The Future of JavaScript – ES6

Rami Sayar - @ramisayar
Technical Evangelist
Microsoft Canada
Agenda

• What’s ECMAScript6?
• Learning about Block Scoping
• Learning about Destructuring
• Learning about Iterators and Generators
Assumptions

• Working knowledge of JavaScript and HTML5.

Note: Slides will be made available. Follow @ramisayar.
I’ve seen the FUTURE
It’s in my BROWSER
Why Should You Care?

JavaScript

All the Things

WORDCAMP VANCOUVER - @RAMISAYAR
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?

<table>
<thead>
<tr>
<th>Edition</th>
<th>Date Published</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>June 1997</td>
<td>First edition.</td>
</tr>
<tr>
<td>3</td>
<td>December 1999</td>
<td>Added regex, string handling, new control statements, try/catch, etc.</td>
</tr>
<tr>
<td>4</td>
<td>ABANDONED</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>December 2009</td>
<td>Strict mode subset, clarification, harmonization between real-world and the spec. Added support for JSON and more object reflection.</td>
</tr>
<tr>
<td>5.1</td>
<td>June 2011</td>
<td>Aligning with ISO standard.</td>
</tr>
<tr>
<td>6</td>
<td>June 2015</td>
<td><strong>NEW SYNTAX</strong></td>
</tr>
<tr>
<td>7</td>
<td>WIP</td>
<td>Very early stage of development.</td>
</tr>
</tbody>
</table>
ECMAScript 6

- Significant update to the language and major JavaScript engines are implementing features as we speak.
- Also known as ES6 or ECMAScript 2015 or Harmony or JavaScript 2015.
- ES6 draft was ratified on June 17, 2015. Read it here: http://www.ecma-international.org/ecma-262/6.0/
- Status Tables:
  - Kangax
  - ES6 Matrix by Thomas Lahn:
 ECMA Script 6

Note: The below might be out of date by the time you read it.

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

| Feature name | Current browser | Tracer | JSLint | JSHUnit | Type-Script | est-\shim | IE 10 | IE 11 | Edge | FF 31 | ES5 | FF 38 | FF 39 | FF 40 | CH 43 | OP 30 | CH 44 | OP 31 | CH 45 | OP 32 | SF 6.1 | SF 7 |
|--------------|----------------|--------|--------|--------|-------------|---------|-------|-------|------|------|-----|------|------|------|-------|------|-------|------|-------|------|-------|-------|-------|
| Optimisation |               |        |        |        |             |         |       |       |      |      |    |      |      |      |       |      |       |      |       |      |       |      |
| proper tail calls (tail call optimisation) | 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 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 |
| Syntax |               |        |        |        |             |         |       |       |      |      |    |      |      |      |       |      |       |      |       |      |       |      |
| default function parameters | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 |
| rest parameters | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 | 0/5 |
| spread (... operator | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 | 0/12 |
| object literal extensions | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 | 0/6 |
| for of loops | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 |
| actual and binary literals | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 | 0/4 |
| template strings | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 | 0/3 |
| class and object flags | 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 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 |
| Bindings |               |        |        |        |             |         |       |       |      |      |    |      |      |      |       |      |       |      |       |      |       |      |
| const | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 | 0/8 |
| let | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 | 0/10 |
| block-level function declaration | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Functions |               |        |        |        |             |         |       |       |      |      |    |      |      |      |       |      |       |      |       |      |       |      |
| arrow functions | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 |
| class | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 | 0/22 |

Legend:
- V8: SpiderMonkey
- JavaScriptCore
- Chakra
- C automobiles
- ES5: Other

- Listful feature (1 point)
- Significant feature (2 points)
- Landmark feature (4 points)
Getting ES6

• ES6 in the Browser
  • Microsoft Edge (Windows 10) has excellent ES6 support!
    • Go to about:flags & turn on “Enable experimental JavaScript features” flag
  • Chrome Canary
    • Go to chrome://flags & turn on “Enable Experimental JavaScript”
  • Firefox Nightly or Firefox Developer Edition
ES6 in Node.js: --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

```javascript
var foo = 'JSOpenDay';
console.log(foo); // Prints 'JSOpenDay'
if (true) {
  var foo = 'BAR';
  console.log(foo); // Prints 'BAR'
}
console.log(foo); // Prints 'BAR'
```
ES6 - Block Scoping

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

```javascript
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'
```
• ‘Hoisting’ in JavaScript

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

```javascript
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.

```javascript
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 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’.

```javascript
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.

```javascript
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.

```javascript
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

• Can be used to swap variables without a temporary variable.
ES6 – Destructuring

- Can be used to swap variables without a temp var... like in Python.

```javascript
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.
ES6 – Destructuring

// Simple example without assigning new names

```javascript
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**!

