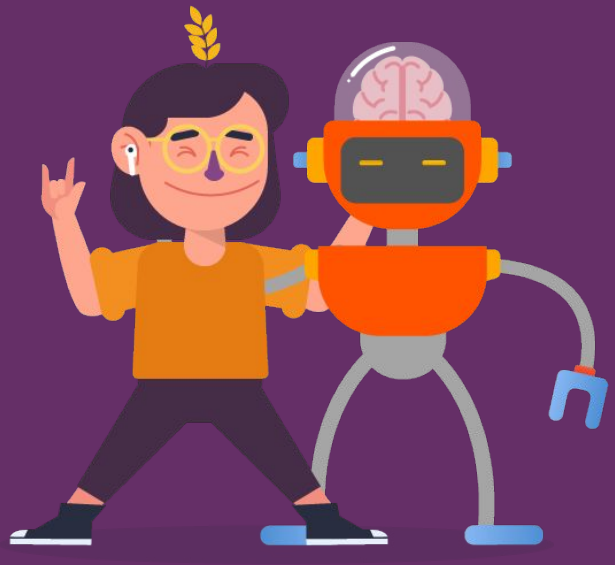
Struktur Data di JavaScript

Gold - Chapter 4 - Topic 1

Selamat datang di Chapter 4 *online course* Full-Stack Web. Selamat belanja~



Eh, salah ya... Maaf, maaf.
Harusnya **BELAJAR**...

Harap maklum yaa, gan. Penulisnya baru nih...

Penulis yang lama sekarang sudah sukses menjadi saudagar.

Udah nggak mau disuruh nulis lagi :(

Ya sudah. Perkenalannya sampai situ saja ya. Lanjut ke bahasan aja deh..

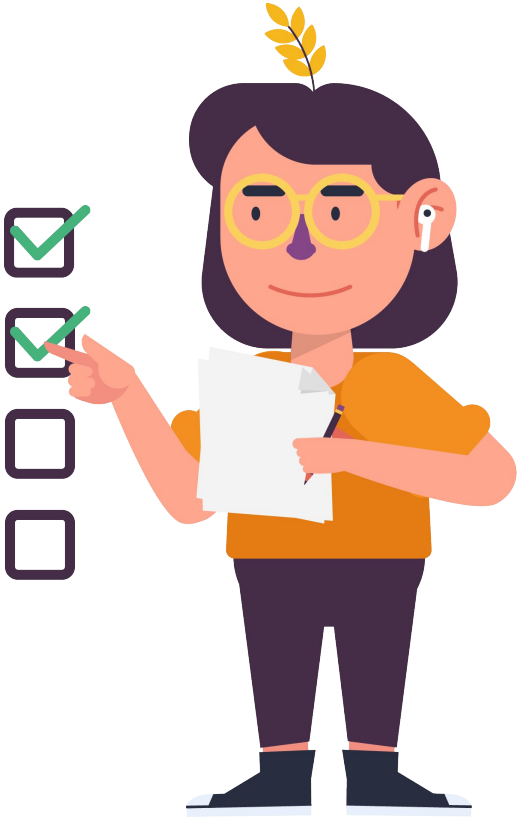
Di **Chapter 4**, kita akan mempelajari bagaimana menulis **Struktur Data di Javascript**.

Khusus di sesi ini, ada tiga hal yang akan kamu pelajari, antara lain:

1. Variabel
2. Tipe data primitif
3. Array

Setelah sesi ini selesai, harapannya kamu bisa mendapatkan beberapa hal, antara lain:

1. Memahami cara menulis variabel di JavaScript
2. Memahami macam-macam tipe data primitif di JavaScript
3. Memahami Array dan penggunaannya di JavaScript



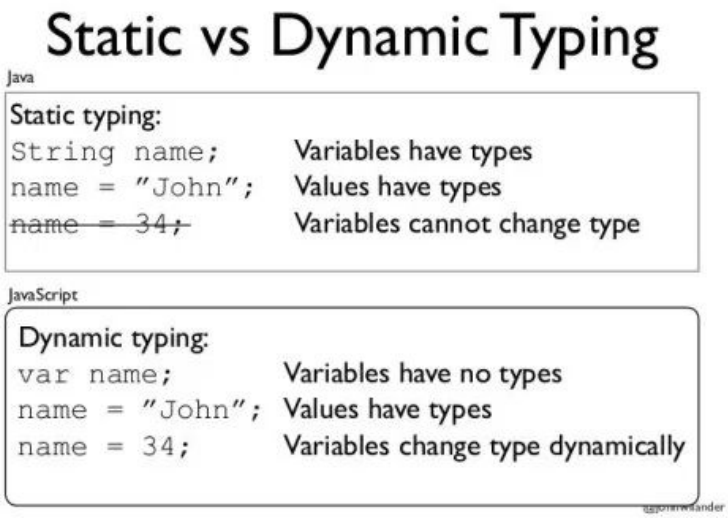
Langsung saja kita mulai!

Apa itu **struktur data**?

Struktur data itu bisa kita anggap sebagai suatu kaidah untuk menyimpan data atau merepresentasikan data di dalam komputer agar bisa digunakan secara efisien.

