

Writing First JavaScript Code

Silver Chapter 1 - Topic 3

Selamat datang di **Chapter 1 Topic 3**
online course **Back End JavaScript** dari
Binar Academy!



Ketemu lagi nih sama Sabrina! 🙌

Sebelumnya, kita sudah mengenal apa itu JavaScript dan alasan kenapa kita perlu mempelajarinya.

Pada Topic 3 ini, kita akan belajar tentang penulisan kode JavaScript dalam pengembangan program.



Detailnya, kita bakal bahas hal-hal berikut ini:

- Comments
- Variable
- Data Types
- Operator



Sebelum dibahas lebih dalam, terlebih dahulu kita akan mencoba untuk menulis dan menjalankan kode JavaScript.

Are you ready, learners? Ayo geser ke slide berikutnya 



Membuat File index.html

1. Buatlah file baru dengan nama index.html
2. Struktur dasar HTML dapat diawali dengan ! di dalam VS Code, lalu pilih dari opsi Emmet untuk menghasilkan kerangka dasar.

Pro Tips :

Saat mengetik !, kita akan melihat opsi auto completion yang memudahkan pembuatan struktur dasar HTML.

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

</body>

</html>
```

Menambahkan Script dalam Body

1. Di dalam tag <body> tambahkan tag <script> untuk menulis kode JavaScript.
2. Kita bisa menuliskan kode JavaScript langsung di dalam tag <script> ini.

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script>

        console.log("Hello world!")

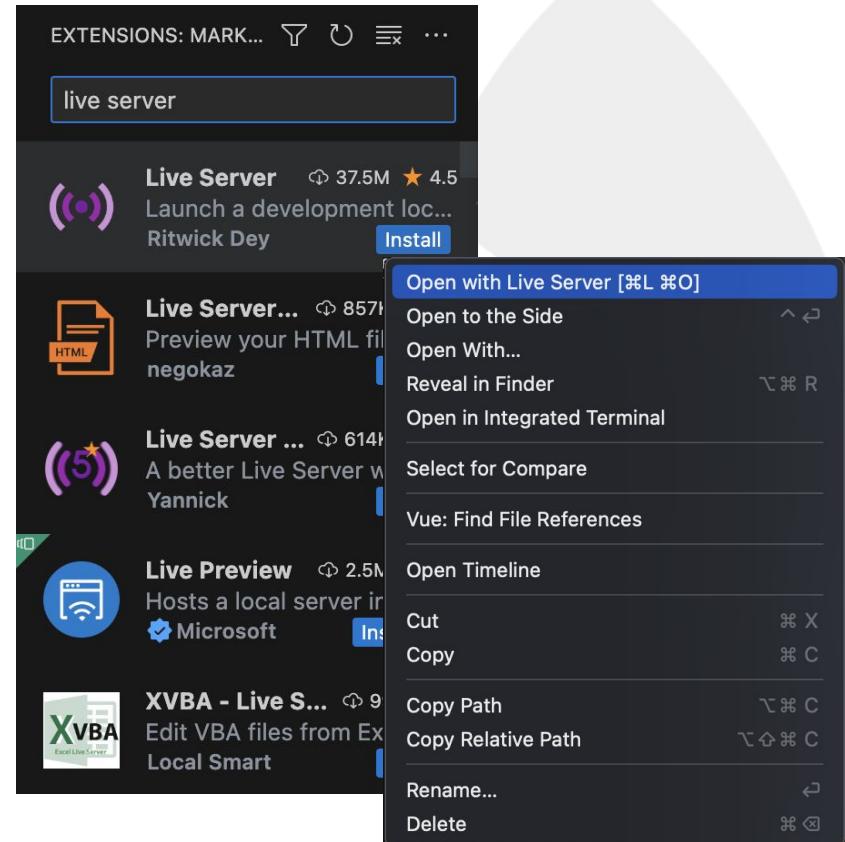
    </script>

</body>

</html>
```

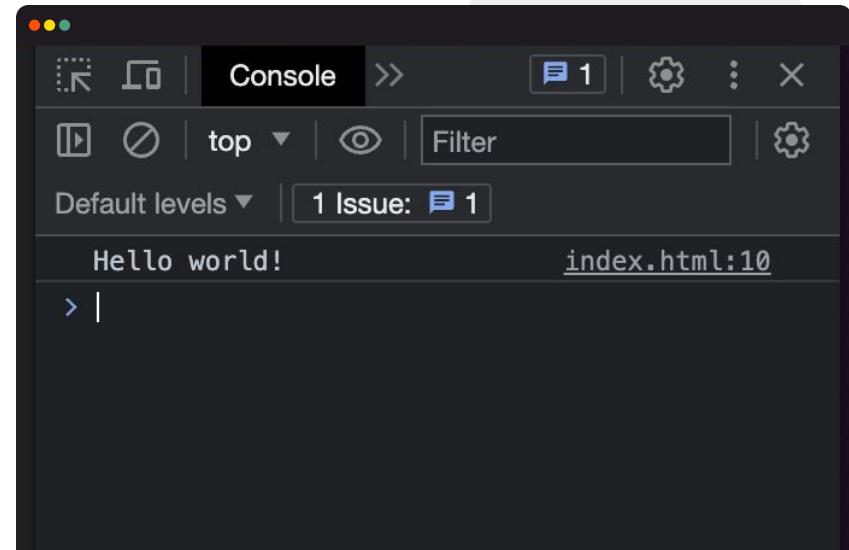
Menjalankan dengan Live Server

1. Install ekstensi Live Server pada VS Code melalui Extensions Marketplace. Ekstensi ini memungkinkan kita untuk menjalankan kode HTML dengan live reload.
2. Klik kanan pada file index.html.
3. Pilih opsi Open with Live Server.
4. Browser akan terbuka otomatis dan menampilkan hasil dari kode HTML dan JavaScript kita.



