The Future of JavaScript – ES6

Rami Sayar - @ramisayar
Technical Evangelist
Microsoft Canada

I've seen the

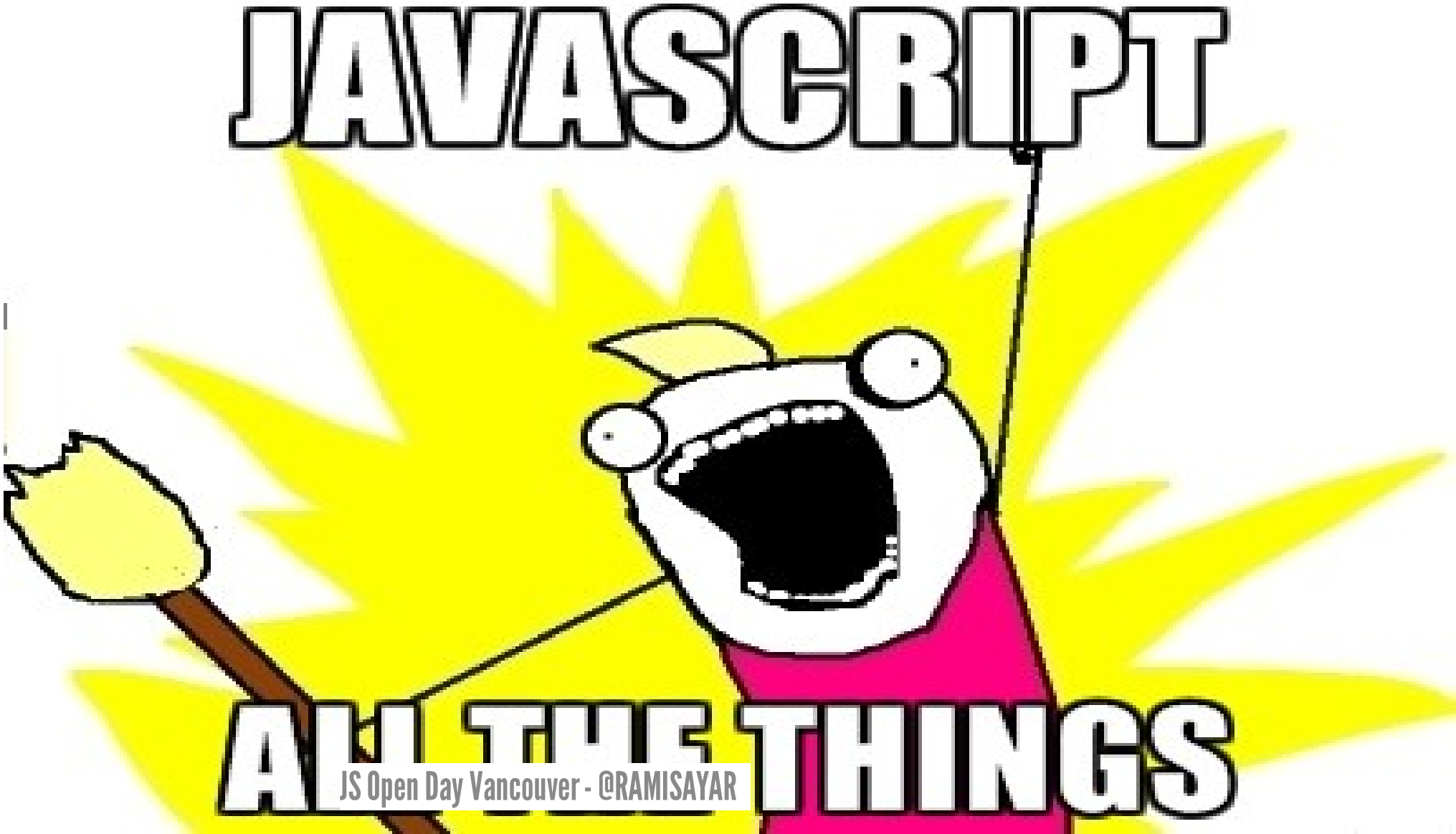
FUTURE

It's in my

BROWSER

JS Open Day Vancouver - @RAMISAYAR

Why Should You Care?



What is ECMAScript?

- ECMAScript is the scripting language standardized by Ecma International as ECMA-262.
- ECMAScript implementations include JavaScript, JScript and ActionScript.
- Most commonly used as the basis for client-side scripting on the Web => JavaScript.