Nah, pembahasan tentang struktur data nggak akan lengkap kalau kita nggak coba bahas **tipe data** juga.

Mengapa? Karena untuk menerapkan struktur data di JavaScript, kita harus mengetahui tipe data yang akan digunakan.



Masih ingat tipe data yang pernah kita bahas di Chapter 2?

Nah, JavaScript itu termasuk bahasa pemrograman dengan model data ***loosely typed***. Artinya, penentuan tipe datanya dinamis.

Dinamis seperti apa, sih? Maksudnya, tipe data di dalam JavaScript bisa kita ubah-ubah, atau biasa dikenal dengan sebutan ***dynamic typing*.**

Ini beda banget dengan Java. Kenapa? Karena Java menerapkan ***static typing***. Maksudnya, ketika membuat suatu data atau mendeklarasikan sebuah variabel, kita tidak boleh mengubahnya dengan tipe data yang berbeda.

Oh, iya! Sebelum kita bahas lebih dalam tentang tipe data di JavaScript, kita akan kenalan dulu dengan **variabel di JavaScript**!

Di **chapter sebelumnya**, kita udah bahas gimana menyimpan suatu nilai ke dalam variabel, **masih ingat**?

Ketika belanja online, kita pasti membutuhkan informasi tertentu; seperti daftar barang yang tersedia, data barang yang sudah masuk ke keranjang belanja, dan harganya.

Nah, **variabel** dibutuhkan untuk menyimpan informasi-informasi tersebut.

Seperti melamar anak orang, penamaan variabel punya syarat dan ketentuan khusus.

Di Javascript, ada dua aturan dalam penamaan variabel:

1. Nama harus mengandung huruf, angka, atau simbol **“$”** dan **“_”**.
2. Karakter pertama tidak boleh angka

Yang menarik, tanda "**$**" dan "**_**" juga dapat digunakan dalam penamaan, lho!

Kenapa? Karena kedua simbol tersebut tidak memiliki makna khusus di dalam Javascript. Jadi, ia dianggap seperti karakter biasa. Sama seperti alfabet.

```
var $ = 10; // $ sebagai nama variabel
let _ = 100; // _ sebagai nama variabel

console.log($);
console.log(_);
```

```
var namaPengguna = "Sabrina";
let topic1 = "JavaScript Fundamental";
```

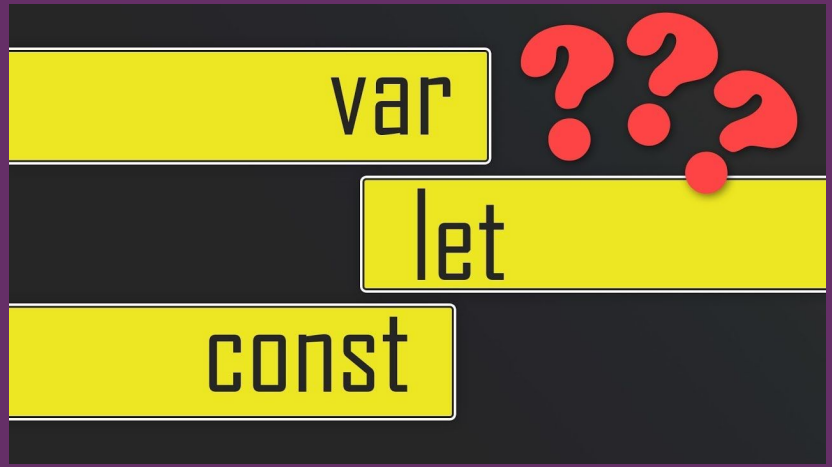
Dalam bahasa sehari-hari, “namaPengguna” seharusnya ditulis terpisah karena terdiri dari dua kata; nama dan pengguna.

Dalam Javascript, kalau kalian mau buat variabel dengan dua kata, maka harus digabung seperti di atas. Ini adalah kaidah **camelCase.**

Di chapter sebelumnya, kita sudah mengenal dua jenis variabel di JavaScript, yaitu **let** dan **const**.

Sebenarnya, JavaScript punya jenis variabel lainnya, yaitu variabel **var**.

Nah, sekarang mari kita bahas perbedaan dari ketiganya agar kita tahu kapan sebaiknya kita menggunakan masing-masing variabel~



Variabel Var

Bisa dibilang, variabel var merupakan variabel yang lawas. Tapi, penggunaannya masih dipertahankan oleh banyak *programmer* untuk menjaga kompatibilitas ke versi sebelumnya.

Dari amatan sekilas, kita bisa tahu bahwa variabel var punya fungsi dan cara penggunaan yang simpel. Tapi, kita perlu berhati-hati dalam menggunakan variabel ini karena ia memiliki beberapa *problem*.

```
/* Buat variabel var dengan cara deklarasi dulu */
var harga; // Declaration
harga = 1000; // Assignment

/* Buat variabel var yang langsung kita kasih nilai */
var harga = 1000
```

```
var diskon = 500 // Global scope
if (true){
 var diskon = 300 // Global scope
}
console.log(diskon)
// Output: 300
// karena var adalah global scope

/* Sebelum ada ES6, solusinya membuat function
scope -> local scope */
var diskon = 500 // Global scope
function diskonScope(){
 var diskon = 300 // Local scope
 console.log(diskon) // Output: 300
}
diskonScope()
console.log(diskon) // Output: 500
```

1. Scope

Scope di dalam JavaScript bisa dikatakan sebagai cakupan/jangkauan program yang ditandai dengan tanda kurung kurawal atau *curly brackets* (**{...}**).

