
# ES6

## & The Future Of JS

Nenad Pečanac
Serengeti

# Contents

JavaCro'16

1) History
2) Tooling
3) Syntax
4) Functions
5) Classes
6) Collections
7) Modules
8) Promises
9) The Future

JavaCro'16

# 1) History

# History (1)

JavaScript was created in 1995 by Brendan Eich for Netscape.

It was developed as Mocha and released as LiveScript.

It was quickly renamed to JavaScript, as a marketing move due to Java's popularity.

New name caused a significant misunderstanding of the language.

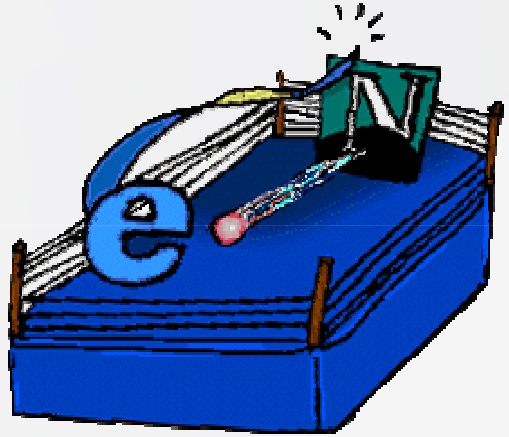
Language was created „in a week" and released incomplete.

Ambitious design influenced by Java, Perl, Scheme and Self.

# History (2)

Microsoft released JScript for IE 3.0 in 1996.

Browser Wars: CSS, DOM, Scripting, Dynamic HTML ...



Netscape delivered JavaScript to Ecma International for standardization in 1996.

ECMAScript was accepted as the compromised trade name.

# History (3)

JavaCro'16

| Release | Year | Changes |
|---|---|---|
| 1 | 1997 | First edition |
| 2 | 1998 | Editorial changes to keep the specification fully aligned with ISO/IEC 16262 international standard |
| 3 | 1999 | Added regular expressions, better string handling, new control statements, try/catch exception handling, tighter definition of errors, formatting for numeric output and other enhancements |
| 4 | - | Abandoned due to political differences concerning language complexity. |

