



Front End React Development sesi 7





Modern JS⁺⁻ ES6

Sesi 7 | Modern JS - ES6

[Variable Declaration]

Di JavaScript ada dua keyword atau kata kunci yang bisa kita gunain untuk mendeklarasikan variable, yaitu: let, dan const. Sebenarnya ada satu lagi keyword yang bisa digunakan untuk mendeklarasikan variable, namanya var. Tapi untuk sekarang kita belum perlu sih pake var. Kita hanya belajar mendeklarasikan variable dengan let dan const saja. Cukup tahu dulu aja kalau kita juga bisa menggunakan var.

Oke, saatnya kita melihat bagaimana cara menulis kode di JavaScript untuk mendeklarasikan variable.

Deklarasi Variabel dengan let

```
1  let playerName = "Budi";  
2  console.log(playerName);  
3  playerName = "Rudi";  
4  console.log(playerName);
```



Sesi 7 | Modern JS - ES6

[Variable Declaration]

Memberi Nama Variabel

Jadi, di JavaScript dan bahasa programming pada umumnya ada istilah yang dinamakan dengan "reserved keyword", atau kata kunci yang sudah "dibooking" oleh bahasa pemrograman kita. Contoh dari reserved word adalah let. Kita tentu tidak bisa menggunakan let sebagai nama variable kita. Kalo kita buat nama variable let, browser akan marah-marah sama kita :)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Keywords



HACKTIV8

Sesi 7 | Modern JS - ES6

[Data Types]

Jenis value ini di dunia pemrograman kita sebut dengan tipe data atau data types. Di JavaScript tipe data yang paling umum digunakan adalah:

- number,
- string,
- dan boolean

Berkenalan dengan tipe data NUMBER

Berkenalan dengan tipe data STRING

Berkenalan dengan tipe data BOOLEAN



HACKTIV8

Sesi 7 | Modern JS - ES6

[Data Types]

Teknik manipulasi tipe data STRING

- menggabungkan string - dengan '+' sederhana

```
1 console.log("I" + " love " + "coding")
```

- menggabungkan string - dengan '+' dalam beberapa baris

```
1 let kalimat = "This"  
2 kalimat = kalimat + " is "  
3 kalimat += "Javascript"  
4 console.log(kalimat)
```

- menggabungkan string - dengan ` (backtick)

```
1 let first = "Modern"  
2 let second = "Javascript"  
3 console.log(`${first} ${second}`)
```



Sesi 7 | Modern JS - ES6

[Array]

Array adalah tipe data terstruktur yang berfungsi untuk menyimpan sejumlah data yang tipenya sama. Bagian yang menyusun **array** disebut dengan isi atau elemen **array**, masing-masing elemen bisa diakses melalui indeks **array**

-- Deklarasi array

```
1 let arr_bil = [1, 2, 3, 4, 5]
2 let arr_str = ["a", "buku", "bisa juga kalimat"]
3 let arr_bol = [true, false, false, true]
```

-- Operasi array - PUSH

```
1 let arr_bil = [10, 29, 2, 3]
2 console.log(arr_bil)
3 arr_bil.push(24)
4 console.log(arr_bil)
```



Sesi 7 | Modern JS - ES6

[Array]

-- Operasi array - SHIFT

```
1  const array1 = [1, 2, 3];
2  const firstElement = array1.shift();
3  console.log(array1);
4  console.log(firstElement);
```

-- Operasi array - UNSHIFT

```
1  const array1 = [1, 2, 3];
2  console.log(array1.unshift(4, 5));
3  console.log(array1);
```

-- Operasi array - SPLICE

```
1  const months = ['Jan', 'March', 'April', 'June'];
2  months.splice(1, 0, 'Feb');
3  console.log(months);
4  months.splice(4, 1, 'May');
5  console.log(months);
```



Sesi 7 | Modern JS - ES6

[Array Multidimensi]

-- Deklarasi array multidimensi

```
1 let arr = [  
2   [1, 2, 3],  
3   [7, 3],  
4   [91, 8, 100, 30]  
5 ]  
6 console.log(arr)
```

-- Operasi - operasi pada array multidimensi adalah sama dengan operasi pada array 1 dimensi, hanya saja kita perlu benar-benar memperhatikan index array yang di operasikan

```
1 let arr = [  
2   [1, 2, 3],  
3   [7, 3],  
4   [91, 8, 100, 30]  
5 ]  
6 console.log(arr)  
7  
8 arr[0].shift()  
9 console.log(arr)  
10  
11 arr[1].unshift(4, 5)  
12 console.log(arr)
```



Sesi 7 | Modern JS - ES6

[this Keyword]

`this` di JavaScript sebenarnya sedikit berbeda dibandingkan dengan bahasa pemrograman lain. `this` memiliki beberapa aturan. Aturan pertama, `this` seringkali merupakan variabel global. Contohnya jika di front-end atau di browser `this` dapat merupakan `window` jika context-nya global.