Umumnya, variabel dalam *scope* akan dianggap sebagai ***local scope*** agar tidak bisa dibaca dari *scope* lain.

Tapi, jika kita menggunakan var, maka variabel akan berubah menjadi ***global scope***. Artinya, ia masih bisa diakses meski berada di dalam *scope*.

Nah, biar nggak bingung dan repot, sebaiknya kita gunakan variabel **let** untuk fungsi *scope*.

```
var name; // Declaration
console.log(name) // Output: undefined

name ='Bot' // Assignment
console.log(name) // Output: Bot

var name ='Bot Sabrina' // Redeclared and Reassigned
console.log(name) // Output: Bot Sabrina
```

2. Reassigned dan Redeclared

Variabel var bisa di-*update* nilainya *(reassigned)* dan dideklarasi ulang namanya *(redeclared)*.

Masalahnya, di sini tidak ada pesan error sama sekali ketika terjadi duplikasi variabel. Ini cukup riskan jika kita melakukannya secara tidak sengaja.

Bisa-bisa hasilnya jadi berbeda dengan yang kita inginkan...

```
name = 'Mentor Sabrina' // Variabel di-assign duluan
var name; // Kemudian baru dideklarasikan
console.log(name) // Output: Mentor Sabrina

/* Di belakang layar terjadi hoisting */
var name;
name = 'Mentor Sabrina'
console.log(name) // Output: Mentor Sabrina
```

3. Hoisting

Hoisting ini ibaratnya seperti variabelnya “diangkat” atau dipindahkan ke atas.

Maksudnya gimana? Jika kita *assign* sebuah variabel var lebih dulu, Maksudnya gimana? Jika kita assign sebuah variabel var lebih dulu, JavaScript akan mendeklarasikan variabel tersebut ke atas atau mengangkatnya ke posisi atas di dalam scope.

Hasilnya tidak error, tapi ini akan membuat kita bingung.

Jadi, sebaiknya variabel dideklarasikan di awal sebelum di-*assign*. Untuk lebih jelasnya bisa lihat contoh ini.

Variabel Let

Sejak kehadiran ES6/ES2015 (EcmaScript versi 6 atau EcmaScript yang rilis tahun 2015), JavaScript memperkenalkan variabel ***let*** dan ***const***.

Kode di samping ini adalah contoh penulisan variabel *let* di JavaScript yang pernah kita bahas di Chapter 2.

```
/* Buat variabel yang langsung kita kasih nilai */
let pesan = "Hello World";
console.log(pesan);

/* Buat banyak variabel sekaligus */
let nama = "Sabrina", // dipisahkan dengan koma
umur = 25, // dipisahkan dengan koma
jenisKelamin = "Perempuan";

console.log(nama); // Output: Sabrina
console.log(umur); // Output: 25
console.log(jenisKelamin); // Output: Perempuan/
```

*Catatan: *EcmaScript adalah sebuah standardisasi scripting language (Javascript) yang dibuat oleh sebuah organisasi bernama European Computer Manufacturers Association (ECMA).*

```
let diskon = 500
if (true) { // Tanda awal scope
 let diskon = 300 // Hanya bisa diakses di dalam scope
 console.log(diskon) // Output: 300
} // Tanda akhir scope
console.log(diskon) // Output: 500
```

1. Scope

Jika kita menggunakan ***let,*** maka variabel akan bertindak menjadi ***block scope***. Artinya, variabel hanya bisa diakses di dalam *scope,* yang ditandai dalam sebuah kurung kurawal.

Dengan menggunakan ***let***, kita tidak perlu berurusan lagi dengan *function* untuk membuat *local scope*.

```
let name; // Declaration
console.log(name) // Output: undefined
name ='Bot' // Assignment
console.log(name) // Output: Bot
name ='Bot Sabrina' // Reassigned
console.log(name) // Output: Bot Sabrina
let name ='Mentor Sabrina' // Can not redeclared
console.log(name)

// Output: SyntaxError: Identifier 'name' has already
been declared
```

2. *Reassigned* dan *Redeclared*

Variabel ***let*** bisa di-*update* nilainya (*reassigned*), tapi tidak bisa di deklarasi ulang namanya (*redeclared*). Jadi, akan ada pesan *error* ketika terjadi duplikasi variabel.

Tentu, ini sangat membantu jika kita nggak sengaja melakukan deklarasi ulang variabel.

3. Hoisting

Sebenarnya, variabel ***let*** di JavaScript juga bisa *hoisted.* Seperti contoh pertama ini, jika dijalankan di codesandbox, tidak akan menghasilkan *error.*

Menariknya, jika dijalankan pada *console* di *web browser,* ia akan menghasilkan *error.* Kenapa? Karena *engine* JavaScript nggak bisa mengakses nama variabel sebelum dideklarasikan atau diinisialisasi.