Where is ECMAScript Now?

Edition	Date Published	Notes
1	June 1997	First edition.
2	June 1998	Editorial changes. Aligning with ISO standard.
3	December 1999	Added regex, string handling, new control statements, try/catch, etc.
4	**ABANDONED**	
5	December 2009	Strict mode subset, clarification, harmonization between real-world and the spec. Added support for JSON and more object reflection.
5.1	June 2011	Aligning with ISO standard.
6	Scheduled for Mid-2015	**NEW SYNTAX**
7	WIP	Very early stage of development.

ECMAScript 6

ECMAScript 5 | 6 | 7 | intl | non-standard compatibility table

Please note that *some of these tests* represent **existence**, not functionality or full conformance.

Sort by number of features? ☐ Show obsolete platforms? ☐ Show unstable platforms? ☑

V8 SpiderMonkey JavaScriptCore Chakra Carakan KJS Other

Useful feature Significant feature Landmark feature

		Compilers/polyfills						Desktop browsers											
	42%	63%	75%	29%	16%	25%	21%	3%	15%	73%	39%	64%	65%	69%	42%	42%	45%	5%	
Feature name	Current browser	Traceur	Babel + core-js[1]	Closure	JSX[2]	Type-Script	es6-shim	IE 10	IE 11	IE Technical Preview[3]	FF 31 ESR	FF 37	FF 38	FF 39	CH 41, OP 28[4]	CH 42, OP 29[4]	CH 43, OP 30[4]	SF 6.1, SF 7	SF
Optimisation																			
proper tail calls (tail call optimisation)	0/2	0/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0
Syntax																			
default function parameters	0/6	3/6	5/6	4/6	0/6	3/6	0/6	0/6	0/6	0/6	3/6	3/6	3/6	3/6	0/6	0/6	0/6	0/6	0
rest parameters	0/5	4/5	3/5	2/5	3/5	3/5	0/5	0/5	0/5	4/5	3/5	3/5	4/5	4/5	0/5	0/5	0/5	0/5	0
spread (...) operator	0/10	10/10	10/10	2/10	0/10	2/10	0/10	0/10	0/10	6/10	8/10	10/10	10/10	10/10	0/10	0/10	0/10	0/10	2/
object literal extensions	0/6	6/6	6/6	4/6	3/6	5/6	0/6	0/6	0/6	6/6	0/6	6/6	6/6	6/6	0/6	0/6	3/6	0/6	1
for..of loops	5/7	7/7	7/7	2/7	0/7	2/7	0/7	0/7	0/7	5/7	4/7	5/7	5/7	5/7	5/7	5/7	5/7	0/7	1
octal and binary literals	4/4	2/4	4/4	2/4	0/4	2/4	0/4	0/4	0/4	4/4	2/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0
template strings	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0
RegExp "y" and "u" flags	0/2	1/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	0/2	0
destructuring	0/30	24/30	25/30	16/30	15/30	20/30	0/30	0/30	0/30	0/30	17/30	22/30	22/30	22/30	0/30	0/30	0/30	0/30	15
Unicode code point escapes	0/2	1/2	1/2	1/2	0/2	1/2	0/2	0/2	0/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0
Bindings																			
const	5/8	6/8	6/8	6/8	0/8	4/8	0/8	0/8	8/8	8/8	3/8	8/8	8/8	8/8	5/8	5/8	5/8	1/8	1
let	5/10	8/10	8/10	8/10	0/10	6/10	0/10	0/10	8/10	8/10	0/10	0/10	0/10	0/10	5/10	5/10	5/10	0/10	0/
block-level function declaration[14]	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes	No	No	No	No	Yes	Yes	Yes	No	N

Getting ES6

- ES6 in the Browser
  - Microsoft Edge has most complete ES6 support – Try it in Windows 10.
    - Enable “Experimental JavaScript features” flag
  - Chrome Canary
    - Go to chrome://flags & turn on “Enable Experimental JavaScript”
  - Firefox Nightly or Firefox Developer Edition

Getting ES6

- ES6 in Node.js
  - Need to use `--v8-options` flag

```
node --v8-options | grep harmony
  --harmony_typeof #(enable harmony semantics for typeof)
  --harmony_scoping #(enable harmony block scoping)
  --harmony_modules #(enable harmony modules (implies block scoping))
  --harmony_proxies #(enable harmony proxies)
  --harmony_collections #(enable harmony collections (sets, maps, and weak maps))
  --harmony #(enable all harmony features (except typeof))
```

Let’s take a look!

