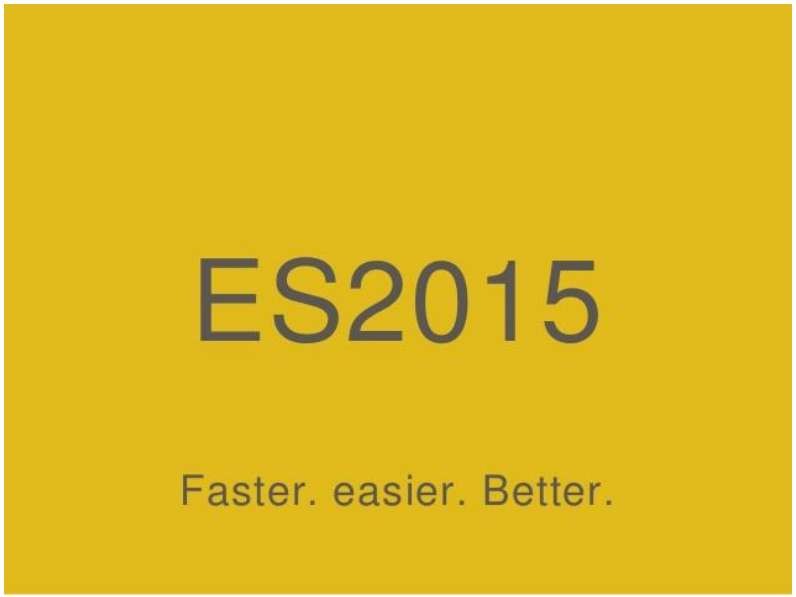
Current standard for JavaScript

Diarmuid O’Connor

Frank Walsh

Main goals of ES6

- fix (*some)* ES5 problems
- backwards compatible (ES5 code is valid in ES6)
- More expressive and intuitive syntax
- Suits bigger apps
- Bring in new stuff…



ECMAScript version history

Edition	Date published	Changes from prior edition	Editor
1	June 1997	First edition	Guy L. Steele Jr.
2	June 1998	Editorial changes to keep the specification fully aligned with ISO/IEC 16262 international standard	Mike Cowlishaw
3	December 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	Mike Cowlishaw
4	Abandoned	Fourth Edition was abandoned, due to political differences concerning language complexity. Many features proposed for the Fourth Edition have been completely dropped; some are proposed for ECMAScript Harmony.	
5	December 2009	Adds "strict mode," a subset intended to provide more thorough error checking and avoid error-prone constructs. Clarifies many ambiguities in the 3rd edition specification, and accommodates behaviour of real-world implementations that differed consistently from that specification. Adds some new features, such as getters and setters, library support for JSON, and more complete reflection on object properties.[10]	Pratap Lakshman, Allen Wirfs-Brock
5.1	June 2011	This edition 5.1 of the ECMAScript Standard is fully aligned with third edition of the international standard ISO/IEC 16262:2011.	Pratap Lakshman, Allen Wirfs-Brock
6	June 2015[11]	The Sixth Edition, initially known as ECMAScript 6 (ES6) and later renamed to ECMAScript 2015 (ES2015)[11] adds significant new syntax for writing complex applications, including classes and modules, but defines them semantically in the same terms as ECMAScript 5 strict mode. Other new features include iterators and `for/of` loops, Python-style generators and generator expressions, arrow functions, binary data, typed arrays, collections (maps, sets and weak maps), promises, number and math enhancements, reflection, and proxies (metaprogramming for virtual objects and wrappers). As the first "ECMAScript Harmony" specification, it is also known as "ES6 Harmony."	Allen Wirfs-Brock
7	June 2016[12]	The Seventh Edition, also known as ECMAScript 2016,[12] intended to continue the themes of language reform, code isolation, control of effects and library/tool enabling from ES2015, includes two new features: the exponentiation operator (**) and Array.prototype.includes.	Brian Terlson
8		New features proposed include concurrency and atomics, zero-copy binary data transfer, more number and math enhancements, syntactic integration with promises, observable streams, SIMD types, better metaprogramming with classes, class and instance properties, operator overloading, value types (first-class primitive-like objects), records and tuples, and traits.[13][14]	

Transpilation (using Babel)

- Browsers cannot execute ES6 Javascript
  - Neither can Node platform (kinda).
- Must transpile code first.
  - The Babel tool suite
  - Can be a bit sloooow!!

ES6/2015

- What's new?
  - Block scope variables; Constant variables;
  - String templates;
  - Arrow functions;
  - Classes;
  - Destructuring;
  - …. lots more(but not here!)

Block scope variables

- **Block: Code enclosed in curly braces – {….}**
  - **if, for, while, function, arbitrary.**
- **Use** let **instead of** var **to declare variables**
- **No hoisting**

```
1  // var is function scoped
2  function foo() {
3    if (true) {
4      var a = 2;
5    }
6    console.log(a); // 2
7  }
8  function bar() {
9    var a;
10   if (false) {
11     a = 2;
12   }
13   console.log(a); // undefined
14 }
15 // let is block scoped
16 // A block is any pair of { }
17 function baz() {
18   let a = 2;
19   if (true) {
20     let a = 3;
21     console.log(a); // 3
22   }
23   console.log(a); // 2
24 }
25 // Principle of Least Privilege:
26 //     If a variable is not used outside
27 //     of a block, don't expose it
28 bar() ;
```