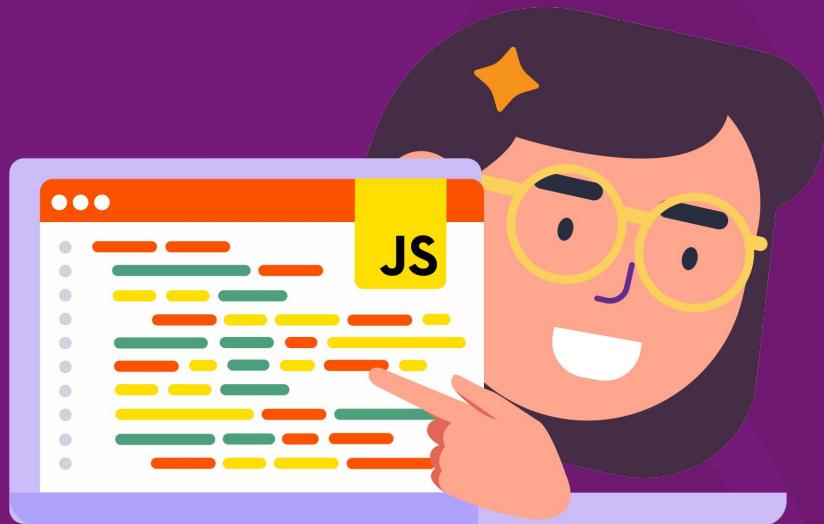
Melihat Output dalam Console Browser

1. Buka konsol browser dengan menekan tombol F12 atau klik kanan lalu pilih opsi Inspect.
2. Pilih tab Console untuk melihat hasil keluaran dari kode JavaScript.
3. Kamu bisa menuliskan `console.log()` dalam kode JavaScript untuk menampilkan informasi di konsol.



FYI, kita juga bisa menuliskan kode JavaScript di dalam file yang terpisah dengan file HTML kita loh.

Kita coba yuk!



Membuat File JavaScript Terpisah

1. Buatlah file baru dengan nama index.js untuk menyimpan kode JavaScript terpisah.
2. Di dalam tag <body>, tambahkan tag <script> dengan atribut src yang mengarahkan ke file index.js.
3. Ini akan menghubungkan kode JavaScript terpisah ke dalam file HTML.

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script src="index.js"></script>

</body>

</html>
```

Gimana sob? Gampang kan?

Berhubung kamu udah tau gimana cara membuat dan menjalankan kode JavaScript, sekarang kita akan membahas Comments.

Langsung aja kita kepoin yuk!



Comments alias Komentar adalah teks dalam kode yang mungkin terlihat tidak terlalu penting, tapi apa itu benar?

Komentar dapat memberikan kita ruang untuk menjelaskan apa yang terjadi di dalam kode kita, mengapa itu dibuat, dan bagaimana kode itu berjalan.

Di Javascript, ada dua cara untuk menuliskan komentar: **Single Line Comments** dan **Multi-line Comments**. Yuk, kita bahas satu per satu!



Single Line Comments

Single Line Comments dimulai dengan //

Selain itu, semua teks yang berada di antara // dan akhir baris akan diabaikan oleh JavaScript dan nggak bakal dieksekusi.

```
// Menghitung luas lingkaran

function luasLingkaran(r) {

    const pi = 22 / 7; // 3.14

    return pi * r * r;

}
```

Multi Line Comments

Sementara itu, **Multi Line Comments** dimulai dengan `/*` dan diakhiri dengan `*/`.

Perlu diingat bahwa semua teks yang ada di antara `/*` dan `*/` tidak akan dijalankan oleh JavaScript ya sob~

```
/*
Fungsi ini menghitung luas dari sebuah lingkaran.

Parameternya adalah:
- r: jari-jari dari sebuah lingkaran

*/
function luasLingkaran(r) {
    const pi = 22 / 7;
    return pi * r * r;
}
```

Perlu dicatat juga, Single Line Comments lebih umum untuk digunakan.

Sementara itu, Multi Line Comments lebih sering dipakai untuk dokumentasi yang lebih formal.



Menggunakan Komentar untuk Mencegah Eksekusi Kode

Komentar juga bisa dipakai untuk mencegah eksekusi kode, hal ini cocok untuk pengujian kode.

Dengan menambahkan `//` di depan baris kode, kita dapat mengubah baris kode dari baris yang dapat dieksekusi menjadi sebuah komentar.

Pro Tips :

kita bisa melakukan seleksi pada kode yang akan di ubah menjadi komentar, lalu tekan `ctrl+{/}` atau `command+{/}`

