# ECMAScript 6

Bahasa JavaScript didefinisikan dalam standar [ECMA-262](https://www.ecma-international.org/publications/standards/Ecma-262.htm" \t "/home/mukhlisyaini/Documents\\x/_blank). Bahasa yang didefinisikan menggunakan standar ini disebut dengan ECMAScript. Browser dan Node.js adalah contoh implementasi dari ECMA-262/ECMAScript. Kedua platform tersebut (Browser dan Node.js) menggunakan bahasa JavaScript dengan fungsionalitas yang berbeda. Bagi browser, JavaScript digunakan untuk membantu website menjadi lebih interaktif. Sedangkan bagi Node.js, JavaScript digunakan untuk pengembangan aplikasi diluar dari Browser, seperti Server, Desktop, Mobile, bahkan Game. Sehingga Browser dan Node.js memberikan fungsionalitas lain dengan menambahkan objek dan method sesuai kebutuhan masing - masing.

Namun kembali lagi, bahwa inti dari JavaScript itu sendiri didefinisikan dalam ECMAScript. Tak heran, pengembangan dari ECMA-262 menjadi sangat vital dalam menentukan kesuksesan bahasa JavaScript.

### Sejarah singkat JavaScript

Merunut sejarah, sebenarnya pada tahun 1995 Netscape melahirkan bahasa pemrograman ini dengan nama “LiveScript”, namun pada saat itu bahasa pemrograman Java sedang populer.

Netspace dan Sun selaku pengembang bahasa pemrograman Java (saat ini Oracle) melakukan perjanjian lisensi dan mengubah penamaan LiveScript menjadi JavaScript. Setelah diadopsi di luar Netscape, JavaScript distandarisasi oleh European Computer Manufacturer’s Association (ECMA). Itulah sebabnya terkadang ada yang menyebutnya dengan ECMAScript.

Terdapat beberapa versi dari JavaScript. Pada tahun 2000 - 2010, ECMAScript 3 (ES3) merupakan versi yang banyak digunakan ketika JavaScript sedang mendominasi. Selama waktu tersebut, ECMAScript 4 (ES4) sedang dalam proses pengembangan dengan harapan akan memberikan improvisasi yang cukup signifikan, namun ambisi tersebut tidak berjalan mulus sehingga pada tahun 2008 pengembangan ES4 ditinggalkan.

Walaupun begitu, itu bukan akhir dari JavaScript. Pengembangan digantikan dengan ECMAScript 5 (ES5) dengan mengurangi ambisinya. Seperti apa? perbaikan hanya terbatas pada hal non-kontroversial. Pembaruan tersebut berhasil dan akhirnya ES5 pun rilis pada tahun 2009.

Akhirnya pada tahun 2015 ****ECMAScript 6 (ES6)**** rilis dengan membawa perubahan yang cukup besar termasuk ide - ide yang sudah direncanakan untuk versi 4.

Saat ini JavaScript sudah menyentuh versi ECMAScript 10 (ES10), akan tetapi mulai dari pembaharuan ES6, JavaScript hanya melakukan update tahunan yang bersifat minor. Kita bisa lihat rincian updatenya pada laman [wikipedia](https://en.wikipedia.org/wiki/ECMAScript" \t "/home/mukhlisyaini/Documents\\x/_blank) (<https://en.wikipedia.org/wiki/ECMAScript>) berikut.

Pada modul kali ini kita akan membahas beberapa pembaharuan besar yang ada pada ES6, mulai dari deklarasi variabel, template string, function syntax, class, promise, hingga module.

## Running Snippet Code

Dalam beberapa pembelajaran ke depan kita akan banyak berhadapan dengan potongan-potongan kode. Anda sangat dianjurkan untuk menjalankannya agar lebih memahami materi yang disampaikan.

Bagi yang sudah familiar dengan Node.js, Anda bisa menjalankan potongan-potongan kode yang diberikan dengan mengeksekusinya di dalam berkas JavaScript (.js).