Kondisi ini biasanya disebut dengan istilah ***Temporal Dead Zone*** **(TDZ)**.

```
// Contoh pertama di codesandbox.io
name = "Bot"; // Assignment
let name; // Declaration
console.log(name); // Output: Bot
// Contoh pertama di console web browser
name = "Bot"; // Assignment
let name; // Declaration
console.log(name);
// Output: Cannot access 'name' before
initialization

/* Setelah revisi kode */
let name = "Bot"; // Initialization
console.log(name); // Output: Bot
```

Nah, untuk contoh kedua, penerapan *hoisting* variabel let di dalam *function* juga akan menghasilkan *error*.

Karena di belakang layar sebenarnya JavaScript mendeklarasikan variabel tersebut ke atas tanpa *value*, sehingga akan menghasilkan ***undefined***.

Tapi, ini justru berguna banget buat kita. Dengan begitu, agar kita harus merevisi kode sesuai kaidah yang baik.

Karena itu, sebaiknya sejak awal variabel sudah harus dideklarasikan atau diinisialisasi dengan sebuah *value*.

```
// Contoh Kedua
let message = "Hello"
function greetings(){
 console.log(message)
 let message = "Hello World!"
}
greetings()
// Output: Uncaught ReferenceError: Cannot
access 'message' before initialization

/* Setelah kode direvisi */
let message = "Hello"
function greetings(){
 let message = "Hello World!"
 console.log(message)
}
greetings()
// Output: Hello World!
```

Variabel Const

Variabel *const* berarti nilainya (atau datanya) tidak akan berubah dalam kondisi apapun, karena itulah ia dinamakan konstan.

Dalam mendefinisikan variabel konstan, kita diwajibkan untuk langsung memasukkan nilai dari variabel tersebut.

```
const bumi = "bulat";
const aku = "tamvan";
const pi = 3.14;
```

```
const WARNA_MERAH = "#F00";
const WARNA_BIRU = "#00F";
const WARNA_HIJAU = "#0F0";

/* Ketika kita ingin memanggil warna
  Kita cuma perlu panggil variabelnya saja */
let warnaBaju = WARNA_MERAH;
console.log(warnaBaju);
```

Kelebihan menggunakan **variabel konstan**

- **WARNA_MERAH** lebih gampang untuk diingat, lebih bermakna, dan mudah dibayangkan daripada kode warnanya yaitu **#F00**
- Menghindari *typo*

Uppercase Constants

Kita dianjurkan untuk menggunakan variabel konstan dalam beberapa kasus. Contohnya, penggunaan **alias**, agar kita lebih mudah mengingatnya.

Variabel konstan seperti itu biasanya diberi nama dengan menggunakan **garis bawah (_)** untuk spasi.

Mari kita buat konstanta untuk warna dalam format yang disebut "web" (heksadesimal)

```
const diskon = 500
if (true) { // Tanda awal scope
 const diskon = 300 // Hanya bisa diakses di dalam scope
 console.log(diskon) // Output: 300
} // Tanda akhir scope
console.log(diskon) // Output: 500
```

1. *Scope*

Seperti halnya menggunakan ***let***, jika kita menggunakan ***const*** maka variabel yang ada akan bertindak menjadi *block scope*. Artinya, variabel hanya bisa diakses di dalam *scope,* yang ditandai dalam sebuah kurung kurawal (**{...}**).

Dengan menggunakan *const*, kita tidak perlu berurusan lagi dengan *function* untuk membuat *local scope*.

2. Reassigned dan Redeclared

```
/* Kita gak bisa ubah nilai bumi */
const bumi = "bulat";
bumi = "datar";
// Output: TypeError: invalid assignment to const `bumi'

/* Kita juga gak bisa deklarasi ulang */
const bumi = "datar";
// Output: SyntaxError:Identifier 'bumi' has already been declared
```

Karena ini konstanta, maka kita nggak boleh mengubah nilai dari variabel tersebut. Dengan kata lain, kita gak bisa *reassigned*. Mengapa? karena variabel ini sifatnya *immutable*. Berarti, *value*-nya nggak bisa diubah.

Dan kita juga gak bisa *redeclared* atau mendeklarasikan ulang, lho! Jangan sampai lupa~

Oh, iya! Tapi ini nggak berlaku untuk *object* dan *array*, ya!

Variabel *const* memang nggak bisa kita *reassigned* dan *redeclared*, tapi *property* dalam *object* masih bisa diubah. Ini karena *object* dan *array* dalam JavaScript bersifat *mutable*, yang berarti *value*-nya bisa kita ubah.

```
/* Object dengan variabel const masih bisa kita ubah property-nya */
const obj = { id:1, name:'Sabrina'}
obj.location="Jakarta"
console.log(obj) // Output: { id:1, name:'Sabrina', location:'Jakarta'}
// Tapi, kita gak bisa reassigned
obj={} // Output: TypeError: Assignment to constant variable.

/* Array dengan variabel const masih bisa kita ubah property-nya */
const arr = [1,2,3,4]
arr.push(5)
console.log(arr) // Output: [1,2,3,4,5]
// Tapi, kita gak bisa reassigned
arr=[] // Output: TypeError: Assignment to constant variable.
```

3. Hoisting

Hoisting di variabel const juga sama uniknya seperti variabel let, karena *engine* JavaScript gak bisa mengakses nama variabel sebelum dideklarasikan atau diinisialisasi. Meski begitu, variabel const akan langsung menunjukkan *error*.