```
function luasLingkaran(r) {  
    // const pi = 22 / 7;  
    const pi = math.pi;  
    return pi * r * r;  
}
```

Sekarang, kita perluas wawasan kita.

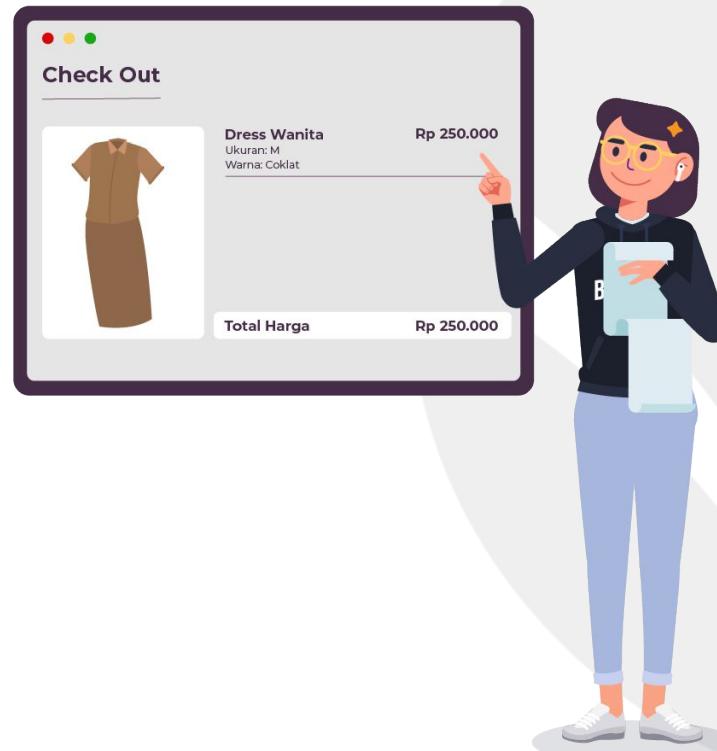
Setelah kita memahami bagaimana komentar digunakan, saatnya kita berpindah ke **Variabel di JavaScript!**

Yuk yuk yuk~



Pas belanja online, kita pasti butuh informasi tertentu kayak daftar barang yang tersedia, data barang yang udah masuk ke keranjang belanja, dan harganya kan?

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



FYI, penamaan variabel punya syarat dan ketentuan khusus lho~

Dalam Javascript, ada dua aturan terkait penamaan variabel seperti berikut:

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



Yang menarik, tanda "\$" dan "_" juga bisa dipakai dalam penamaan, lho! Kenapa begitu?

A dark-themed terminal window icon with three colored dots (red, yellow, green) at the top left.

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

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

Hal ini karena **kedua simbol tersebut nggak punya makna khusus di dalam Javascript**. Jadi, ia dianggap kayak karakter biasa seperti alfabet gitu~



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

Umumnya, penulisan “namaPengguna” harusnya ditulis terpisah karena terdiri dari dua kata, yaitu nama dan pengguna kan?

Nah, dalam Javascript nggak gitu sob. Kalau kamu mau buat variabel dengan dua kata, penulisannya harus digabung seperti di atas. Kaidah penulisan ini biasa disebut sebagai **camelCase**.

Variabel Var~

Bisa dibilang, variabel var merupakan variabel yang lawas.

Tapi, penggunaannya masih dipertahankan oleh banyak **programmer** untuk menjaga kompatibilitas ke versi sebelumnya.

Sekilas, kita bisa tahu kalau variabel var punya fungsi dan cara penggunaan yang sederhana. Tapi, kita perlu berhati-hati ketika memakainya karena ia punya 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
```

Untuk mendeklarasikan variabel, kita dapat memakai keyword seperti:

1. Scope

Dalam JavaScript, Scope merupakan cakupan/jangkauan program yang ditandai dengan tanda kurung kurawal atau curly brackets `{...}`.

Umumnya, variabel dalam scope akan dianggap sebagai **local scope** supaya nggak bisa dibaca dari scope lain.

```
● ● ●

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
```

Kalau kita pakai `var`, maka variabel bakal berubah jadi **global scope**. Artinya, ia masih bisa diakses meski berada di dalam scope.

Biar nggak bingung dan repot, sebaiknya kita gunakan variabel **let** buat fungsi scope.

```
● ● ●

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
```

2. Reassigned dan Redeclared

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

Masalahnya, di sini nggak ada pesan error sama sekali pas terjadi duplikasi variabel. Hal ini cukup berisiko jika kita melakukannya tanpa sengaja.

Bisa-bisa hasilnya jadi berbeda sama yang kita inginkan~

```
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
```

3. Hoisting

Ibaratnya kayak variabel “diangkat” atau dipindahkan ke atas. Maksudnya gimana?

Begini, kalau kita assign sebuah variabel var lebih dulu, JavaScript bakal mendeklarasikan variabel tersebut ke atas atau mengangkatnya ke posisi atas di dalam scope.

Hasilnya nggak error, tapi ini bakal bikin kita bingung. Jadi, **lebih baik variabel dideklarasikan di awal sebelum di-assign.**

```
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
```

Variabel Let~

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

Contoh penulisan variabel let dapat kamu lihat pada gambar di samping.

Sebagai catatan, [**EcmaScript**](#) merupakan sebuah standardisasi scripting language (Javascript) yang dibuat oleh sebuah organisasi bernama European Computer Manufacturers Association (ECMA).

```
/* 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/
```

Sama seperti Var yang memiliki keyword untuk mendeklarasikan variabel, variabel let juga memiliki keyword sendiri seperti berikut:

1. Scope

Kalau kita memakai **let**, variabel akan bertindak menjadi **block scope**. Artinya, variabel cuma bisa diakses di dalam scope yang ditandai dalam sebuah kurung kurawal.

Dengan menggunakan **let**, kita nggak perlu berurusan lagi dengan function buat bikin local scope.

```
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
```

2. Reassigned dan Redeclared

Variabel **let** bisa di-update nilainya (reassigned), tapi nggak bisa di deklarasi ulang namanya (redeclared).

Jadi, bakal ada pesan error pas terjadi duplikasi variabel.

Tentu saja hal ini sangat membantu ketika kita nggak sengaja melakukan deklarasi ulang variabel.



```
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 redeclare
console.log(name)

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

3. Hoisting

Sebenarnya, variabel **let** di JavaScript juga bisa hoisted.

Menariknya, jika dijalankan pada console di web browser, ia bakal menghasilkan error karena engine JavaScript nggak bisa mengakses nama variabel sebelum dideklarasikan atau diinisialisasi.

Kondisi ini biasa 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
```

Untuk contoh kedua, penerapan hoisting variabel let di dalam function juga bakal menghasilkan error.

Hal ini karena di belakang layar, sebenarnya JavaScript mendeklarasikan variabel tersebut ke atas tanpa value sehingga menghasilkan **undefined**.