Bagaimana jika Anda belum familiar dengan Node.js? Anda bisa menjalankan potongan kode yang ada dengan menggunakan platform online [repl.it](https://repl.it/" \t "/home/mukhlisyaini/Documents\\x/_blank). (<https://repl.it/>) Apa itu? Repl.it merupakan sebuah coding platform yang berjalan secara online. Dengan repl.it, kita dapat menuliskan kode dan mengeksekusinya secara online. Bahkan lebih dari itu.

Terdapat banyak sekali bahasa pemrograman yang didukung platform ini, salah satunya adalah JavaScript, HTML dan CSS.

## Persiapan Project Latihan

Sebelum membahas fitur dan improvisasi sintaks JavaScript pada ECMAScript 2015 (ES6). Selama pembelajarannya nanti, Anda akan mengimplementasikan apa yang sudah dipelajari melalui sebuah aplikasi sederhana bernama “Club Finder”. Sesuai namanya, aplikasi tersebut berfungsi untuk melakukan pencarian klub olahraga, baik itu basket, sepak bola, dsb.

Saat ini, aplikasi sudah berjalan dengan sesuai yang diharapkan. Namun penulisan kode JavaScript pada aplikasi tersebut tidak dituliskan menggunakan standar ECMAScript 2015. Itulah tugas Anda! Sambil belajar fitur dan sintaks terbaru ECMAScript 2015, Anda juga akan menerapkan ilmu yang Anda pelajari pada proyek aplikasi sederhana. Untuk itu, mari kita persiapkan dulu kebutuhannya.

Silakan untuk unduh project starter pada link berikut ini: [ClubFinder download](https://github.com/dicodingacademy/a163-bfwd-labs/raw/099-starter-proyek-files/ClubFinder.zip" \t "/home/mukhlisyaini/Documents\\x/_blank).

Setelah Anda mengunduh berkas tersebut, silakan extract berkas zip. Kemudian pada komputer Anda akan terdapat satu folder dengan nama  “****ClubFinder****”.

Jika berhasil, Anda bisa menghapus berkas ZIP yang di unduh tadi.

Kemudian silakan buka folder **ClubFinder** menggunakan Text Editor yang Anda miliki. Contohnya di sini kita gunakan Visual Studio Code.

Pada text editor, Anda bisa melihat bagaimana aplikasi Club Finder dibangun. Aplikasi tersebut cukup sederhana dan hanya dibangun menggunakan HTML, CSS dan JavaScript murni tanpa bantuan framework seperti bootstrap, ataupun jQuery. Jika Anda sudah lulus kelas [Belajar Dasar Pemrograman Web](https://www.dicoding.com/academies/123" \t "/home/mukhlisyaini/Documents\\x/_blank)dengan baik, Anda pun bisa membuatnya.

Sekarang project tersebut milik Anda. Anda bebas bereksplorasi untuk mengetahui logika yang ada di dalamnya. Namun seiring pembelajaran yang Anda ikuti, cobalah mengubah kode dengan menggunakan standar ECMAScript 2015.

Sudah siap? Mari kita mulai dengan membahas cara baru dalam mendeklarasikan variabel.

## Variable Declaration

Potongan kode untuk materi ini: [https://repl.it/@dicodingacademy/163-02-variable-declaration?lite=true](https://repl.it/@dicodingacademy/163-02-variable-declaration?lite=true" \t "/home/mukhlisyaini/Documents\\x/_blank)

Awalnya memahami deklarasi variabel di JavaScript agak sedikit rumit. Karena pada JavaScript berbeda dari bahasa pemrograman berbasis bahasa C yang umumnya variabel tersedia pada blok ketika ia dibuat. Namun pada JavaScript kasus tersebut tidak selalu benar. Variabel pada JavaScript akan dibuat tergantung bagaimana cara kita mendeklarasikannya. Sedikit membingungkan bukan? Jangan khawatir, kita akan  bahas masalah ini lebih detail.