# History (4)

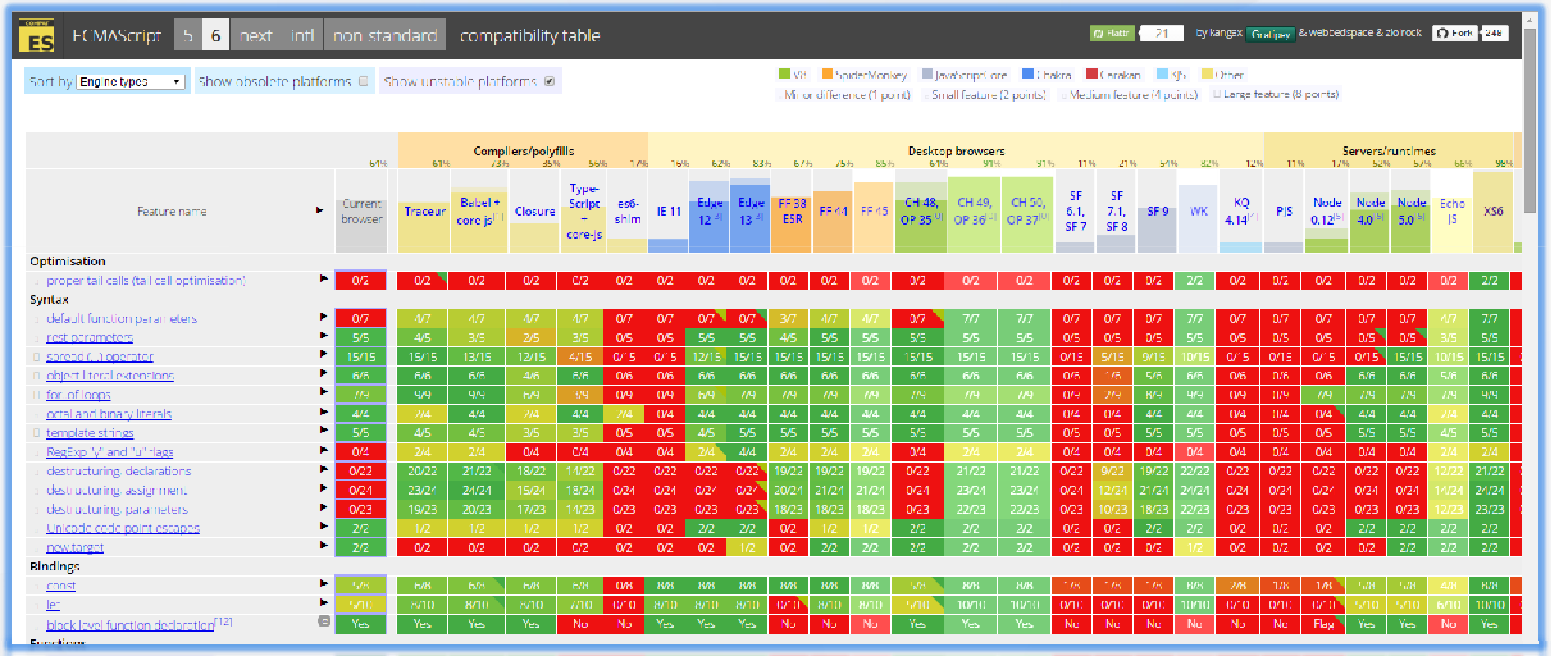
JavaCro'16

| Release | Year | Changes |
|---|---|---|
| 5 | 2009 | Adds strict mode, a subset intended to provide more thorough error checking, avoid error-prone constructs and clarifies many ambiguities. Adds some new features, such as getters and setters and library support for JSON. |
| 5.1 | 2011 | Aligned with third edition of the ISO/IEC 16262:2011. |
| 6 | 2015 | ECMAScript Harmony or ES6 Harmony. |
| 7 | ... | Work in progress. |

# Current browser support

JavaCro'16

http://kangax.github.io/compat-table/es6/



JavaCro'16

# 2) Tooling

# Tooling

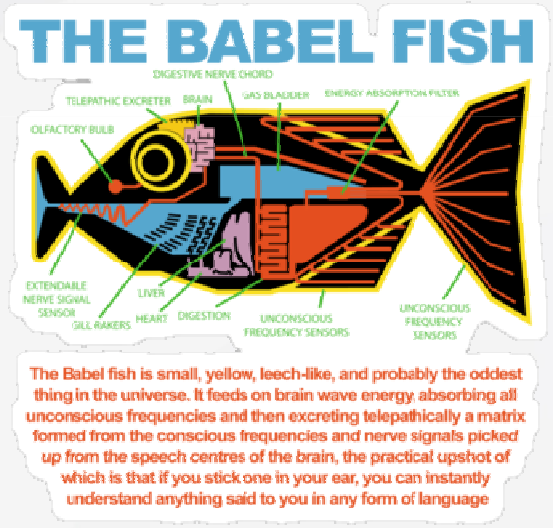
JavaCro'16

JS-to-JS transpiler is currently required to run ES6.

Transpiler compiles code from the latest version into older versions of the language.

As browser support gets better ES7 and ES8 will be transpilled into ES6 and beyond.

Transpilers like Babel also provide human-readable output.



JavaCro'16

# 3) Syntax

# Let & Const

JavaCro'16

Let and const are alternatives to var when declaring variables.

Let is block-scoped instead of lexically scoped to a function.

Let is hoisted to the top of the block, while var declarations are hoisted to top of the function.

```
for(let i = 0, l = list.length; i < l; i++) {
  // do something with list[i]
}
console.log(i); // undefined
```

# Let & Const

JavaCro'16

Const is also block-scoped, hoisted and must be initialized

Assigning to const after initialization fails silently

(or with an exception under strict mode).

```
const MY_CONSTANT = 1;
MY_CONSTANT = 2 // Error, attempt to change
const SOME_CONST; // Error, not initialized
```

Object properties can still be changed

```
const MY_OBJECT = {some: 1};
MY_OBJECT.some = 'body';
```

# Template Strings

JavaCro'16

Template strings provide syntactic sugar for constructing strings similar to string interpolation features in Perl, Python..

```
var text = (
    `foo
    bar
    baz`)
```

```
var name = "Bob", time = "today";
`Hello ${name}, how are you ${time}?`
```

# Destructuring

JavaCro'16

Destructuring provides binding using pattern matching,
with support for matching arrays and objects.

```
// Array matching
  var list = [ 1, 2, 3 ]
  var [ a, , b ] = list   // a=1, b=3
  [ b, a ] = [ a, b ]     // a=3, b=1

// Fail-soft matching
var [missing] = [];
console.log(missing); // undefined
```

```
// Object matching
var robotA = { name: "Bender" };
var robotB = { name: "Flexo" };

var { name: nameA } = robotA;
console.log(nameA);   // "Bender„
var { name: nameB } = robotB;
console.log(nameB);   // "Flexo"
```

# Object Literals

JavaCro'16

Object literals are extended with several new features:

```
var obj = {
    // __proto__
    __proto__: theProtoObj,
    // Shorthand for ‘handler: handler’
    handler,
    // Methods
    toString() {
     // Super calls
     return "d " + super.toString();
    },
    // Computed (dynamic) property names
    [ 'prop_' + (() => 42)() ]: 42
};
```