Tapi, kondisi ini justru berguna banget buat kita lho. Soalnya, kita bisa merevisi kode sesuai kaidah yang baik.

Karena itu, ada baiknya 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~

Seperti namanya, variabel ini berarti **nilainya (atau datanya) nggak bakal berubah dalam kondisi apapun.**

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



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

Uppercase Constants~

Dalam beberapa kasus, kita dianjurkan untuk memakai variabel konstan. Contohnya penggunaan **alias** supaya kita lebih mudah mengingatnya.

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

Selanjutnya, mari kita buat konstanta untuk warna dalam format yang disebut "web" (heksadesimal).

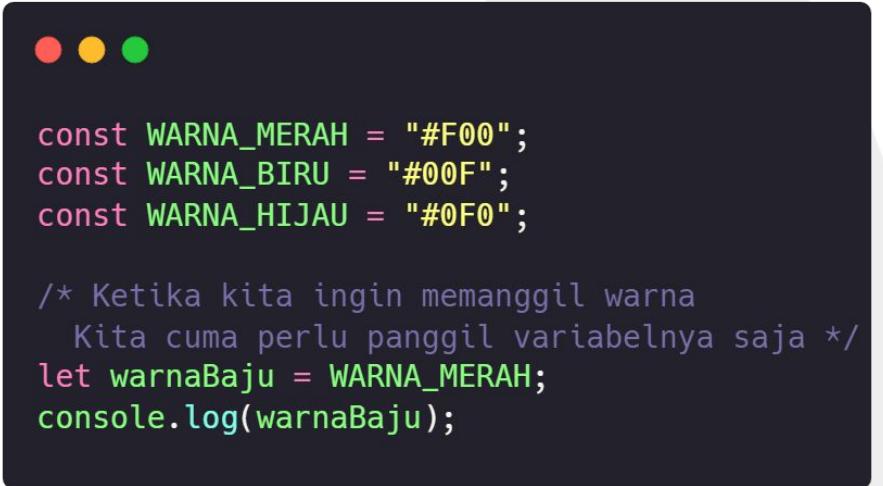


```
const WARNA_MERAH = "#F00";
const WARNA_BIRU = "#00F";
const WARNA_HIJAU = "#0F0";

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

Oh iya, ada beberapa kelebihan ketika memakai **variabel konstan**, seperti:

- **WARNA_MERAH** lebih gampang buat diingat, lebih bermakna, dan mudah dibayangkan daripada kode warnanya yaitu **#FOO**.
- Menghindari typo.



A screenshot of a Mac OS X desktop. In the top-left corner, there are three colored window control buttons: red, yellow, and green. Below them is a dark terminal window. The terminal contains the following code:

```
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);
```

Selain var dan let, variabel konstan juga memiliki keyword-nya sendiri, seperti:

1. Scope

Seperti halnya menggunakan **let**, ketika memakai **const**, variabel yang ada bakal bertindak jadi block scope.

Artinya, variabel cuma bisa diakses di dalam scope, yang ditandai dalam sebuah kurung kurawal **{...}**.

Dengan menggunakan const, kita nggak perlu berurusan lagi dengan function buat bikin local scope.

```
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
```

2. Reassigned dan Redeclared

Oleh karena ini konstanta, kita nggak boleh mengubah nilai dari variabel tersebut. Dengan kata lain, kita nggak bisa reassigned karena variabel ini sifatnya immutable. Ini berarti value-nya nggak bisa diubah.

```
/* 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
```

Perlu diingat juga bahwa kita nggak bisa redeclared atau mendeklarasikan ulang. Jadi, jangan sampai lupa ya!

But wait~ ketentuan tadi nggak berlaku untuk object dan array, ya!

Variabel const memang nggak bisa kita reassigned dan redeclared. Tapi, property dalam object-nya masih bisa diubah. Hal 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 nggak bisa mengakses nama variabel sebelum dideklarasikan atau diinisialisasi. Meski begitu, variabel const bakal 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 udah tahu ya perbedaan ketiga jenis variabel di JavaScript dan kapan sebaiknya kita menggunakaninya. Biar kamu makin ingat, tabel berikut adalah rangkuman dari perbedaan ketiga jenis variabel tadi. Kita pahami dulu yuk!

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

Sekarang kamu udah kenal variabel Javascript kan?

Selanjutnya kita bakal bahas lebih dalam tentang **Tipe Data di JavaScript** sebagai bagian dari struktur data.

Kuy nggak nih? Kuy dong~

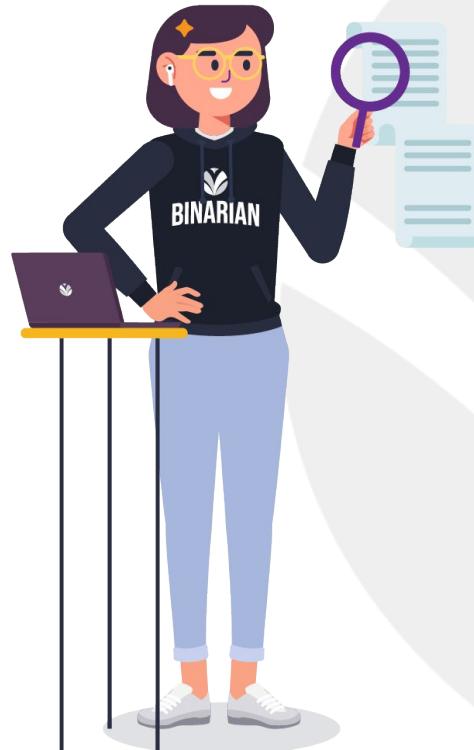


Kelompok tipe data di Javascript

Di dalam Javascript, ada 2 kelompok tipe data, yaitu: data primitif dan data nonprimitif. Perbedaan keduanya seperti berikut ya gengs:

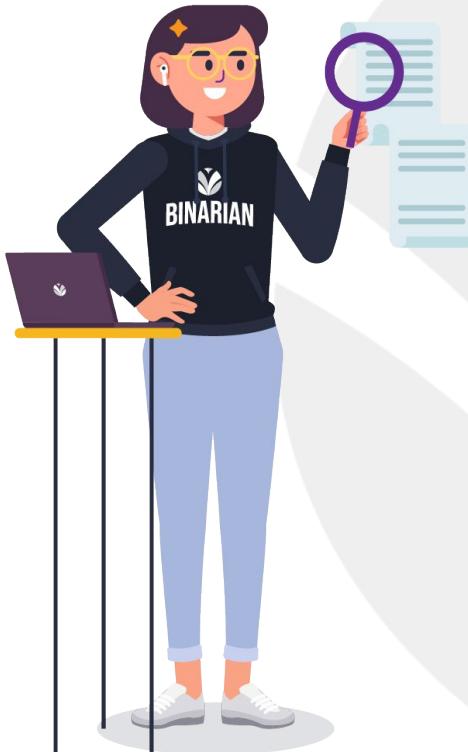
1. **Data primitif**, yakni jenis data yang sudah disediakan langsung oleh Javascript dan bisa secara langsung merepresentasikan suatu nilai.

Contoh: String, Number, Boolean, Null, Undefined



2. **Data nonprimitif**, yakni data yang dibuat sendiri oleh developer, yang mana data ini bisa menyimpan beberapa nilai pada variable.

Contoh: Array, dan Object.



Untuk mengetahui tipe data, kita bisa melakukan checking codingan yang ada nih. Caranya dengan memakai operator **typeof**.

Dalam menulis **syntax typeof**, ada 2 cara seperti berikut:

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

Contoh penggunaannya bisa kamu cek pada gambar di samping ini ya~