### Deklarasi var dan Hoisting

Variabel yang dideklarasikan menggunakan **var** akan selalu diangkat pada tingkatan atas sebuah fungsi walaupun kita menuliskannya bukan pada tingkatan atas fungsi. Proses pengangkatan deklarasi variabel ini disebut dengan hoisting.

Perhatikan contoh kode berikut:

1. function makeTea(isCold) {
2. if(isCold) {
3. var tea = "Make an Ice Tea!"
4. } else {
5. var tea = "Make a Hot Tea!"
6. }
7. return tea;
8. }
10. console.log(makeTea(false));
12. /\* output
13. Make a Hot Tea!
14. \*/

Kode di atas akan menghasilkan output “Make a Hot Tea!”. Mengapa bisa demikian? Padahal kita mendeklarasikan variabel ****tea**** di dalam blok **if** dan blok ****else****. Seharusnya kita tidak dapat mengaksesnya diluar blok tersebut dan menghasilkan error:

1. ReferenceError: tea is not defined

Nah, inilah yang akan kita dapatkan jika menggunakan keyword **var** dalam mendeklarasikan variabel. Di belakang layar, JavaScript mengangkat proses deklarasi variabel **tea** pada tingkatan atas fungsi. Sehingga variabel tersebut akan tersedia selama kita berada di dalam fungsi **makeTea**. Dengan begitu kode sebenarnya akan menjadi seperti ini:

1. function makeTea(isCold) {
2. var tea;
3. if(isCold) {
4. tea = "Make an Ice Tea!"
5. } else {
6. tea = "Make a Hot Tea!"
7. }
8. return tea;
9. }
11. console.log(makeTea(false));
13. /\* output
14. Make a Hot Tea!
15. \*/

Bahkan karena proses hoisting inilah kita bisa menginisialisasi nilai dan menggunakan variabel sebelum ia dideklarasikan. Hal ini sedikit tidak masuk akal bukan?

1. function getFood() {
2. food = "choocolate";
3. console.log(food);
4. var food;
5. }
7. getFood();
9. /\* output
10. choocolate
11. \*/

Proses hoisting menjadi kontroversial karena tidak jarang developer yang dibuat bingung akan hal ini.

### let dan const

Sejak ES6 selain **var**, menginisialisasikan variabel dapat menggunakan **let** dan **const**. ES6 melakukan improvisasi pada deklarasi variabel karena menggunakan **var** terdapat beberapa hal yang kontroversial, salah satunya hoisting yang sudah kita bahas tadi.

Variabel yang dideklarasikan dengan **let** atau **const** akan menghilangkan permasalahan dari hoisting karena variabel akan tersedia pada cakupan block (sama seperti bahasa pemrograman berbasis C), bukan pada fungsi.

Perhatikan kode berikut. Mari gunakan contoh kode sebelumnya namun menggunakan **let** dalam mendeklarasikan variabel:

1. function makeTea(isCold) {
2. if(isCold) {
3. let tea = "Make an Ice Tea!"
4. } else {
5. let tea = "Make a Hot Tea!"
6. }
7. return tea;
8. }
10. console.log(makeTea(false));
12. /\* output
13. ReferenceError: tea is not defined
14. \*/

Variabel yang dideklarasikan menggunakan **let** ataupun **const** juga tidak dapat diakses sebelum ia dideklarasikan, karena variabel akan terhenti pada tempat yang biasa disebut dengan temporal dead zone hingga akhirnya variabel tersebut dideklarasi. Jika kita mencoba melakukannya maka akan menghasilkan eror yang sama.

1. function getFood() {
2. food = "choocolate";
3. console.log(food);
4. let food;
5. }
7. getFood();
9. /\* error:
10. ReferenceError: food is not defined
11. \*/

**let** dan **const** menjadi solusi dari permasalahan hoisting pada JavaScript. Hal ini menjadikan JavaScript lebih ketat dalam pendeklarasian variabel, sehingga dapat meminimalisir peluang terjadinya bug.

