Nowe wersje języka JavaScript

ES6(ES2015) ES7(ES2016) ES8(ES2017)

const/let

```
var x = 1;
function f(cond) {
  if (cond) {
    var x = 2;
    return x;
  }
  return x;
}
f(false);
```

```
let x = 1;
function f(cond) {
  if (cond) {
    let x = 2;
    return x;
  }
  return x;
}
f(false);
```

- Nie ma potrzeby korzystania z IIFE

```
(function () {
  var tmp = ···;
  ···
}());
console.log(tmp);
```

```
{
  let tmp = ···;
  ···
}
console.log(tmp);
```

Operacje na ciągach znaków

```
function witaj(kogo) {
  console.log('Hello '+kogo+'!');
}
```

```
function witaj(kogo) {
  console.log(`Hello ${kogo}!`);
}
```

```
var dokument = '\
  <!doctype html>\n\
  <html>\n\
  <head>\n\
    <meta charset="UTF-8">\n\
  </head>\n\
  <body>\n\
  </body>\n\
  </html>';
```

```
const dokument = `
  <!doctype html>
  <html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
  </body>
  </html>`;
```

- Nowe funkcje
  - startsWith, endsWith, includes, repeat

...

```
function f() {
  for (var i in arguments) {
    console.log(arguments[i]);
  }
}
```

```
function f(...args) {
  for (const arg of args) {
    console.log(arg);
  }
}
```

```
function f(x) {
  var args = [].slice.call(arguments, 1);
  ...
}
```

```
function f(x, ...args) {
  ...
}
```

```
var t1 = ['a', 'b'];
var t2 = ['c', 'd'];
var t3 = ['e', 'f'];

console.log(t1.concat(t2, t3));
```

```
const t1 = ['a', 'b'];
const t2 = ['c', 'd'];
const t3 = ['e', 'f'];

console.log([...t1, ...t2, ...t3]);
```

Funkcje ze strzałką

- Skrócony zapis anonimowej funkcji

```
(param1, param2, ..., paramN) => { instrukcje }
(param1, param2, ..., paramN) => wyrażenie

(singleParam) => { instrukcje }
singleParam => { instrukcje }
singleParam => wyrażenie

() => { instrukcje }

(...args) => {
  for (const e of args) {
    console.log(e);
  }
}
```

```
var t = [1, 2, 3];
var t2 = t.map(function (x) {
  return x * x;
});
```

```
const t = [1, 2, 3];
const t2 = t.map(x => x * x);
```

Operacje na tablicach

```
const t = ['a', NaN];
t.findIndex(x => Number.isNaN(x));

const t2 = Array.from(arguments);

const t3 = new Array(10).fill(undefined);
```

Zwracanie wielu wartości

- Forma obiektu

```
f = () => { return { x:1, y:2, z:3 }; };

const {x, z} = f();
```

- Forma tablicy

```
f = () => { return [ 1, 2, 3 ]; };

const [, y, z] = f();
```

Pętla for-of

```
const t = [5, 10, 15];

for (var i = 0; i < t.length; i++) {
  console.log(t[i]);
}

for (var i in t) {
  console.log(t[i]);
}

t.forEach((e) => { console.log(e); });

for (const e of t) {
  console.log(e);
}

for (const [i, e] of t.entries()) {
  console.log(`t[${i}] = ${e}`);
}
```

Domyślne wartości parametrów

```
function f(x, y) {
  x = x || 0;
  y = y || 0;
  ...
}
```

```
function f(x=0, y=0) {
  ...
}
```

Klasy

```
function Osoba(imię, nazwisko) {
  this.imię = imię;
  this.nazwisko = nazwisko;
}
Osoba.prototype.przedstawSię = function () {
  console.log('Nazywam się ' + this.imię + ' ' + this.nazwisko + '.');
};
```

```
class Osoba {
  constructor(imię, nazwisko) {
    this.imię = imię;
    this.nazwisko = nazwisko;
  }
  przedstawSię() {
    console.log(`Nazywam się ${this.imię} ${this.nazwisko}.`);
  }
}
```

Dziedziczenie

```
class Osoba {
  constructor(imię, nazwisko) {
    this.imię = imię;
    this.nazwisko = nazwisko;
  }
  przedstawSię() {
    console.log(`Nazywam się ${this.imię} ${this.nazwisko}.`);
  }
}

class Student extends Osoba {
  constructor(imię, nazwisko, nrIndeksu) {
    super(imię, nazwisko);
    this.nrIndeksu = nrIndeksu;
  }
  przedstawSię() {
    super.przedstawSię();
    console.log(`Mój numer indeksu to ${this.nrIndeksu}.`);
  }
}
```

Obietnice

- Alternatywa dla metod typu callback
- Bez obietnic

```
console.log("Czekamy, aż coś się skończy...");
setTimeout(function () {
  console.log("Skończyło się. Teraz możemy zrobić rzecz nr 2.");
  console.log("Czekamy, aż coś się skończy...");
  setTimeout(function () {
    console.log("Skończyło się. Teraz możemy zrobić rzecz nr 3.");
    console.log("Czekamy, aż coś się skończy...");
    setTimeout(function () {
      console.log("Skończyło się. Teraz możemy lecieć dalej.");
    }, 5000);
  }, 5000);
}, 5000);
```

Obietnice

- Alternatywa dla metod typu callback
- Z obietnicami

```
const work = time => new Promise((resolve) => {
  console.log("Czekamy, aż coś się skończy...");
  setTimeout(resolve, time);
});

work(5000)
  .then(res => {
    console.log("Skończyło się. Teraz możemy zrobić rzecz nr 2.");
    console.log("Czekamy, aż coś się skończy...");
    return work(5000);
  })
  .then(() => {
    console.log("Skończyło się. Teraz możemy zrobić rzecz nr 3.");
    return work(5000);
  })
  .then(() => {
    console.log("Skończyło się. Teraz możemy lecięć dalej.");
  })
```

Obietnice

```
function asyncFunc() {
  return new Promise(function (resolve, reject) {
    ...
    resolve(result);
    ...
    reject(error);
  });
}

asyncFunc()
  .then(result => { ··· })
  .catch(error => { ··· });

//lub

asyncFunc()
  .then(
    result => { ··· },
    error => { ... }
  );
```

Obietnice

- Możliwe stany obietnicy
  - Pending
  - Fulfilled
  - Rejected

Przydatne materiały

- http://speakingjs.com/es5/
- http://exploringjs.com/es6/index.html
- http://exploringjs.com/es2016-es2017/
- https://developers.google.com/web/fundamentals/primers/promises
- https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-promise-27fc71e77261