```
● ● ●

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

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

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

Sekarang, kita bahas tipe-tipe data, mulai dari tipe data primitif

1. String

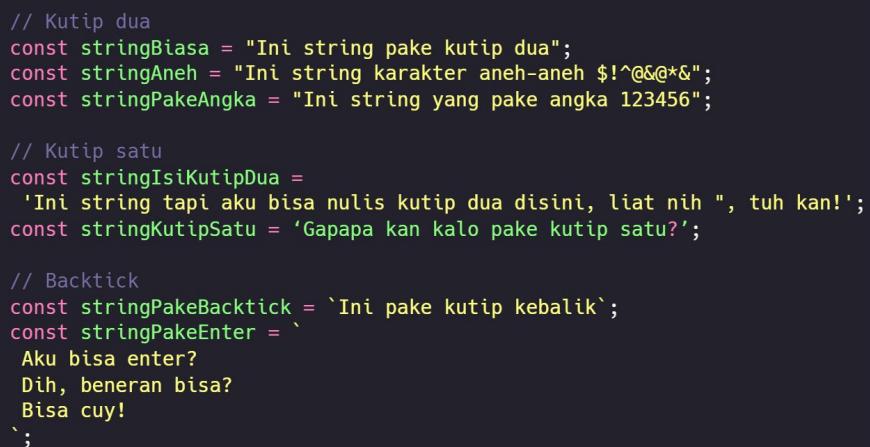
Tipe data ini **menyimpan karakter sebagai nilainya**. Bisa dalam format huruf, angka, simbol, dan sebagainya.

Karakter-karakter tersebut bakal diapit dengan tanda kutip (' atau "').



Nah, hasil dari data string tadi bisa ditampilkan sesuai dengan karakter yang dituliskan di antara tanda kutip (' atau "').

Agar lebih jelas, kamu bisa melihat contohnya pada gambar di samping.



The screenshot shows a dark-themed code editor window with three circular status indicators at the top left. The code itself is written in JavaScript and demonstrates different ways to handle strings:

```
// 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!
`;
```

Di dalam string, kita bisa **memanggil ulang variabel yang sudah dideklarasikan untuk membuat ekspresi tertentu**. String yang seperti ini biasa disebut sebagai template literal.

FYI, **template literal wajib memakai backtick (`)**. Untuk menambahkan suatu kode yang mau dieksekusi di dalam string, kita bisa menambahkan karakter seperti ini \${...}.

Selanjutnya, kode-kode yang ada di dalam karakter itu akan dijalankan oleh Javascript.

```
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
```

2. 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 sebagainya.



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

Selain tipe data number biasa, ada juga tipe data yang disebut **special numeric values**, seperti:

- **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. Hal ini terjadi karena ada operasi matematika yang salah.



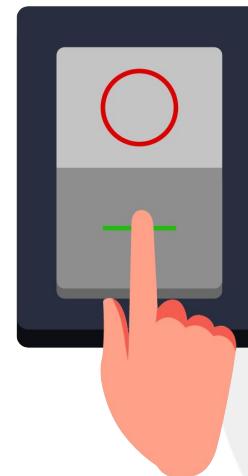
```
console.log( 1 / 0 ); // Output: Infinity
console.log( "Bukan number" / 2 );
// Output: NaN, Pembagian yang salah yaitu string dan number

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

3. Boolean

Tipe data boolean **cuma punya dua nilai**, yaitu **true** atau **false**. Tipe data ini bakal sering dipakai di pengecekan suatu data.

Tipe data ini bisa diibaratkan sebagai saklar yang cuma punya dua kondisi, yakni menyala atau mati.



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

Nilai Boolean bisa aja datang dari suatu perbandingan. Perbandingan yang dimaksud di sini adalah **Logical Operator**. Untuk memahaminya, perhatikan gambar di bawah.



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



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

4. Null

Tipe data ini cuma mewakili kekosongan. Jadi, kita juga bisa menyebutnya sebagai **“data yang nggak ada”**.

Contoh: kita mencari buku A di sebuah perpustakaan dan di perpustakaan itu nggak punya indeks dari buku A.

Ketika kita bertanya kepada pustakawannya, dia bisa menjawab **null** atau nggak ada.



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

5. Undefined

Tipe data ini adalah tipe data yang nggak terdefinisi. Artinya, **nggak ada sumber nilai pada data tersebut**.

Nilai ini biasanya muncul pas kita mendefinisikan variabel tetapi kita nggak memasukkan nilai ke dalamnya sama sekali.



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

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

FYI, object dan array itu berbeda dengan tipe data primitif ya gengs!

Ini dia perbedaannya:

1. Object

Object merupakan sebuah **tipe data yang menyimpan koleksi tipe data yang lain**, ibaratnya kayak brankas gitu.

Jadi, untuk memanggil koleksi data tertentu, kita harus memakai kunci.