### Aturan penggunaan let dan const

Setelah kita mengetahui mengapa kita harus menggunakan **let** dan **const** daripada **var** dalam mendeklarasikan variabel, lantas apa perbedaan dari **let** dan **const** itu sendiri? Kapan kita harus menggunakan **let** daripada **const**? Begitu pula sebaliknya.

Variabel yang dideklarasikan dengan **let** atau pun **const** memiliki kesamaan dan perbedaan karakteristik. Persamaannya adalah variabel yang dideklarasikan dengan **let** atau **const** tidak dapat di deklarasikan ulang pada cakupan yang sama (kita dapat melakukan hal ini ketika menggunakan **var**).

* **[let](https://www.dicoding.com/academies/163/tutorials/6588?from=6581" \l "tab1-code1)**
* [const](https://www.dicoding.com/academies/163/tutorials/6588?from=6581" \l "tab1-code2)

1. let name = "John";
2. let name = "Doe";
4. console.log(name);
6. /\* error:
7. SyntaxError: Identifier 'name' has already been declared
8. \*/

Perbedaanya antara **let** dan **const** adalah jika kita menggunakan **let** maka variabel tersebut dapat diinisialisasi ulang nilainya. Sedangkan **const** tidak bisa, sehingga jika kita menggunakan **const** pastikan kita langsung menginisialisasi nilai dari variabel tersebut.

* [let](https://www.dicoding.com/academies/163/tutorials/6588?from=6581" \l "tab2-code1)
* **[const](https://www.dicoding.com/academies/163/tutorials/6588?from=6581" \l "tab2-code2)**

1. const name = "John";
2. name = "Doe";

5. console.log(name);

8. /\* error:
9. TypeError: Assignment to constant variable.
10. \*/

Jadi intinya kapan kita harus menggunakan **let** dan **const**? Untuk aturan umum penggunaanya adalah sebagai berikut:

* Gunakan **let** ketika variabel yang kita buat akan diinisialisasikan kembali nilainya.
* Gunakan **const** ketika variabel yang kita buat tidak ingin diinisialisasikan kembali nilainya.

**const** merupakan cara yang paling ketat dalam membuat variabel, sehingga pastikan kita menggunakan **const** jika memang kita tidak berencana untuk menginisialisasikan kembali nilainya.

Ada sedikit catatan, bahwa mengubah dan menginisialisasikan ulang nilai pada variabel merupakan hal yang berbeda. Kita bisa lihat perbedaanya dengan jelas ketika sebuah variabel tersebut merupakan array atau objek.

Menginisialisasikan ulang

* [Array](https://www.dicoding.com/academies/163/tutorials/6588?from=6581" \l "tab3-code1)
* **[Object](https://www.dicoding.com/academies/163/tutorials/6588?from=6581" \l "tab3-code2)**

1. const people = { name: "John", age: 18 };
2. people = { name: "John", age: 18, regency: "Bandung" };

5. console.log(people);

8. /\* TypeError: Assignment to constant variable. \*/

Mengubah

* [Array](https://www.dicoding.com/academies/163/tutorials/6588?from=6581" \l "tab4-code1)
* **[Object](https://www.dicoding.com/academies/163/tutorials/6588?from=6581" \l "tab4-code2)**

1. const people = { name: "John", age: 18 }

4. people.regency = "Bandung"

7. console.log(people);

10. /\* output
11. { name: 'John', age: 18, regency: 'Bandung' }
12. \*/

Membuat variabel dengan **const** akan membuat variabel tersebut bersifat read-only, tapi bukan berarti tidak dapat diubah nilainya. Mungkin variabel yang menampung nilai primitif seperti string, number, boolean akan sulit mengubah nilainya tanpa melalui inisialisasi ulang.