Atau jika kita eksekusi `this` di terminal via node REPL, `this` adalah global variable dari Node.js.

Aturan kedua, jika `this` berada dalam konteks sebuah object, `this` dipakai untuk memanggil properties yang dimiliki object tersebut.

Aturan ketiga, dalam konteks object-oriented di JavaScript yang akan kita pelajari nanti di bagian-bagian berikutnya, `this` digunakan untuk mendeklarasikan public properties.

```
1 let counter = {  
2   val: 0,  
3   increment: function() {  
4     this.val += 1  
5   }  
6 }
```



Sesi 7 | Modern JS - ES6

[Arrow Function]

Ada beberapa keuntungan menggunakan arrow function

1. Sintaks jadi lebih pendek,
2. Implicit Return,
3. Memudahkan memahami this keyword

```
1 // Traditional Function (no arguments)
2 let a = 4;
3 let b = 2;
4 function (){
5   return a + b + 100;
6 }
7
8 // Arrow Function (no arguments)
9 let a = 4;
10 let b = 2;
11 () => a + b + 100;
```

```
1 // Traditional Function
2 function (a){
3   return a + 100;
4 }
5
6 // Arrow Function
7
8 // Cara 1
9 (a) => {
10   return a + 100;
11 }
12
13 // Cara 2
14 (a) => a + 100;
15
16 // Cara 3
17 a => a + 100;
```

```
1 // Named function, stored as value
2 const tambah = (a, b) => a + b
```



Sesi 7 | Modern JS - ES6

[Class]

Selain class-class yang sudah disediakan oleh JavaScript, sekarang kita pun bisa mendefinisikan class baru, sesuai dengan kebutuhan kita

```
1  class Polygon {  
2    constructor(height, width) {  
3      this.area = height * width;  
4    }  
5  }  
6  
7  console.log(new Polygon(4, 3).area);  
8  
9  // Another way  
10 let pol = new Polygon(3, 3)  
11 console.log(pol.area)  
12
```





--- Destructuring Object

Sesi 7 | Destructuring Object

[Review Object Literal]

So, masih inget ga guys dengan Object Literal ? Kita refresh sedikit yuks, sebelum kita masuk ke materi yang lebih dalam soal object ini

```
1 // Deklarasi cara 1
2 let obj1 = {
3   nama: "Agnes",
4   umur: 20
5 }
6
7 // Deklarasi cara 2
8 let obj2 = {}
9 obj2.nama = "Flavia"
10 obj2.umur: 25
```

```
1 // Menampilkan object
2 console.log(obj1)
3 console.log(obj2)
4
5 // Akses Object
6 console.log(obj1.nama)
7 console.log(obj2.umur)
8
9
10
```



Sesi 7 | Destructuring Object

[Destructuring Object]

ES6 memperkenalkan fitur baru yang namanya destructuring. Untuk memahami apa itu destructuring, mari kita review bagaimana penggunaan Object Literal di JavaScript. Untuk menambahkan properti baru dalam object, kita menggunakan dot notation.

```
1  const user = {};  
2  user.name = 'Adi Nugroho';  
3  user.handle = '@adinugroho';  
4  user.location = 'Jakarta, Indonesia';
```

Dengan menggunakan dot notation seperti disamping kita dipaksa harus menambahkan properti satu per satu. Dot notation juga dapat kita gunakan untuk mendapatkan data dari object literal atau istilah kerennya ekstraksi data.

```
1  const user = {};  
2  user.name = 'Adi Nugroho';  
3  user.handle = '@adinugroho';  
4  user.location = 'Jakarta, Indonesia';  
5  
6  // extraction  
7  const name = user.name;  
8  const handle = user.handle;  
9  const location = user.location;  
10 console.log(name, handle);
```



Sesi 7 | Destructuring Object

[Destructuring Object]

Sekarang, dengan adanya destructuring object, kita dapat menghemat waktu dan kode dengan menuliskan lebih sedikit baris kode, terutama untuk extraction. Tapi, tentu saja, kecermatan dan ketelitian tetap harus temen-temen perhatikan

```
1  const user = {};  
2  user.name = 'Adi Nugroho';  
3  user.handle = '@adinugroho';  
4  user.location = 'Jakarta, Indonesia';  
5  
6  // extraction  
7  const { name, handle, location } = user;  
8  console.log(name, handle, location);
```



Sesi 7 | Destructuring Object

[Destructuring Function Result]

Lanjut, kita pun dapat melakukan destructuring terhadap *return value* dari sebuah *function* yang mengembalikan nilai dalam bentuk object. WOW...

```
1  const getUser = () => {  
2    return {  
3      name: 'Adi Nugroho',  
4      handle: '@adinugroho',  
5      location: 'Jakarta, Indonesia'  
6    };  
7  }  
8  
9  // extraction  
10 const { name, handle, location } = getUser();  
11 console.log(name, handle, location);
```



Sesi 7 | Destructuring Object

[Destructuring Array]

