();
// stringX
// 5
```

Funkcje na poziomie bloku

nie są windowane

```
(function() {
    function f1() { console.log("F1"); }
    if (true) {
        function f1() { console.log("F2"); }
        f1();
    }
    f1();
}());
// F2
// F1
```

Arrow functions

pozwalają skrócić zapis funkcji anonimowej

```
const a = [10, 11, 12];
const action1 = v => console.log(v);
const action2 =
    (v, i) => console.log(v + ": " + i);

a.forEach(action1);
a.forEach(action2);
```

nie mają swojego kontekstu wykonania (domykają `this`)

```
function f1() {
    window.setTimeout(function() {
        console.log(this);
    }, 0);
}

function f2() {
    window.setTimeout(
        () => console.log(this), 0
    );
}

f1.apply({'a': 15});
f2.apply({'a': 15});
```

pozwalają pominąć słowo kluczowe `return`

```
const sum = (a,b) => a + b;
console.log(sum(10, 40));
```

Template Literals

wygodna **interpolacja Stringów**, również wielolinijkowych

```
const user = {
    'firstname': 'Jan',
    'lastname': 'Kowalski',
    'born': 1964
};
const fullData = `
    Name: ${user.firstname} ${user.lastname}
    Age: ${(new Date().getFullYear()) - user.born}
`;
console.log(fullData);
```

Uproszczenia literału obiektu

skrót propercji

wyliczane klucze

propercje-metody

```
const a = 222, b = 444;
const prefix = 'primary', key = 'Prop';

const obj = {
    a, b,
    [prefix + key]: 888,
    sum() {      return this.a + this.b; }
}

console.log(obj);
```

destructuring assignment

array matching

```
let arr = [15, 16, 17];
let [x, y, z] = arr;

console.log(y);
```

object matching

```
let user = {
    'name': 'John',
    'age': 49
};

function f1({name, age, active = true}) {
    console.log(name, age, active);
}

f1(user);
```

Klasy

definicje klas

```
class Shape {
    constructor (id, x, y) {
        this.id = id;
        this.move(x, y);
    }
    move (x, y) {
        this.x = x;
        this.y = y;
    }
}

console.log(new Shape(1, 20, 40));
```

dziedziczenie

```
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y);
        this._width = width;
        this._height = height
    }
    get area() {
        return this._width * this._height;
    }
}

const r = new Rectangle(1, 0, 0, 5, 9);
console.log(r.area);
```

Moduły

export/import

```
// math.js
function sum(a,b) { return a+b; }
export { sum };
export const PI = 3.14;
export default (x) => x*x;

// calc1.js
import * as math from "math";
math.sum(4, 5);

// calc2.js
import {PI, sum} from "math";
sum(3, PI);

// calc3.js
```

JavaScript Build Tools

- weryfikacja
- transpilacja
- konkatenacja
- minifikacja

NodeJS Shell

silnik JavaScriptu "wyciągnięty" z Chrome'a i użyty na konsoli

Pozwala wykonywać kod w JS bez użycia przeglądarki

Kod ma dostęp do zasobów systemu operacyjnego

NPM, NodeJS package manager

domyślny manager pakietów dla NodeJS

pozwala zarządzać zależnościami projektu przez pliik `package.json`

NPM to także repozytorium bibliotek on-line

Grunt, The JavaScript Task Runner

wzorowany na narzędziu **cmake**

pierwszy popularny build tool dla JavaScriptu

duży zbiór pluginów

procesowanie plików oparte o pliki tymczasowe - problemy performance'owe

Gulp

zasada działania podobna do Grunt

procesowanie oparte o wirtualną reprezentację plików (VinylJS)

duży zbiór pluginów

Webpack

Nieco inna zasada działania: **bundler** a nie **task runner**

każdy plik traktuje jako moduł

buduje graf zależności i łaczy wszystkie niezbędne moduły w **bundle**

bardzo dobrze wspiera moduły ES6