```
const object = {  
    hello: 'world'  
}
```

Kunci tersebut adalah **key**, sedangkan besaran nilai dari key disebut sebagai **value**. Sementara itu, sepasang key dan value bisa disebut sebagai **property**.

Gambar di samping memuat ketiga istilah di atas. Secara rinci, key di gambar adalah hello, sedangkan value-nya adalah world.



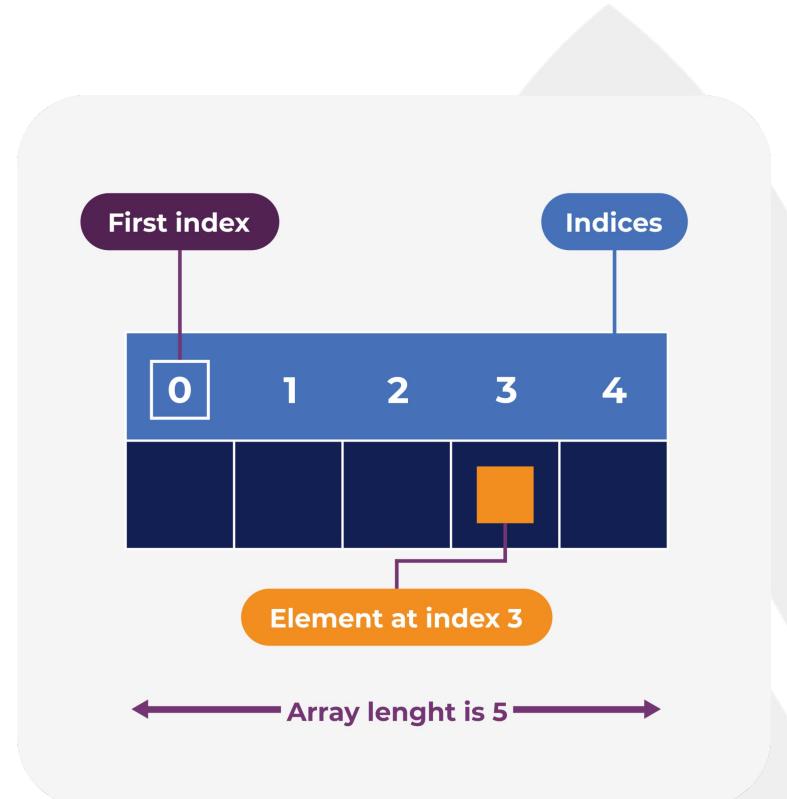
```
const object = {
    hello: 'world'
}
```

2. Array

Array merupakan sebuah data yang menyimpan kumpulan tipe data primitif.

Berbeda dengan object, penyimpanan data di dalam array **dipetakan dengan index**.

Nah, index dalam array dimulai dari 0, **bukan** 1.
Ingin-ingat yes~



Ini dia perbedaan object vs array. Kira-kira, yang mana ya contoh object dan array?

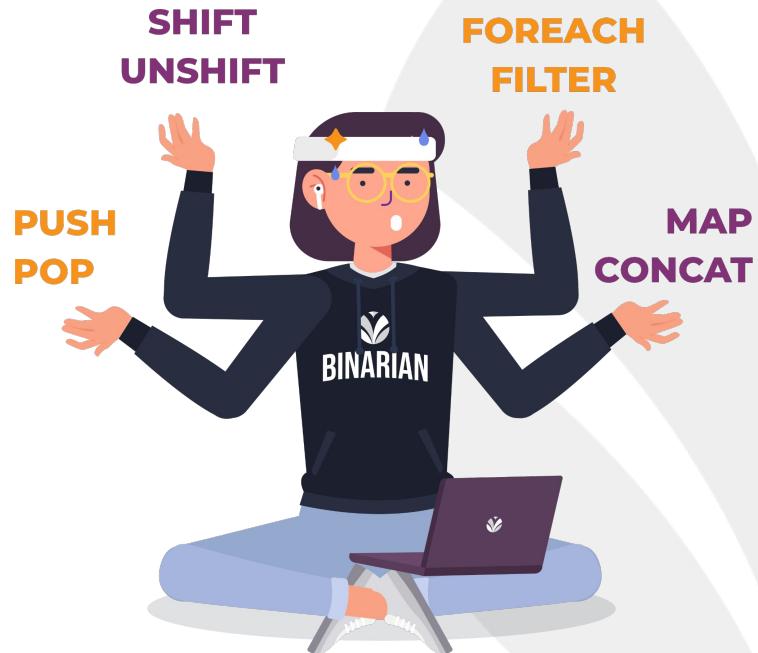
```
● ● ●  
  
// Object literal  
let ktp = {  
  nama: "Sabrina",  
  alamat: {  
    jalan: "Jl. Kabupaten",  
    "rt/rw": "05/010"  
  },  
  pekerjaan: [  
    "Dosen UI",  
    "Mentor Binar"  
  ]  
};  
console.log(ktp.nama) // Output: Sabrina  
console.log(ktp["alamat"]["rt/rw"]) // Output: 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 metode yang bisa dipakai, misalnya untuk memanipulasi data atau hanya sekedar mencari data di dalam array.