ES6 – Block Scoping

ES6 – Block Scoping

```
var foo = 'JSOpenDay';
console.log(foo); // Prints 'JSOpenDay'
if (true) {
  var foo = 'BAR';
  console.log(foo); // Prints 'BAR'
}
console.log(foo); // Prints 'BAR'
```

Challiyil Eswaramangalath Pavithran Vipin

ES6 – Block Scoping

- Scoping in JS is at the function-level for variables and functions.

```
var foo = 'JS';
console.log(foo); // Prints 'JS'
if (true) {
  var foo = 'BAR';
  console.log(foo); // Prints 'BAR'
}
console.log(foo); // Prints 'BAR'
```

```
var foo;
foo = 'JS';
console.log(foo); // Prints 'JS'
if(true) {
    foo = 'BAR';
    console.log(foo); // Prints 'BAR'
}
console.log(foo); // Prints 'BAR'
```

ES6 – Block Scoping

- ‘Hoisting’ in JavaScript

```
var foo = 'JS';
if(!bar) {
    console.log(foo + ' ' + bar);
    // Prints 'JS undefined'
}
var bar = '2015';
console.log(foo + ' ' + bar);
// Prints 'JS 2015'
```

```
var foo, bar;
foo = 'JS';
if(!bar) {
    console.log(foo + ' ' + bar);
    // Prints 'JS undefined'
}
bar = '2015';
console.log(foo + ' ' + bar);
// Prints 'JS 2015'
```

- Variables are ‘hoisted’ to the top even if they will never be executed in any statement.
- You can enforce ‘hoisting’ syntactically with JSLint ‘onevar’.

ES6 – Block Scoping

- Scoping in JS is at the function-level for variables and functions.

```
var foo = 'JS';
console.log(foo); // Prints 'JS'
if (true) {
  var foo = 'BAR';
  console.log(foo); // Prints 'BAR'
}
console.log(foo); // Prints 'BAR'
```

```
var foo;
foo = 'JS';
console.log(foo); // Prints 'JS'
function foobar() {
    var foo = 'BAR';
    console.log(foo); // Prints 'BAR'
}
foobar();
console.log(foo); // Prints 'JS'
```

ES6 – Block Scoping

- ES6 introduces ‘let’ & ‘const’.

- Variables declared with ‘let’ are bound to the lexical (in the code) environment as opposed to the variable environment.
- All “block” statements (e.g. if, for, etc.) now have a lexical environment context.
- Variables declared with ‘let’ are scoped to the block statement.
- This is inline with other C-like languages like Java, C++, etc.

ES6 – Block Scoping

- Variable ‘foo’ declared with ‘let’.

```
let foo = 'JS';
console.log(foo); // Prints 'JS'
if (true) {
  let foo = 'BAR';
  console.log(foo); // Prints 'BAR'
}
console.log(foo); // Prints 'JS'
```

- No ‘hoisting’ of variables when declared with ‘let’.

ES6 – Block Scoping

- Variable ‘foo’ declared with ‘const’ is also scoped to the block.

```
const foo = 'JS';
console.log(foo); // Prints 'JS'
if (true) {
  let foo = 'BAR';
  console.log(foo); // Prints 'BAR'
}
// foo = 'BAR';
// The above line causes “SyntaxError: Assignment to constant variable.”
console.log(foo); // Prints 'JS'
```

ES6 – Destructuring

ES6 – Destructuring

- Destructuring is a syntax feature that allows you to pattern match values to variables or properties thus extracting data.

```
var [foo, bar, ABC] = ['bar', 'foo', 3];
console.log(foo + ' ' + bar + ' ' + ABC);
// Prints 'bar foo 3'

var foo = 'bar', bar = 'foo', ABC = 3;
console.log(foo + ' ' + bar + ' ' + ABC);
// Prints 'bar foo 3'
```

ES6 – Destructuring

- Destructuring is a syntax feature that allows you to pattern match values to variables or properties.
- Can be used to swap variables like in Python.

```
var [foo, bar] = ['bar', 'foo'];
[foo, bar] = [bar, foo];
console.log(foo + ' ' + bar);
// Prints 'foo bar'
```