Constant variables

- **Cannot be** reassigned**.**
- **Must be assigned at declaration.**
- **Block scoping.**
- **Value (e.g. data object) associated with a const is mutable.**

```
6   const VAL = 12345;
7   const STR = "a constant string";
8   const VALUES = [1, 2, 3, 4];
9   const MY_OBJECT = {
10    key1: 'value'
11  };
12
13  // const is not reassignable, but it is mutable.
14  MY_OBJECT.key1 = 'otherValue';
15  //Add new keys
16  MY_OBJECT.key2 = 'value';
17
18  //VALUES = [5,6] ; // TypeError
```

String templates

- Must use backquote (`) to enclose string, not single quote.
- Interpolation: Embed variable / expressions in strings, using ${ …. }.
  - Expression is evaluated and result inserted into string
- Multi-line strings.

```
1  // Interpolation
2  let name = 'Diarmuid' ;
3  let height = 5.8 ;
4  let meters = (height*0.3048).toFixed(2)
5  let text = `My name is ${name} and my height is ${meters} meters`;
6  console.log(text);
7  let today = new Date();
8  // Multi-line strings
9  text = (
10 `My name is ${name}
11 and my height is ${meters} meters
12 as of  ${today.toLocaleString()}
13 `);
14 console.log(text);
15 // Embedded quotes in a string
16 text = `I'm "amazed" that we have so many quotation marks to choose from!` ;
17 console.log(text);
18
```

```
My name is Diarmuid and my height is 1.77 meters
My name is Diarmuid
and my height is 1.77 meters
as of  8/18/2017, 8:34:05 PM

I'm "amazed" that we have so many quotation marks to choose from!
```

Classes.

- **Same purpose** as constructor functions **(ES5) – for declaring** a custom object type.
- **Use** new operator**, as before.**
- **Custom type (class) has a data part and behavior part.**
- **Behavior declared in** methods**, e.g. print – like functions.**
- **Data declared/initialized in special method,** constructor**.**
- **Classes often derived from real world concepts.**
- **No hoisting.**
- **Classes really 'syntatic sugar' for constructor functions.**

```
9  // ES5 - constructor function
10 function Point(x, y) {
11   this.x = x;
12   this.y = y;
13   this.print = function() {
14     console.log(this.x + ', ' + this.y);
15   }
16 }
17 let p1 = new Point(1,2);
18 p1.print() ;
19
20 // ES6 = Class
21 class PointES6 {
22   constructor(x, y) {
23     this.x = x;
24     this.y = y;
25   }
26   print() {
27     console.log(`${this.x}, ${this.y}`);
28   }
29 }
30 let p2 = new PointES6(2,3);
31 p2.print();
```

Class inheritance

- **Used when classes** share some common behavior and/or data properties**.**
- **Superclasses and subclasses**
- **Use** super() **to call superclass' constructor**
- **Occurs naturally in real world modeling.**

```
 6  class Point {
 7    constructor(x, y) {
 8      this.x = x;
 9      this.y = y;
10    }
11    print() {
12      console.log(`${this.x}, ${this.y}`);
13    }
14  }
15
16  class Point3D extends Point {
17    constructor(x, y, z) {
18      super(x, y);
19      this.z = z;
20    }
21    print() {
22      let output = (
23  `       ${this.x},
24        ${this.y},
25        ${this.z}`) ;
26      console.log(output);
27    }
28  }
29
30  let p1 = new Point3D(1,2,3)
31  p1.print();     // 1,2,3
32  p1.x = 6 ; // Set x in p1
33  p1.print();     // 6,2,3
34  p1 instanceof Point3D; // true
35  p1 instanceof Point; // true
```

Modularity

- Split **application code into multiple files, termed** modules.
- Reusability - make **modules available to other modules.**.
- **Old JS provided module system via separate library; ES6 modules built into language.**
- **Two options: Default exports; Named exports (also Mixed exports)**

```
29  // lib.js
30  export default function() {
31    ...
32  }
33  // ------------------------
34  // bar.js
35  import myFunc from './lib';
36  myFunc();
37
```

```
39  // lib.js  (Named exports)
40  export var myVar = 12345;
41  export let myStr = 'important string';
42  export const pi = 3.14159526;
43  export function myFunc() {
44    ...
45  }
46  export class myClass {
47    ...
48  }
49  // ------------------------
50  // bar.js
51  import {myFunc, myStr} from './lib';
52  myFunc() ;
53  console.log(myStr)
```

Arrow functions

- **A cleaner syntax for anonymous functions.**
- **The => (arrow) separates function body from its parameters.**
- **Enclose multi-line body with curly braces, { … }**
- **Enclose multi parameter list with parentheses, (…).**

```
let people = [ { name:'Sheila', credit: 10 },
               { name:'Frank', credit: 20 },
               { name: 'Aoifa',credit: 30 } ] ;
// ES5
people.forEach(function(person) {
    console.log(`${person.name} says Hello!`);
}) ;
// ************************
// ES6
people.forEach( person => console.log(`${person.name} says Hello!`) );
 // Multi-line anonymous function
people.forEach(person => {
    if (person.credit > 25) {
        person.credit += 2;
    }
});
// Multi-parameter anonymous function
let print = (name,credit) => console.log(`${name} has ${credit} credit`) ;
people.forEach(person => print(person.name,person.credit));
```

Destructuring

- Extract multiple values from data stored in Objects and Arrays
- Object Destructuring:

```
const obj = { first: 'Jane', last: 'Doe' };
const {first: f, last: l} = obj;
        // f = 'Jane'; l = 'Doe'

// {prop} is short for {prop: prop}
const {first, last} = obj;
        // first = 'Jane'; last = 'Doe'
```

- Array Destructuring:

```
const iterable = ['a', 'b'];
const [x, y] = iterable;
        // x = 'a'; y = 'b'
```