Beberapa metode yang disediakan oleh Array antara lain: **push**, **pop**, **shift**, **unshift**, **forEach**, **filter**, **map**, dan **concat**

Kita bahas dan kasih contohnya satu-satu yaaa!



Push

Metode ini dipakai 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

Metode ini dipakai untuk mengeluarkan item yang terakhir di array.



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

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

Shift

Metode ini dipakai untuk mengeluarkan item pertama di dalam array.



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

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

Unshift

Metode ini dipakai 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

Metode ini dipakai sebagai cara singkat untuk melakukan perulangan (looping) di dalam array.



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

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

/*
Apple
Strawberry
Durian
*/
```

Filter

Metode ini dipakai 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

Metode ini dipakai 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

Metode ini dipakai untuk menggabungkan beberapa array menjadi satu array.



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

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

Pas sekolah, kita belajar banyak operator di pelajaran matematika, mulai dari penjumlahan (+), perkalian (*), pengurangan (-), dan operator-operator lainnya.

Supaya semakin kaya ilmunya, kali ini kita bakal fokus membahas aspek operator yang nggak kita dapat di bangku sekolah~



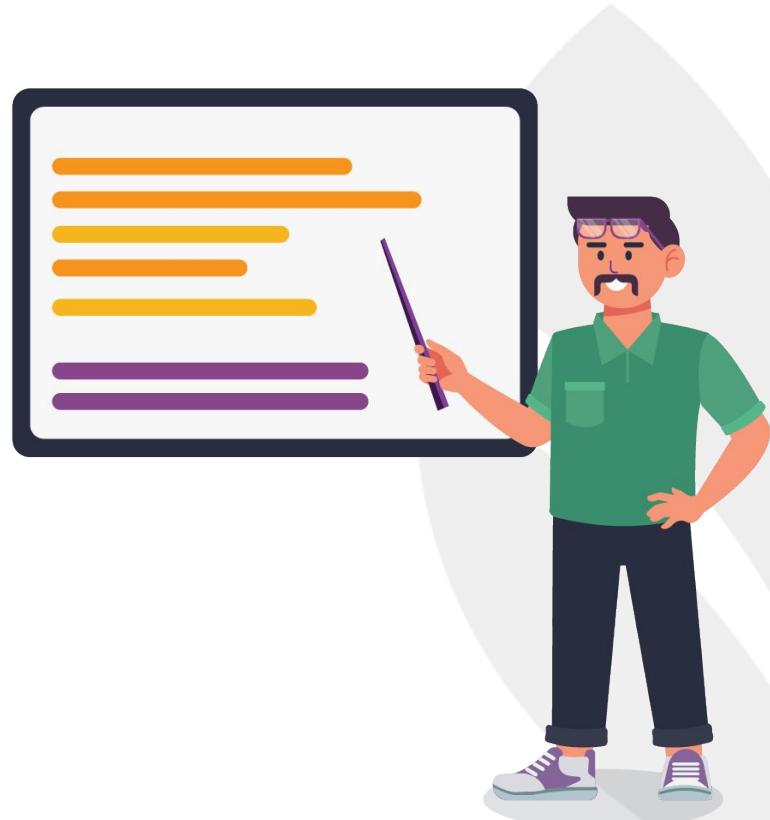
Sebelum masuk ke pembahasan, kita pemanasan dulu yuk. Kita coba pecahkan dulu apa aja yang ada di dalam sebuah **operasi**.

Operan adalah pelaku dari operasi itu.

Contoh: **2, 3, 4, 10**

Operator berfungsi memberi tahu operan apa yang harus dilakukan.

Contoh: **+, -, *, %**



FYI, **Operasi** ada dua jenis nih sob.

Pertama **Unary**, yaitu suatu operasi yang cuma punya satu buah operan.

Kalau kamu cek di gambar, $-x$ adalah unary-nya.

```
let x = 1;  
x = -x; // - (minus) di depan x kita sebut dengan unary  
console.log(x)
```

Output:
-1

Kedua adalah **Binary**.

Binary adalah suatu operasi yang punya lebih dari satu operan.

Contohnya kayak `x + y`, yang tertulis dalam console.



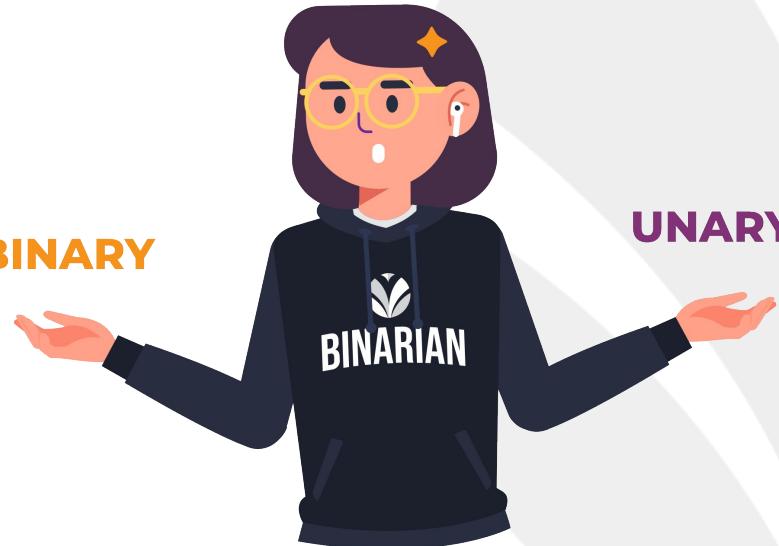
```
let x = 50;  
let y = 100;  
console.log(x + y);
```

Terus nih, dalam perangkaian string, **operator plus (+)** punya dua bentuk, yaitu bentuk **binary** dan **unary**.

Kita bahas satu-satu, yuk!

BINARY

UNARY



Binary + pada perangkaian string~

Ternyata, **operator plus (+)** itu nggak cuma bisa dipakai dalam operasi aritmatika saja lho. Ia juga bisa dipakai untuk **concatenation (perangkaian)** sebuah string.

Dalam string, operator plus (+) fungsinya bukan untuk menjumlahkan, melainkan untuk **menyatukan**.

Biar makin paham, coba deh kamu cek gambar di samping. Apa nih yang bisa kamu pahami dari gambar di samping?