ES6 – Destructuring

- Destructuring is a syntax feature that allows you to pattern match values to variables or properties.
- Not limited to arrays, you can apply destructuring to objects.

```
// Simple example without assigning new names
var {x, y} = {x: "X", y: "Y"};
console.log(x); // X
console.log(y); // Y
// getTalk() returns -> { speaker: { name: "Rami" }, title: "Future of JS"}
var { title: talk_title, speaker: { name: speaker_name }} = getTalk();
console.log(talk_title); // "Future of JS“
console.log(speaker_name); // "Rami"
```

ES6 – Destructuring

- Destructuring is a syntax feature that allows you to pattern match values to variables or properties.
- Can be used to name parameter positions, AWESOME!

```
function g({name: x}) {
  console.log(x);
}
g({name: 5})
```

ES6 – Destructuring

- Destructuring is a syntax feature that allows you to pattern match values to variables or properties.

```
// Fail-soft destructuring
var [a] = [];
a === undefined;

// Fail-soft destructuring with defaults
var [a = 1] = [];
a === 1;
```

ES6 – Iterators & Generators

ES6 – Iterators & Generators

“An Iterator is an object that knows how to access items from a collection one at a time, while keeping track of its current position within that sequence. In JavaScript an iterator is an object that provides a next() method which returns the next item in the sequence.”- MDN

ES6 – Iterators

```
var obj, it, pair;

obj = { foo: 'bar', conference: 'JSOpenDay' };
it = Iterator(obj);

pair = it.next(); // ["foo", "bar"]
pair = it.next(); // ["conference", "JSOpenDay"]
pair = it.next(); // StopIteration exception thrown
```

ES6 – Iterators

- for... in loop calls .next() for you automatically when used with an iterator.
- You can also use iterators with arrays.

```
var evangelists = ['@ramisayar', '@tommylee'];
var iterator = Iterator(evangelists);
for (let [index, item] in iterator)
  console.log(index + ': ' + item);
  // prints "0: @ramisayar" etc.
```

ES6 – Generators

- Generators are factories for iterators. They are functions that continue execution on **next()** and save their context for re-entrances.
- Generators are familiar to Python programmers.
- Generators introduce **function *** and **yield**.
- Generators can replace callbacks.

ES6 – Generators

```
function *foo() {
    yield 1;
    yield 2;
    yield 3;
    yield 4;
    yield 5;
}
```

Eva Blue

Attribution 2.0 Generic

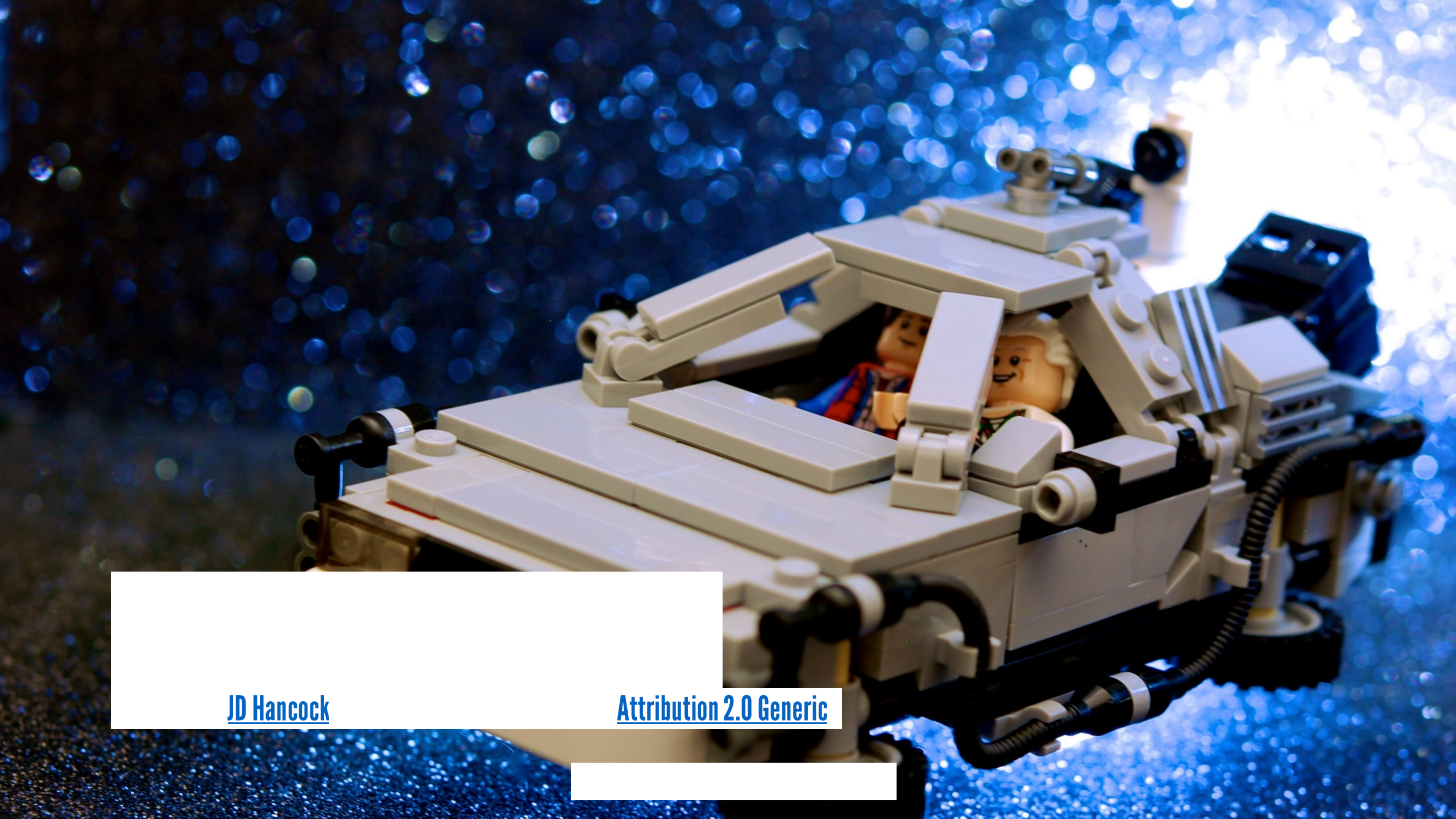
ES6 – Who Has It?