Jadi, kita harus menginisialisasi nilai dari sebuah variabel di awal ketika mau menggunakan variabel const.

```
name = "Mentor Sabrina"; // Assignment
const name; // Declaration
console.log(name);
// Output: Uncaught SyntaxError: Missing initializer in const declaration
```

Nah, sekarang kita sudah tau ya perbedaan ketiga jenis variabel di JavaScript dan kapan sebaiknya kita menggunakannya.

	var	const	let
scope	global or local	block	block
redeclare?	yes	no	no
reassign?	yes	no	yes
hoisted?	yes	no	no

Tipe Data Primitif
JavaScript

Nah, masih ingat kan gimana cara cek tipe data dari sebuah data?

Yap! Kita bisa cek tipe data dari sebuah data dengan menggunakan operator **typeof.**

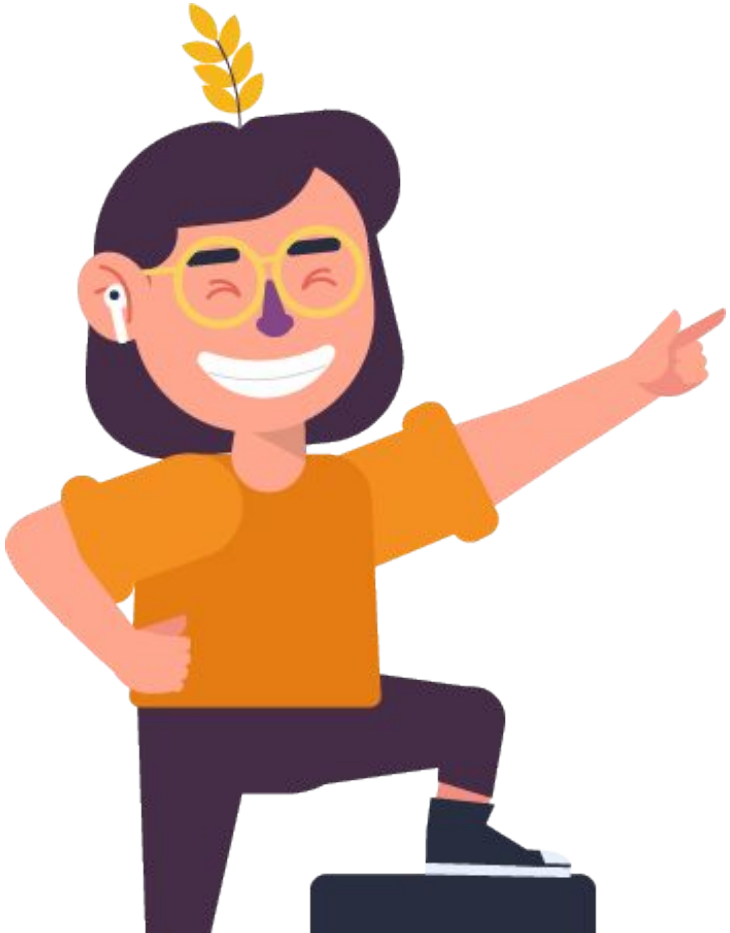
Ada dua cara dalam menulis ***syntax typeof***.

1. Sebagai operator → typeof x
2. Sebagai *function* → typeof(x)

```
let pesan = "Hello World";
console.log(typeof pesan);
// Output: String

console.log(typeof 10);
// Output: Number

console.log(typeof(true));
// Output: Boolean
```



Btw, masih ingat soal data primitif kan?

Ini adalah tipe data secara langsung merepresentasikan suatu nilai.

Biar nggak lupa-lupa-ingat, kita bahas lagi penggunaan **tipe data primitif** yang ada di **Javascript**!

String

Tipe data yang menyimpan karakter sebagai nilainya. Mulai dari huruf, angka, simbol, dan sebagainya. Lalu, nilainya akan diapit oleh tanda kutip.

```
// Kutip dua
const stringBiasa = "Ini string pake kutip dua";
const stringAneh = "Ini string karakter aneh-aneh $!^@&@*&";
const stringPakeAngka = "Ini string yang pake angka 123456";

// Kutip satu
const stringIsiKutipDua =
 'Ini string tapi aku bisa nulis kutip dua disini, liat nih
", tuh kan!';
const stringKutipSatu = 'Gapapa kan kalo pake kutip satu?';

// Backtick
const stringPakeBacktick = `Ini pake kutip kebalik`;
const stringPakeEnter = `
 Aku bisa enter?
 Dih, beneran bisa?
 Bisa cuy!
`;
```

```
const nama = "Sabrina";
const pekerjaan = "Fasilitator Binar";

console.log(`Halo, namaku ${nama}, aku
kerja sebagai ${pekerjaan}`);
// Output: Halo namaku Sabrina, aku kerja
sebagai Fasilitator Binar
```

Template Literal

Di dalam *String*, kita bisa memanggil ulang variabel yang sudah kita deklarasikan dan membuat suatu ekspresi. String yang semacam inilah yang kita sebut dengan ***template literal*.**

Template literal wajib memakai ***backtick*** (`). Untuk menambahkan suatu kode yang akan dieksekusi di dalam String, kita menambahkan karakter seperti ini ${...}.

