DISKUSIKAN MATERI

LAPORKAN MATERI

Daftar Modul

Masukkan kata kunci

Persetujuan Hak Cipta
Modul 1: Introduction Course (Gratis)
Prasyarat Kelas (Gratis)
Apa yang Akan Kita Pelajari? (Gratis)
Tools Requirement
Modul 2: ECMAScript 6 (Gratis)
Running Snippet Code (Gratis)
Persiapan Project Latihan (Gratis)

Variable Declaration

Potongan kode untuk materi ini: https://repl.it/@dicodingacademy/163-02-variable-declaration?lite=true

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:

```
function makeTea(isCold) {
    if(isCold) {
        var tea = "Make an Ice Tea!"
    } else {
        var tea = "Make a Hot Tea!"
    }
    return tea;
}

console.log(makeTea(false));

/* output
Make a Hot Tea!
*/
```

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:

```
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:

```
function makeTea(isCold) {
    var tea;
    if(isCold) {
        tea = "Make an Ice Tea!"
    } else {
        tea = "Make a Hot Tea!"
    }
    return tea;
}

console.log(makeTea(false));

/* output
Make a Hot Tea!
*/
```

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

```
function getFood() {
    food = "choocolate";
    console.log(food);
    var food;
}

getFood();

/* output
choocolate
*/
```

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:

```
function makeTea(isCold) {
```

```
    if(isCold) {
        let tea = "Make an Ice Tea!"
    } else {
        let tea = "Make a Hot Tea!"
    }
    return tea;
}

console.log(makeTea(false));

/* output
ReferenceError: tea is not defined
*/
```

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.

```
function getFood() {
    food = "choocolate";
    console.log(food);
    let food;
}

getFood();

/* error:
ReferenceError: food is not defined
*/
```

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 const

```
let name = "John";
let name = "Doe";

console.log(name);

/* error:
SyntaxError: Identifier 'name' has already been declared
*/
```

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 const

```
let name = "John";
name = "Doe";


console.log(name);


/* output:
Doe
*/
```

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 Object

```
const fruits = ["apple", "orange"];
fruits = ["apple", "orange", "banana"];


console.log(fruits);


/* output
TypeError: Assignment to constant variable. */
```

Mengubah

Array Object

```
const fruits = ["apple", "orange"];
fruits.push("banana");


console.log(fruits);


/* output
[ 'apple', 'orange', 'banana' ]
*/
```

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.

⬅ KEMBALI KE MATERI SEBELUMNYA

LANJUTKAN KE MATERI BERIKUTNYA ➡