JavaCro'16

# 4) Functions

# Arrow Functions

Arrows are a function shorthand using the

param => return_value syntax:

```
// Expression bodies
var odds = evens.map(v => v + 1);
var nums = evens.map((v, i) => v + i);
var pairs = evens.map(v => ({even: v, odd: v + 1}));

// Statement bodies
nums.forEach(v => {
  if (v % 5 === 0)
    fives.push(v);
});
```

# Arrow Functions

Arrow functions inherit THIS value from the enclosing scope:

```
this.nums.forEach((v) => {
    if (v % 5 === 0)
        this.fives.push(v)
})
```

In ES5 we have to use self/that trick:

```
var self = this;
this.nums.forEach(function (v) {
    if (v % 5 === 0)
        self.fives.push(v);
})
```

JavaCro'16

# 5) Classes

# Classes (1)

JavaCro'16

ES6 classes are syntactic sugar over the prototype-based OO pattern.

Classes are a well-debated feature of ES6.

Some believe that they go against the prototypal nature of JavaScript, while others think they lower the entry barrier for beginners and people coming from other languages.

# Classes (2)

JavaCro'16

```
class Vehicle {
  constructor(name) {
    this.name = name;
    this.kind = 'vehicle';
  }
  getName() {
    return this.name;
  }
}

// Create an instance
let myVehicle = new Vehicle('rocky');
```

JavaCro'16

# 6) Collections

# Iterators

JavaCro'16

```
let fibonacci = {
  [Symbol.iterator]() {    // default iterator for an object.
    let pre = 0, cur = 1;
    return {
      next() {                  // required method
        [pre, cur] = [cur, pre + cur];
        return { done: false, value: cur }
      }
    }
  }
}
```

# For .. of Loop

JavaCro'16

For .. of loop is new loop for all iterables.

It starts by calling the [Symbol.iterator]() method

which returns a new iterator object.

An iterator object can be any object with a next() method.

```
for (var n of fibonacci) {
  // truncate the sequence at 1000
  if (n > 1000)
    break;
  console.log(n);
}
```

# Map + Set

JavaCro'16

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

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

# WeakMap + WeakSet

JavaCro'16

Weak collections allow GC collection of their keys.

```
// Weak Maps
var wm = new WeakMap();
wm.set(s, { extra: 42 });
wm.size === undefined
// Weak Collections are not enumerable and do not have size

// Weak Sets
var ws = new WeakSet();
ws.add({ data: 42 });
// If data has no other references, it can be GC collected
```

JavaCro'16

# 7) Modules

# Modules (1)

JavaCro'16

Language-level support for modules for component definition.

Codifies patterns from AMD, CommonJS ..

Runtime behaviour defined by a host-defined default loader.

Implicitly async model – no code executes until requested modules are available and processed.

# Modules (2)

JavaCro'16

```
// lib/math.js
export function sum(x, y) {
  return x + y;
}
export var pi = 3.141593;

// app.js
import * as math from "lib/math";
alert("2π = " + math.sum(math.pi, math.pi));
```

JavaCro'16

# 8) Promises

# Promises

JavaCro'16

Promises are used for asynchronous programming.

Promises are first class representation of a value that may be made available in the future.


Promises are used in many existing JavaScript libraries.

# Promises

JavaCro'16

```
function resolveUnderThreeSeconds (delay) {
  return new Promise(function (resolve, reject) {
    setTimeout(resolve, delay);
    // once a promise is settled, it's result can't change
    setTimeout(reject, 3000);
  })
}
resolveUnderThreeSeconds(2000); // resolves!
resolveUnderThreeSeconds(7000);
 // fulfillment took so long, it was rejected.
```

JavaCro'16

# 9) The Future

# Future releases (1)

JavaCro'16

ES7 is due 12 months after ES6.

TC39 (the ECMAScript standard committee)
is embracing 12 month release cycle.

New, naming convention:

ES6 = ES2015

ES7 = ES2016

ES8 = ES2017 ...

Completed features will be published once a year,
others will be scheduled for the next release.

# Future releases (2)

JavaCro'16

*„ECMAScript is a vibrant language and the evolution of the language is not complete. Significant technical enhancement will continue with future editions of this specification."*

... ES 5 specification

# Future releases (3)

JavaCro'16

New proposed features for ES7:

- concurrency and atomics,
- zero-copy binary data transfer,
- number and math enhancements,
- observable streams,
- better metaprogramming with classes,
- class and instance properties,
- operator overloading,
- value types (first-class primitive-like objects),
- records, tuples, traits ...

# The End

JavaCro'16

To be continued ...