ES6/ES2015

Minimum Standard for this module

ECMAScript version history

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 were incorporated into the sixth edition.	
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.[9]	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[10]	The sixth edition, initially known as ECMAScript 6 (ES6) and later renamed to ECMAScript 2015 (ES2015)[10] 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[11]	ECMAScript 2016 (ES2016)[11], the seventh edition, 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	June 2017[12]	ECMAScript 2017 (ES2017), the eighth edition, includes features for concurrency and atomics, syntactic integration with promises (async/await).[13][12]	Brian Terlson
9	June 2018[8]	ECMAScript 2018 (ES2018), the ninth edition, includes features for asynchronous iteration and generators, new regular expression features and rest/spread parameters.[8]	Brian Terlson

ES6/2015

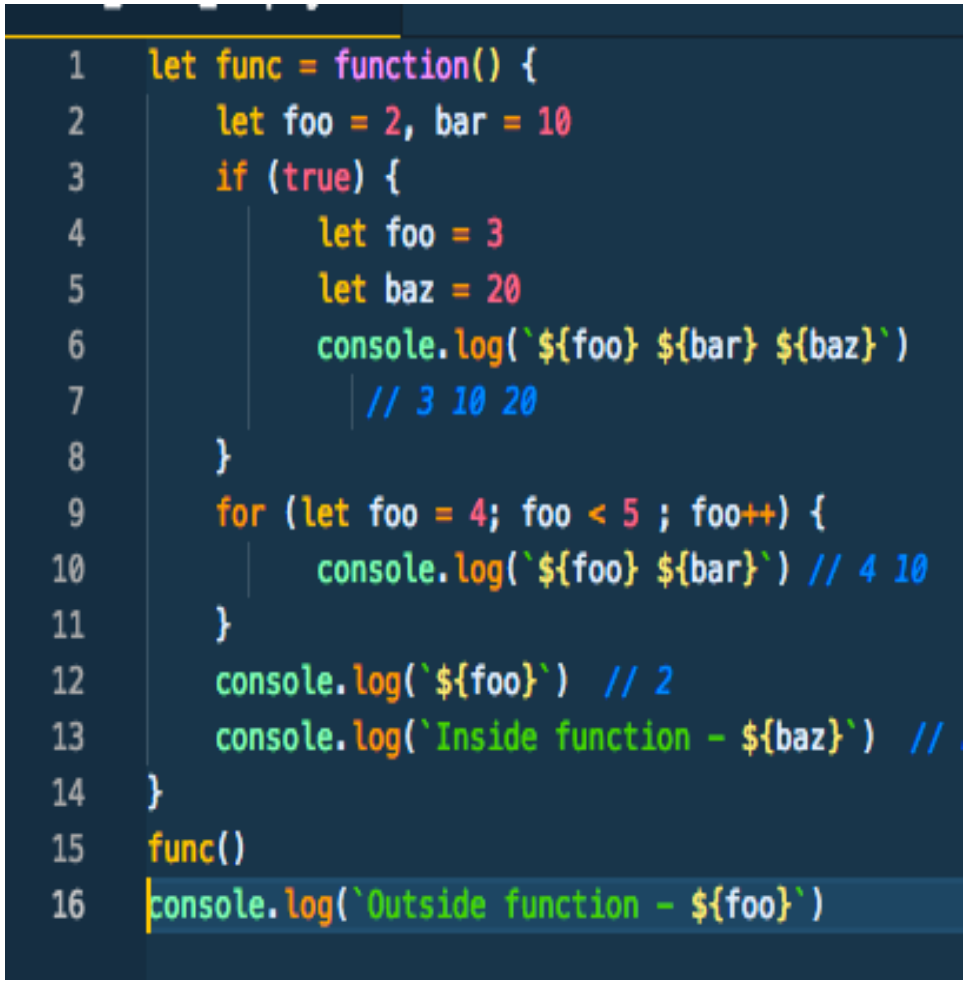
- What's new?
  - Block scope variables; Constant variables;
  - String templates;
  - Destructuringn;
  - Arrow functions;
  - Classes;
  - …. lots more …..

Transpilation (using Babel)

- Older Browsers cannot execute ES6+ JavaScript.
  - Must transpile code first.
- Newer browsers incrementally adopting ES6+.
  - Same for Node.js platform.
- The Babel tool suite.
  - One-stop shop for all transpilitation needs.

Block scope variables

- **Block: Code enclosed in curly braces – {….}**
  - **If, for, while, function, arbitrary.**
- let **and** const **have block scope.**
- **Pre ES6 only had** function **scope, using** var **keyword.**
- **E.g.** var foo = 10;



```
1   let func = function() {
2       let foo = 2, bar = 10
3       if (true) {
4           let foo = 3
5           let baz = 20
6           console.log(`${foo} ${bar} ${baz}`)
7             // 3 10 20
8       }
9       for (let foo = 4; foo < 5 ; foo++) {
10          console.log(`${foo} ${bar}`) // 4 10
11      }
12      console.log(`${foo}`)  // 2
13      console.log(`Inside function - ${baz}`)  //
14  }
15  func()
16  console.log(`Outside function - ${foo}`)
```

Constant variables

- **Cannot be** reassigned**.**
- **Must be initialized on 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

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

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

```
My name is Diarmuid and my height is 1.77 meters
My name is Diarmuid
and my height is 1.77 meters

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

Classes.

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

Class inheritance

- **Used when classes** share some common behavior and/or data properties**.**
- **Occurs naturally in real world modeling.**
- **E.g. Student and Lecturer classes inherit from Person class**

- **Superclasses and subclasses.**
- **Use** super() **to call superclass' constructor**

Modularity

- Split **application code into multiple files, termed** modules.
- Reusability - make **modules available to other modules.**.
- **Pre-ES6 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
```

```
// lib,js  (Named exports)
export let myStr = 'important string';
export const pi = 3.14159526;
export function myFunc() {
  ...
}
export class myClass {
  ...
}
// ------------------------
// bar.js
import {myFunc, myStr} from './lib';
myFunc() ;
console.log(myStr)
```

Arrow functions

- **A cleaner syntax for anonymous functions.**

  function (parameters) { body }

- **The => (arrow) separates function body from its parameters.**

  (parameters) => { body }

- **Enclose body with curly braces, { …… }.**
  - **Unless body is a single expression (optional)**
- **Enclose parameter list with parentheses, (…).**
  - **Unless a single parameter (optional)**
- **Omit return token when single-expression body (optional)**

- **Sensible binding for the** this **special variable.**
  - **Traditional anonymuous functions very problematic!!**

- ……. A lot more we did not cover ….