Selanjutnya, kode-kode yang ada di dalam karakter itu akan dievaluasi oleh JavaScript.

Number

Tipe data number mewakili *number integer* (angka bilangan bulat) dan *floating number* (angka bilangan desimal).

Ada banyak operasi untuk number, contohnya perkalian (*), pembagian (/), penjumlahan (+), pengurangan (-), dan seterusnya.

```
let nilaiUjianKehidupan = 50;
nilaiUjianKehidupan = 95.5;
```

```
console.log( 1 / 0 ); // Output: Infinity
console.log( "Bukan number" / 2 );
// Output: NaN, Pembagian yang salah yaitu
string dan number

/* Jika operasi yg pertama adalah NaN maka
operasi selanjutnya gak bisa di jalankan: */
console.log( "Bukan number" / 2 + 5 );
// Output: NaN
```

Special Numeric Values

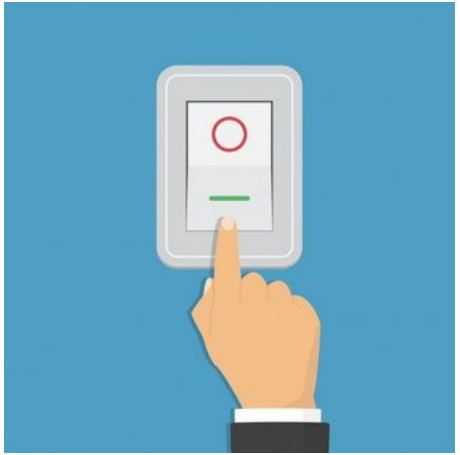
Selain tipe data number biasa, ada juga yang disebut ***special numeric values***, yang termasuk dalam tipe data ini yaitu:

- ***Infinity*:** nilai khusus yang lebih besar daripada angka berapapun jumlahnya. Misalnya, kita bisa mendapatkan nilai infinity ini dari hasil pembagian dengan nol.
- ***NaN (Not a Number)***: kesalahan komputasi. Ini terjadi karena ada operasi matematika yang salah.

Boolean

Tipe data yang hanya memiliki dua nilai saja, ***true*** atau ***false***. Tipe data ini akan sering dipakai pada pengecekan suatu data.

Simpelnya, data ini semacam saklar yang cuma memiliki dua kondisi, nyala atau mati.



```
let apakahSedangTerjadiPandemi = true;
let apakahAkuHarusKeluarRumah = false;
```

Nilai Boolean bisa saja datang dari suatu perbandingan

Perbandingan yang dimaksud di sini adalah ***Logical Operator***

```
let apakahLebihDariDua = 1 > 2;
console.log(apakahLebihDariDua);
// Output: false
```

```
let a = 1;
let b = 1;
let apakahASamaDenganB = a == b;
console.log(apakahASamaDenganB);
// Output: true
```

Null

Tipe data ini hanya mewakili kekosongan. Atau, bisa disebut juga dengan “data yang tidak ada”.

Sebagai contoh, kita mencari buku A di sebuah perpustakaan, dan di perpustakaan itu tidak memiliki indeks dari buku A. Maka ketika kita bertanya kepada Pustakawannya, maka dia bisa menjawab ***null*** atau tidak ada.

```
let bukuA = null;
console.log(bukuA);
// Output: null;
```

BINAR

Undefined

Tipe data yang nggak terdefinisi. Artinya, gak ada sumber nilai pada data tersebut.

Nilai ini biasanya muncul ketika kita mendefinisikan variabel tetapi kita tidak memasukkan nilai ke dalamnya sama sekali.

```
let tidakTerdefinisi;
console.log(tidakTerdefinisi);
// Output: undefined

tidakTerdefinisi = undefined;
console.log(tidakTerdefinisi)
// Output: undefined
```

Array
JavaScript

Object dan ***array*** itu berbeda dengan tipe data primitif. Hayooo, ingat bedanya, nggak?

Object itu sebuah tipe data yang menyimpan koleksi tipe data yang lain. Dia mirip seperti brangkas. Karena itu, untuk memanggil koleksi data tertentu, kita harus menggunakan kunci.

