ES 2015, historie a transpilace

- ES6 = formálně ECMAScript 2015
  - nejnovější standardizovaná iterace jazyka
  - dříve označována jako Harmony

Let, const

- **let** = var + block scope
- **const** = let + read-only

```
1. const N = 8;
2. N = 4; // exception
3.
4. let x = 1;
5. if (true) {
6.     let x = 2;
7. }
8. alert(x); // 1
```

Arrow functions

- zkrácená syntaxe definice funkcí
- lexical this (nelze call, apply, new)
- pokud má tělo funkce jediný příkaz, není třeba return ani závorky

```
1. var square = a => a*a;
2. var add = (a, b) => a+b;
3.
4. // lexical this
5. setTimeout( () => this.doStuff(), 1000 );
```

Enhanced object literals

- zkrácená definice objektů

```
1. var x = 42;
2.
3. var obj = {
4.     x,                    // "x":42
5.     y() { return x; },
6.     ["data_" + x]: x      // "data_42":42
7. }
```

Template string literals

- nahrazování řetězců
- odpadá nutnost „sčítání“
- smí obsahovat newline
- možnost vlastní interpolační funkce

```
var x = "world";
var y = `hello ${x}`;
var z = `this is a
            very long string`;

// html je uživ. funkce, která dostane jednotlivé tokeny k naformátování
html`<div> ${unsafe} </div>`;
```

Destructuring

- snažší přístup k vlastnostem struktur a polí

```
var [a, b, c] = [1, 2, 3];

var f = function() { return {x:42}; }
var { x } = f();
```

Default + Rest + Spread

- výchozí hodnoty parametrů
- převod (podmnožiny) parametrů na pole a zpět

```
var f = function(x, y = 12) {
    return x + y;
}
f(10); // 22

var f = function(x, ...y) {
    alert(y.length);
}
f(1, 2, 3); // 2

var f = function(a, b, c) { return c; }
f(...[1, 2, 3]); // 3
```

Classes

- nová syntaxe, staré chování (stále se jedná o prototypovou dědičnost)

```
class B extends A {
    constructor(x) {
        super(); // v konstruktoru dědící třídy povinné; před ním neexistuje this
        this.x = x;
    }

    get something() { /* .... */ }

    f1() {
        super.f1();
        return this.x;
    }

    static f2() {}
}
```

Modules

- modularizace na syntaktické úrovni
- jeden výchozí a libovolně dalších pojmenovaných exportů

```
// a.js
export var A = function() {};
export default function() {};

// b.js
import { A } from "./a.js";
A();

import myLocalName from "./a.js"; // default
```

{Weak,}{Map,Set}

- Set = množina unikátních hodnot
- Map = dvojice cokoliv-cokoliv
- WeakMap, WeakSet = bez reference na objekt, bez iterovatelnosti

```
var s = new Set();
s.add("hello").add("goodbye").add("hello");
s.size == 2;
s.has("hello") == true;

var m = new Map();
m.set("hello", 42);
m.set(s, 34);
m.get(s) == 34;
```

Symbols

- nový datový typ pro řízení přístupu
- není zcela privátní, ale alespoň je unikátní

```
(function() {
    var moneyKey = Symbol("money");
    typeof(moneyKey) == "symbol";

    var Person = function() {
        this[moneyKey] = 10000;
    }

    var person = new Person();
    person.money == undefined;

    Object.getOwnPropertySymbols(person); // :-(
})();
```

Iterators + for..of

- programovatelná iterovatelnost
- cokoliv, co má metodu „**next**“ je iterátor
- cokoliv, co má symbol **Symbol.iterator**, je iterovatelné cyklem for..of

```
let fibonacci = {
    [Symbol.iterator]() {
        let pre = 0, cur = 1;
        return {
            next() {
                [pre, cur] = [cur, pre + cur];
                return { done: false, value: cur }
            }
        }
    }
}

for (var n of fibonacci) {
  if (n > 1000) break;
  console.log(n);
}
```

Generators

- speciální druh funkce
- návratovou hodnotou je iterátor
- klíčové slovo „**yield**“ odpovídá přerušení po jedné iteraci

```
var generator = function*() {
    var tmp = 1;
    while (true) {
        tmp *= 3;
        yield tmp;
    }
}

var iterator = generator();
iterator.next().value; // 3, next() vrací i done:true/false
iterator.next().value; // 9
iterator.next().value; // 27
```

Proxies

- monitorování libovolného přístupu k objektům
- čtení, zápis, volání,…

```
var obj = {};

var interceptor = {
    get: function (receiver, name) {
        return `Hello, ${name}!`;
    }
};

var p = new Proxy(obj, interceptor);
p.world === "Hello, world!"
```

Reflect

- rozhraní pro introspekci objektů
- metody podobné těm v **Object.***
- namísto výjimek vrací false

```
Reflect.defineProperty(obj, name, descriptor);
Reflect.construct(F, args);
Reflect.get(obj, property, thisForGetter);
/* ... */
```

Rozšíření ES5

```
Number.EPSILON
Number.MAX_SAFE_INTEGER
Number.MIN_SAFE_INTEGER
Number.isInteger(Infinity)                    // false
Number.isNaN("NaN")                           // false

Math.acosh(3)                                 // 1.762747174039086
Math.hypot(3, 4)                              // 5
Math.imul(Math.pow(2, 32)-1, Math.pow(2, 32)-2) // 2
Math.sign(5)                                  // 1
Math.trunc(3.1)                               // 3
/* ... */

"abc".repeat(3)                               // "abcabcabc"
```

```
Array.from(document.querySelectorAll("*")) // real Array
Array.of(1, 2, 3)                          // without special one-arg behavior
[0, 0, 0].fill(7, 1)                       // [0, 7, 7]
[1, 2, 3].find(x => x == 3)                // 3
[1, 2, 3].findIndex(x => x == 2)           // 1
[1, 2, 3, 4, 5].copyWithin(3, 0)           // [1, 2, 3, 1, 2]
["a", "b", "c"].entries()                  // iterator [0, "a"], [1,"b"], [2,"c"]
["a", "b", "c"].keys()                     // iterator 0, 1, 2
["a", "b", "c"].values()                   // iterator "a", "b", "c"

Object.assign(target, { source: "data" });
```

- dále:
  - práce s Unicode znaky mimo BMP (code points > 65535): 😂, 💩, 🤣, 🐃,…
  - subclassing věstavěných objektů (Array, Element,…)
  - new Promise((resolve, reject) => {}), Promise.all, Promise.race

Jak zkoušet ES 2015+?

- pro něco lze polyfill (Array.from, Promise,…)
- některou syntaxi lze transpilovat
- něco nelze vůbec (WeakMap, WeakSet, Proxy)

Transpilace ES 2015+

- proces konverze syntaxe ES 2015+ do ES5
- Babel, Google Traceur Compiler,…
- https://babeljs.io/repl/

- moduly v praxi:
    - implementováno jen v nejnovějších verzích prohlížečů
    - alternativa #1: bundling do jednoho souboru nástrojem Rollup
    - alternativa #2: transpilace do jiného (kompatibilnějšího) formátu modulů

Nové ES

ES 2016

- operátor **
- Array.prototype.includes

ES 2017

- async/await
- String.prototype.pad{Start, End}
- SharedArrayBuffer, Atomics
- funkcionální iterace objektů

ES 2018

- rest/spread pro objekty
- asynchronní iterace for-await-of
- Promise.prototype.finally
- nové schopnosti regulárních výrazů: lookbehind, named capture,…