Ada kabar baik gaes. Array pun bisa kita destructuring. Meskipun tidak banyak kasus yang membutuhkannya, destructuring bisa membantu banyak dalam hal ini. Hanya saja, kita tidak menggunakan kurung kurawal ({ }) ya gaes, tapi kita menggunakan kurung siku ([])

```
1 let spec = ["Mercedes Benz", "Mercy", "GLA 200"]
2
3 let [brand, short_name, type] = spec
4 console.log(brand, short_name, type)
```





Callback &⁺ Promise

Sesi 7 | Callback

JavaScript bersifat sekuensial dalam meng eksekusi baris-baris kode. Berikut salah satu contoh untuk memanggil 2 function yang sekuensial

```
1 function myDisplayer(some) {
2   document.getElementById("demo").innerHTML = some;
3 }
4
5 function myCalculator(num1, num2) {
6   let sum = num1 + num2;
7   return sum;
8 }
9
10 let result = myCalculator(5, 5);
11 myDisplayer(result);
12
```

Dengan teknik *callback* kita bisa membuat pemanggilan 2 function tersebut jadi lebih nice...

```
1 function myDisplayer(some) {
2   document.getElementById("demo").innerHTML = some;
3 }
4
5 function myCalculator(num1, num2, myCallback) {
6   let sum = num1 + num2;
7   myCallback(sum);
8 }
9
10 myCalculator(5, 5, myDisplayer);
11
```

Dan dikutip dari w3schools : *Callbacks are working good in asynchronous functions, where one function has to wait for another function (like waiting for a file to load).*



Sesi 7 | Promises

[Darkside of Callback]

Sebelumnya, kita menggunakan callback untuk handle proses asynchronous, namun lama-kelamaan kita akan merasa kesulitan ketika callback yang digunakan semakin banyak, bahkan, bakalan ada callback didalam callback dan seterusnya. Masalah ini sering disebut *callback hell*

```
1 // Callbak HELL
2 fs.readFile('content1.txt', 'utf-8', (err, content1) => {
3   if (err) throw err
4   fs.readFile('content2.txt', 'utf-8', (err, content2) => {
5     if (err) throw err
6     fs.readFile('content3.txt', 'utf-8', (err, content3) => {
7       if (err) throw err
8       fs.readFile('content4.txt', 'utf-8', (err, content4) => {
9         if (err) throw err
10        fs.writeFile('result.txt', content1 + content2 + content3 + content4, err => {
11          if (err) throw err
12          console.log('Writing done!')
13        })
14      })
15    })
16  })
17 })
18
```



Sesi 7 | Promises

[Promise]

Promise adalah salah satu konsep yang hadir di ES6 (ES2015). Dikutip dari beberapa sumber, berikut adalah beberapa penjelasan mengenai Promise :

- Sebuah object yang “mewakili” hasil akhir dari sebuah proses. Hasil ini bisa jadi sebuah keberhasilan atau kegagalan
- Sebuah object yang “dikembalikan”, dimana kita bisa menyambungkannya dengan 1 atau beberapa callbacks. Teknik menyambungkan callbacks nya berbeda dengan teknik yang sebelumnya, dimana kita melakukan pemanggilan callback sebagai parameter dari sebuah function

Promise memiliki 3 buah state :

- Pending (sedang dalam proses)
- Fulfilled (berhasil)
- Rejected (gagal)

Promise memiliki fasilitas *chaining*, yaitu fasilitas dimana kita bisa melakukan pemanggilan beberapa callbacks secara sekuensial, dengan memanfaatkan return value dari callbacks sebelumnya sebagai parameter input dari callbacks selanjutnya

Chaining dapat kita lakukan dengan syntax *then()* yang disediakan oleh object Promise



Sesi 7 | Promises

[Callback vs Promise]

```
1 // Callback style
2 setTimeout(function() {
3   myFunction("I love You !!!");
4 }, 3000);
5
6 function myFunction(value) {
7   document.getElementById("demo").innerHTML = value;
8 }
9
10
11 // Promise style
12 let myPromise = new Promise(function(myResolve, myReject) {
13   setTimeout(function() {
14     myResolve("I love You !!!");
15   }, 3000);
16 });
17
18 myPromise
19   .then(function(value) {
20     document.getElementById("demo").innerHTML = value;
21   });
22
```

```
1 // Callback style
2 doSomething(function(result) {
3   doSomethingElse(result, function(newResult) {
4     doThirdThing(newResult, function(finalResult) {
5       console.log('Got the final result: ' + finalResult);
6     }, failureCallback);
7   }, failureCallback);
8 }, failureCallback);
9
10 // Promise style + arrow function
11 // Note : every function built returning Promise object
12 doSomething()
13   .then(result => doSomethingElse(result))
14   .then(newResult => doThirdThing(newResult))
15   .then(finalResult => {
16     console.log(`Got the final result: ${finalResult}`);
17   })
18   .catch(failureCallback);
19
```