	IE11	Project Spartan (IE11+)	FF39	Chrome 43	Node	io.js
const	**8/8**	**8/8**	**8/8**	**5/8**	**1/8**	**5/8**
let	**8/10**	**8/10**	**0/10 w/Flag**	**5/10**	**0/10**	**5/10**
block-level function declaration	**Yes**	**Yes**	**No**	**Yes**	**Flag**	**Yes**
destructuring	**0/30**	**0/30**	**22/30**	**0/30**	**0/30**	**0/30**
classes	**0/23**	**20/23**	**20/23**	**Flag**	**0/23**	**Flag**
generators	**0/21**	**0/21**	**18/21**	**14/21**	**Flag**	**12/21**

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

Going Back In Time

- Google Traceur, ES6 Compiler: https://github.com/google/traceur-compiler
- Babel, ES6 Compiler: https://babeljs.io/
  - Special support for JSX & React
  - Support for extensions and plugins
- Continuum, ES6 Virtual Machine written with ES3: https://github.com/Benvie/continuum
  - Theoretically, support goes all the way back to IE6.



JD Hancock

Attribution 2.0 Generic

Back to the Future

- xto6, convert JavaScript to ES6: https://github.com/mohebifar/xto6
- es6-shim, adding support for ES6: https://github.com/paulmillr/es6-shim
- es6-module-loader, module loader support: https://github.com/ModuleLoader/es6-module-loader

What did we learn?

- What's ECMAScript6?
- Block Scoping
- Destructuring
- Modules and Classes
- Iterators and Generators

- There is plenty more in ES6!

Thank You! Questions?

tw: @ramisayar | gh: @sayar

gist.github.com/sayar/d8f64a80d3a410ba5cba

slideshare.net/ramisayar

Resources, References, Links

- ES6 Compatibility Table
- ES6 Browser Support
- What's new in JavaScript?
- An introduction to ES6 Part 1: Using ES6 Today
- An introduction to ES6 Part 2: Block Scoping
- An introduction to ES6 Part 3: Destructuring
- Tracking ECMAScript 6 Support
- ES6 (a.k.a. Harmony) Features Implemented in V8 and Available in Node
- React Introduces Support for ES6 Classes

Resources, References, Links

- ECMAScript 6 Features - Introduction
- ECMAScript 6 modules: the final syntax
- The Basics Of ES6 Generators
- ECMAScript 6 and Block Scope
- Understanding ES6 Generators
- MDN - Iterators and generators
- Classes in JavaScript ES6
- ECMAScript 6 modules: the future is now

Resources, References, Links

- es6-shim
- es6-module-loader
- Continuum
- Xto6
- Koa.js
- Babel.js
- traceur-compiler