ES6

By : PNA & YK

Batam, 01 Maret 2019

INTERMEZO : VAR VS LET & CONST

Sebelum ES6 hanya ada satu cara untuk membuat variabel

```
var x // declaration
x = 1 // assignment
```

atau bisa di singkat dengan cara :

```
var x = 1
```

Hasilnya sama saja

THE PROBLEM WITH VAR

Problem #1 : Reassign

```
var foo ='hello1'

var foo ='hello2'

console.log(foo) //=> hello2
```

Masalahnya disini tidak ada pesan error sama sekali ketika terjadi duplikasi variabel. Tentu ini cukup riskan jika terjadi tanpa sengaja.

Problem #2 : Hoisting

```
x=10

var x

console.log(x) //=> 10
```

Ini sama dengan yang sebelumnya. Kenapa hasilnya bisa 10 ? itu karena *Hoisting*. *Hoisting* artinya mengangkat. Jadi di belakang layar sebenarnya mengangkat var x di angkat keatas menjadi seperti ini :

```
var x

x=10

console.log(x) //=> 10
```

SCOPE

Block scope dalam javascript di tandai dengan simbol { }. Scope artinya pembagian program, ini sering di temui pada *if, for, switch, while* dan sebagainya. Logikanya variabel dalam scope harusnya menjadi *private* dan tidak bisa di baca dari *scope* lain. Tapi mari kita coba buktikan:

var ternyata menjadi global variabel, meskipun ada di dalam scope. Sebelum es6, solusinya adalah membuatnya dalam *function* scope

```
var b=2
     if (true){
     var b=3
     }
console.log(b) //=> 3


//solution
```

```
var b=2
function myScope(){
var b=3
}
myScope()
console.log(b) //=> 2
```

VAR

`var` : Pendeklarasian variabel yang dapat diubah (*reassign*) isinya, bersifat *function-scoped*.

```
1.  function tesvar(x) {
2.
3.   if(x > 1) {
4.     var hello = 'hello'
5.   }
6.   console.log(hello)
7.  }
8.
9.  tesvar(2)
```

Skrip di atas akan mengeluarkan log “hello” walaupun variabel `hello` berada dalam blok `if`, variabel tersebut masih bisa diakses.

LET

`let`: Mirip dengan `var` nilainya dapat diubah kembali (*reassign*), namun bersifat *block-scoped.*

```
1.  function teslet(x) {
2.
3.   if(x > 1) {
4.     let hello = "test"
5.         hello = 'hello'
6.     console.log(hello)
7.   }
8.   console.log(hello)
9.  }
10. testlet(2)
```

fungsi `console.log` yang pertama akan mengeluarkan “hello”, sedangkan pada fungsi `console.log` yang kedua akan mengeluarkan pesan *error : Uncaught ReferenceError: hello is not defined.*
Pesan itu muncul karena variabel `hello` hanya ada di blok `if`.

CONST

`const`: Hampir mirip dengan `const` bersifat *block-scoped,* bedanya `const`nilainya tidak dapat diubah.

```
1.  function tesconst(x){
2.   if(x > 1) {
3.     const hello = "hello"
4.     console.log(hello)
5.   }
6.   console.log(hello)
7.  }
8.  tesconst(2)
```

fungsi `console.log` yang pertama akan mengeluarkan “hello”, sedangkan pada fungsi `console.log` yang kedua akan mengeluarkan pesan *error : Uncaught ReferenceError: hello is not defined.* Sekarang coba skenario kedua, saya ubah nilai `hello`

```
1.  function tesconst(x){
2.   if(x > 1) {
3.     const hello = "hello"
4.           hello = "ubah"
5.     console.log(hello)
6.   }
7.   console.log(hello)
8.  }
9.  tesconst(2)
```

Fungsi di atas akan langsung menampilkan pesan *error : Uncaught TypeError: Assignment to constant variable*.

LITERAL TEMPLATE

Template Strings use back-ticks (``) rather than the single or double quotes we're used to with regular strings. A template string could thus be written as follows:

```
var greeting = `Yo World!`;
```

Template Strings can contain placeholders for string substitution using the `${ }` syntax, as demonstrated below:

```
// Simple string substitution

var name = "Brendan";

console.log(`Yo, ${name}!`);


// => "Yo, Brendan!"
```

As all string substitutions in Template Strings are JavaScript expressions, we can substitute a lot more than variable names. For example, below we can use expression interpolation to embed for some readable inline math:

```
var a = 10;

var b = 10;

console.log(`JavaScript first appeared ${a+b} years ago. Crazy!`);

//=> JavaScript first appeared 20 years ago. Crazy!

console.log(`The number of JS MVC frameworks is ${2 * (a + b)} and not ${10 * (a + b)}.`);

//=> The number of JS frameworks is 40 and not 200.
```

GUARD CLAUSE

The idea is that when you have something to assert in the beginning of a method—do this using a fast return.