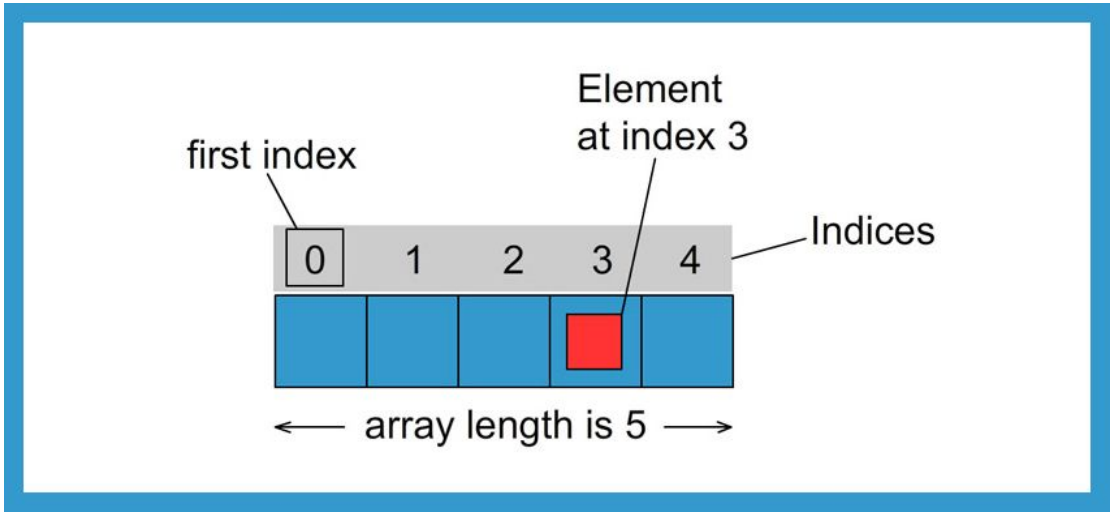
Nah, kunci tersebut kita sebut dengan ***key.*** Sedangkan, besaran/nilai dari *key* disebut dengan ***value***. Dan, sepasang ***key*** dan ***value*** bisa kita sebut sebagai ***property***.

"Trus, kalau array itu apa bhanggg?"

Ia adalah sebuah data yang menyimpan kumpulan tipe data primitif.

Bedanya dengan ***object,*** kalau penyimpanan data di dalam array dipetakan dengan menggunakan yang namanya ***index***.

Oh, iya! Index dalam array dimulai dari 0, **bukan** 1. Ingat-ingat yes~



Object vs *Array*

```
// Object literal
let ktp = {
 nama: "Sabrina",
 alamat: {
   jalan: "Jl. Kabupaten",
   "rt/rw": "05/010"
 },
 pekerjaan: [
    "Dosen UI",
    "Mentor Binar"
 ]
};
console.log(ktp.nama) // Output: Sabrina
console.log(ktp["alamat"]["rt/rw"]) // Output: 05/010
console.log(ktp.alamat["rt/rw"]) // Output: 05/010
// Array di dalam Object
ktp.pekerjaan.forEach(function(item) {
 console.log(`${item} adalah pekerjaan ${ktp.nama}`)
})
/* Output: Dosen UI adalah pekerjaan Sabrina */
/* Output: Mentor Binar adalah pekerjaan Sabrina */
```

```
// Array literal
let namaKeluarga = ["Sabana", "Sabrina", "Sabarna"];
// Array with multiple lines
let namaKartuKeluarga = [
 "Pak Sabana",
 "Mak Sabrina",
 "Nak Sabrina"
];
// Array with new keyword
let namaKK = new Array("Sabana", "Sabrina", "Sabarna");
console.log(namaKeluarga)
/* Output:
 Index 0     Index 1    Index 2
["Sabana", "Sabrina", "Sabarna"]
*/
console.log(namaKartuKeluarga.length) // Output: 3
// Nested Array
const arrays = [namaKeluarga, namaKartuKeluarga]
console.log(arrays)
/* Output: [Array[3], Array[3]] -> ["Sabana", "Sabrina",
"Sabarna"], ["Pak Sabana", "Mak Sabrina", "Nak Sabrina"] */
```

Di JavaScript, array memiliki beberapa method yang bisa kita pakai, misalnya untuk memanipulasi data, ataupun hanya sekedar mencari data di dalam array.

Beberapa method yang disediakan oleh Array, yaitu: ***push, pop, shift, unshift, forEach, filter, map,*** dan ***concat***

Push

Method ini kita pakai untuk menambahkan item yang disimpan ke dalam index terakhir di array.

```
const fruits = ["Apple", "Orange", "Durian"]
fruits.push("Lemon")

console.log(fruits)
// ["Apple", "Orange", "Durian", Lemon"]
```

Pop

Method ini kita pakai untuk mengeluarkan item yang terakhir di array.

```
const fruits = ["Apple", "Orange", "Durian"]
fruits.pop()

console.log(fruits) // ["Apple", "Orange"]
```

Shift

Method ini kita pakai untuk mengeluarkan item pertama di dalam array.

```
const fruits = ["Apple", "Orange", "Durian"]
fruits.shift()

console.log(fruits) // ["Orange", "Durian"]
```

Unshift

Method ini kita pakai untuk menambahkan item di index pertama di dalam array.

```
const fruits = ["Apple", "Orange", "Durian"]
fruits.unshift("Strawberry")

console.log(fruits)
// ["Strawberry", "Apple", "Orange", "Durian"]
```

forEach

Method ini kita pakai sebagai cara singkat melakukan perulangan (looping) di dalam array.

```
const fruits = ["Apple", "Strawberry", "Durian"]

fruits.forEach(function(item) {
 console.log(item)
})

/*
Apple
Strawberry
Durian
*/
```