```javascript
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

• Object that knows how to access the next item in a collection.
• Iterators provide next() function that returns an object containing two properties, done && value:

```javascript
{
    done: true, // true|false
    value: undefined // any variable
}
```
ES6 – Iterators

```javascript
function makeIterator(array){
    var nextIndex = 0;
    return {
        next: function(){
            return nextIndex < array.length ? {
                value: array[nextIndex++],
                done: false
            } : {
                done: true
            };
        }
    }
}
```
ES6 – Iterators

• You can also use the `for... of` loop (This is different from `for... in`).
• You can also use iterators with arrays.

```javascript
var evangelists = ['@ramisayar', '@tommylee'];
for (let item of iterator)
    console.log(item);
// prints "@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 introduce `function *` and `yield`.
• Generators can replace callbacks.
ES6 – Generators

```javascript
function *foo() {
    var x = 2;
    while (true) {
        x = x * x;
        yield x;
    }
}

var answer = foo();
answer.next().value; // 4
answer.next().value; // 16
```
function *foo() {
    var x = 1, next = 1;
    while(true) {
        x = x * next;
        next = yield x;
    }
}

var answer = foo(); answer.next().value; //1
answer.next(2).value; //2 answer.next(2).value; //4
ES6 – Modules & Classes
I think I've had milk last longer than some JavaScript frameworks.
class SkinnedMesh extends THREE.Mesh {
  constructor(geometry, materials) {
    super(geometry, materials);

    this.idMatrix = SkinnedMesh.defaultMatrix();
    this.bones = [];
    this.boneMatrices = [];
    //...
  }
}
ES6 – Classes

update(camera) {
    //...
    super.update();
}
get boneCount() {
    return this.bones.length;
}
set matrixType(matrixType) {
    this.idMatrix = SkinnedMesh[matrixType]();
}
```javascript
static defaultMatrix() {
  return new THREE.Matrix4();
}
```
ES6 – Modules

• Modularization in software architecture is extremely important.
• Plenty of attempts to implement modules in JavaScript. CommonJS and Asynchronous Module Definition (AMD) patterns had 100s of different implementations.
• Node.js had a built-in module system.
• ES6 Modules borrow the best concepts from CommonJS and AMD.
ES6 – Modules

• The implementation is defined by the JavaScript host.
• Implicitly asynchronous loading.
• Two keywords: “import” & “export”
// math.js
export var pi = 3.141593;

export function add(num1, num2) {
    return num1 + num2;
}

export function circle_area(r) {
    return pi * r * r;
}
ES6 – Modules

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

// otherApp.js
import {circle_area, pi} from "math";
alert("Area of Circle with Radius of 5 = " + circle_area(5));
ES6 – Module Loading API

• Programmatically load modules like in AMD with `system.import`

• Why?
  • Customize how modules are mapped to files.
  • Automatically lint modules on import.
  • Automatically compile CoffeeScript/TypeScript on import.
  • Use other module systems...
# ES6 – Who Has It?

Note: The below might be out of date by the time you read it.

<table>
<thead>
<tr>
<th></th>
<th>IE11</th>
<th>Edge</th>
<th>FF40</th>
<th>Chrome 43</th>
<th>Node</th>
<th>io.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>const</td>
<td>8/8</td>
<td>8/8</td>
<td>8/8</td>
<td>5/8</td>
<td>1/8</td>
<td>5/8</td>
</tr>
<tr>
<td>let</td>
<td>8/10</td>
<td>8/10</td>
<td>Flag</td>
<td>5/10</td>
<td>0/10</td>
<td>5/10</td>
</tr>
<tr>
<td>block-level</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Flag</td>
<td>Yes</td>
</tr>
<tr>
<td>function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>declaration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destructuring</td>
<td>0/32</td>
<td>0/32</td>
<td>23/32</td>
<td>0/32</td>
<td>0/32</td>
<td>0/32</td>
</tr>
<tr>
<td>classes</td>
<td>0/23</td>
<td>Flag</td>
<td>20/23</td>
<td>Flag</td>
<td>0/23</td>
<td>Flag</td>
</tr>
<tr>
<td>generators</td>
<td>0/22</td>
<td>Flag</td>
<td>19/22</td>
<td>15/22</td>
<td>Flag</td>
<td>15/22</td>
</tr>
</tbody>
</table>

Going Back In Time

- **Google Traceur**, ES6 Compiler: [https://github.com/google/traceur-compiler](https://github.com/google/traceur-compiler)
- **Babel**, ES6 Compiler: [https://babeljs.io/](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](https://github.com/Benvie/continuum)
  - Theoretically, support goes all the way back to IE6.
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
Dev tools for the modern web

Check and debug your site for compatibility with these cross platform tools during development

**Virtual machines**
Test versions of IE from 6 through 11 using virtual machines you download and manage locally.

**RemoteIE**
Free test service using Azure RemoteApp to run IE on your Windows, Mac, iOS or Android device.

**Quick site scan**
Run a quick static code scan on any URL to check for out-of-date libraries, layout issues and accessibility.

**Generate screenshots**
See how your site renders across 9 common browsers and devices.
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/d8f64a80d3a410ba5cda
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