EXERCISE :

Refactor those code using literal template and guard clause

5 MINUTE

```
function glintsWarrior(){
    var nama = 'Badai'
    var peran = 'Penyihir'
    if (nama.length == 0) {
        console.log('Nama harus diisi!')
    }   else  if (peran.length == 0) {
        console.log('Halo ' + nama + ', Pilih peranmu untuk memulai game!')
    }   else if (peran === 'Ksatria') {
        console.log ('Selamat datang di Dunia GlintsWarrior, ' + nama + '! Halo Ksatria ' +
        nama + ', Tugas kamu adalah menyerang musuh!' )
    }   else if (peran === 'Tabib') {
        console.log ( 'Selamat datang di Dunia GlintsWarrior, ' + nama + '! Halo Tabib ' +
        nama + ', kamu akan membantu temanmu yang terluka' )
    }   else  if (peran === 'Penyihir') {
        console.log ( 'Selamat datang di Dunia GlintsWarrior, ' + nama + '! Halo Penyihir '
        + nama + ', ciptakan keajaiban yang membantu kemenanganmu!' )
    }   else {
        console.log ( ' Maaf ' + nama + ', peran ini tidak tersedia.')
    }
}
glintsWarrior()
```

ARROW FUNCTION

Arrow functions (also called “fat arrow functions”) are undoubtedly one of the more popular features of ES6. They introduced a new way of writing concise functions.

(arrow function itu hanyalah penyederhanaan penulisan sebuah function.)

Here is a function written in ES5 syntax:

```
function timesTwo(params) {
 return params * 2
}
timesTwo(4);  // 8
```

Now, here is the same function expressed as an arrow function:

```
var timesTwo = params => params * 2
timesTwo(4);  // 8
```

EXAMPLE : REFACTOR BANDINGKANANGKA FUNCTION WITH ARROW FUNCTION

```
const bandingkanAngka = (angka1, angka2) => {
if (angka1 === angka2 ){
   return "angka sama"
 } else if ( angka1 < angka2){
   return true
 } else{
   return false
 }
}

console.log(bandingkanAngka(5, 8)); // true
```

LATIHAN

Implementasikan function `ubahString` untuk mengganti angka-angka
yang ada di dalam `str` menjadi sebuah huruf yang sesuai dengan aturan:

1 = i

4 = a

3 = e

7 = u

0 = o

Contoh ada di test cases

gunakan arrow function dan atau guard clause untuk latihan ini !

10 menit

```
// Test cases
console.log(numberLetters('prat1w1n7r4m1n1'));
// pratiwinuramini
console.log(numberLetters('y7d1kr1sn4nd1')); // mas yudi
console.log(numberLetters('b4d41')); // badai
```

SPREAD OPERATOR

What does it do? The spread operator allows an expression to be expanded in places where multiple elements/variables/arguments are expected.

```
arr = [1, 2, 3, 4]
arr2 = [...arr, 5, 6, 7]


console.log(arr2);
```

```
let club = {
  id: 1,
  country: 'Italia',
}

let milan =  {
  ...club,
  name: 'AC Milan'
}

console.log(milan)
```

FILTER

The filter() method creates an array filled with all array elements that pass a test (provided as a function).

Note: filter() does not execute the function for array elements without values.

```
let animes = [
  {
    id: 1,
    name: 'Saint seiya'
  },
  {
    id: 2,
    name: 'Naruto'
  },
  {
    id: 3,
    name: 'Nanatsu no Tazai'
  }
]

let anime1 = animes.filter(anime => anime.id !== 3);
console.log(anime1);
```

FOR EACH

The forEach() method calls a provided function once for each element in an array, in order.

Note: forEach() does not execute the function for array elements without values.

```
let animes = [
  {
    id: 1,
    name: 'Saint seiya'
  },
  {
    id: 2,
    name: 'Naruto'
  },
  {
    id: 3,
    name: 'Nanatsu no Tazai'
  }
]

animes.forEach(anime => console.log(anime.name))
```