Filter

Method ini kita pakai untuk melakukan penyaringan dalam suatu array.

```
const numbers = [1,2,3,10,9,2,2,1,4,5]
const filteredItems = numbers.filter(function(item)
{
 return item >= 9
})
console.log(filteredItems) // [9, 10]
```

Map

Method ini kita pakai untuk memanipulasi isi di dalam array tanpa mengubah array aslinya. Map akan membuat array baru dengan melakukan fungsi pada setiap elemen array.

```
const numbers = [1,2,3,4,5]
const mutatedItems = numbers.map(function(i) {
 return i * 2
})

console.log(mutatedItems) // [2, 4, 6, 8, 10]
```

Concat

Method ini kita pakai untuk menggabungkan beberapa array menjadi satu array.

```
const mamals = ["cats", "dogs"]
const birds = ["eagles", "penguins"]

const animals = mamals.concat(birds)
console.log(animals)
// ["cats", "dogs", "eagles", "penguins"]
```

Rangkuman

- **Variabel di JavaScript:** var, let, const
- **Tipe Data Primitif di JavaScript:**
  - **String:** Tipe data yang menyimpan karakter.
  - **Number:** Tipe data yang berupa angka dan dapat melakukan operasi aritmatik.
  - **Boolean:** Tipe data yang melakukan penegasan ya atau tidak.
  - **Null:** Tipe data yang datanya kosong atau tidak ada data.
  - **Undefined:** Tipe data yang tidak terdefinisi, tidak ada nilai pada variabelnya.
- **Method yang disediakan Array:**
  - **push:** untuk menambahkan item yang disimpan ke dalam index terakhir.
  - **pop:** untuk mengeluarkan item yang terakhir di array.
  - **shift:** untuk mengeluarkan item pertama di dalam array.
  - **unshift:** untuk menambahkan item di index pertama di dalam array.
  - **forEach:** cara singkat melakukan perulangan (*looping*) di dalam array.
  - **filter:** untuk melakukan penyaringan dalam suatu array.
  - **map:** untuk memanipulasi isi di dalam array tanpa mengubah array aslinya.
  - **concat:** untuk menggabungkan beberapa array menjadi satu array.

BINAR

Saatnya

QUIZ

BINAR

1

Jenis variabel yang bisa di update nilainya (*reassigned*), tapi **tidak** bisa di deklarasi ulang namanya (*redeclared*) adalah ...

A. var
B. let
C. const

BINAR

2

Manakah yang akan menghasilkan tipe data special numeric values **NaN**?

A. console.log(“Dua puluh”/2);

B. console.log(1/0);

C. console.log(“Halo-halo ” + “Bandung”);

BINAR

3

Method Array yang digunakan untuk **mengeluarkan item yang terakhir** adalah ...

A. push
B. shift
C. pop

BINAR

4

Method Array yang digunakan untuk **melakukan perulangan dengan cara singkat** adalah ...

A. map
B. filter
C. forEach

BINAR

5

Method Array yang digunakan untuk **memanipulasi array tanpa mengubah array aslinya** adalah ...

A. concat
B. map
C. filter

BINAR

Pembahasan

Quiz

BINAR

1

Jenis variabel yang bisa di update nilainya (*reassigned*), tapi **tidak** bisa di deklarasi ulang namanya (*redeclared*) adalah ...

B. let

Variabel let bisa di update nilainya (*reassigned*), tapi **tidak** bisa di deklarasi ulang namanya (*redeclared*). Sedangkan variabel var bisa keduanya, dan sebaliknya kalau variabel const tidak bisa keduanya.

2

Manakah yang akan menghasilkan tipe data special numeric values **NaN**?

A. console.log(“Dua puluh”/2);

Opsi A akan menghasilkan NaN karena hasil pembagian yang tidak tepat, tipe data string dibagi dengan tipe data number. Opsi B akan menghasilkan infinity, sedangkan opsi C akan menghasilkan tipe data string.

3

Method Array yang digunakan untuk **mengeluarkan item yang terakhir** adalah ...

C. pop

push untuk menambahkan item yang disimpan ke dalam index terakhir di array, sedangkan **shift** untuk mengeluarkan item pertama di array.

4

Method Array yang digunakan untuk **melakukan perulangan dengan cara singkat** adalah ...

C. forEach

map untuk memanipulasi isi array, sedangkan **filter** untuk menyaring array.

BINAR

5

Method Array yang digunakan untuk **memanipulasi array tanpa mengubah array aslinya** adalah ...

B. map

filter untuk menyaring array, sedangkan **concat** untuk menggabungkan beberapa array menjadi satu array.

Referensi

- https://www.w3schools.com/js/js_variables.asp
- https://www.w3schools.com/js/js_datatypes.asp
- https://www.w3schools.com/js/js_arrays.asp
- https://medium.com/coderupa/es6-var-let-const-apa-bedanya-1cd4daaee9f0
- https://medium.com/@rivayudha/javascript-hoisting-alasan-kamu-benci-javascript-part-ii-93ed3220630d
- https://medium.com/swlh/deep-and-easy-es6-explanation-part-1-let-and-const-f96ec1d6d332
- https://blog.bitsrc.io/hoisting-in-modern-javascript-let-const-and-var-b290405adfda
- https://dev.to/anwargul0x/javascript-es6-declarations-guide-44a2