```
let a = "Hello";
let b = "World";
console.log(a + b);
```



Output:

HelloWorld

Sebagai tambahan, **jika salah satu operan adalah string, operan lainnya bakal ikut dikonversi menjadi string juga.**

Kayak istilah “pokoknya, kemana pun kamu pergi, aku ikut!” 😅



```
/* Jika salah satu operan adalah string  
Maka operan lain yang berinteraksi dengannya  
akan dianggap String juga */  
console.log("1" + 2);
```



Output:
"12"

Sebagai catatan, kita juga harus memastikan bahwa operasinya berjalan **dari kiri ke kanan**.

Kalau ada **dua angka diikuti oleh string**, angka-angka itu bakal **ditambahkan terlebih dahulu** sebelum dikonversi jadi string.



```
console.log(2 + 2 + "1");
```



Output:

"41"

Unary + (numeric conversion)

Operator plus (+) dalam unary diterapkan di nilai tunggal dan kelihatannya nggak memberi aksi apapun di angka.

Tapi, kalau operan bukan angka, plus unary bakal mengubahnya jadi angka.

Terus, untuk mengkonversi string jadi number, kita bisa memakai function **Number("12345")**. Tapi, cara di atas tetap lebih singkat dan mudah buat dipraktekan.



```
/* Gak memiliki efek apapun
   Kalo diterapin ke angka */
let x = 1;
console.log(+x); // Output: 1
let y = -2;
console.log(+y); // Output: -2

/* Namun kalo diterapkan di tipe data selain angka */
let z = true;
console.log(+z); // Output: 1
let w = "";
console.log(+w); // Output: 0
```

Udah tau belum?

Konversi dari string jadi number bakal sering banget kita jumpai ketika bikin form HTML lho! Kenapa begitu?

Karena eh karena~

Bakal memungkinkan banget kalau data hasil input dari form HTML bisa berupa string, gan 😊

Contoh yang salah

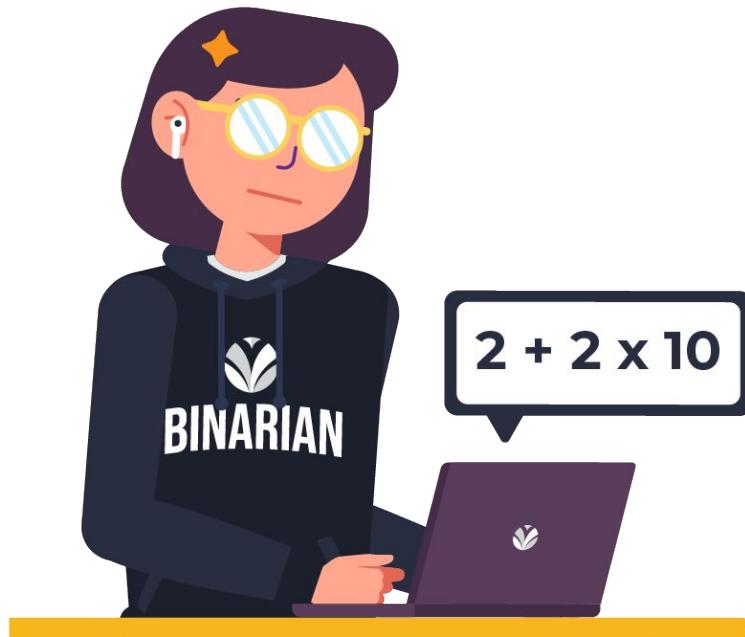
```
let a = "1";
let b = "2";

console.log(a + b); // Output: 12
```

Contoh yang benar

```
let a = "1";
let b = "2";

// a dan b kita konversi dulu menjadi angka
console.log(+a + +b); // Output: 12
```



Tingkatan Prioritas Operasi~

Sama seperti matematika yang diajarkan di sekolah, setiap operasi memiliki tingkat prioritas terkait mana yang harus dikerjakan dulu.

Contoh: **$2 + 2 * 10$**

Dari operasi di atas, bisa dipastikan kalau operasi yang bakal dijalankan duluan adalah **$2 * 10$**

Kalau di programming, prioritasnya gimana?

Ada banyak operator dalam bahasa pemrograman Javascript. Setiap operator ini punya urutan prioritas yang sesuai, di mana angka yang paling tinggi bakal dieksekusi duluan.

Terus, kalau prioritas pengurutannya adalah sama, maka urutan eksekusi dilakukan dari kiri ke kanan.

Unary plus memiliki prioritas **17**, yang mana prioritasnya lebih tinggi dari "penambahan" (binary plus) yang bernilai **13**.

Prioritas	Nama	Tanda
17	Unary plus	+
17	Unary negation	-
15	perkalian	*
15	pembagian	/
13	penambahan	+
13	pengurangan	-
3	sama dengan	=

Assignment (Sama Dengan “ = ”)~

Assignment adalah penentuan nilai dari sebuah data. Ketika kita bilang **let a = 1**, berarti kita sudah melakukan sebuah assignment terhadap **variabel "a"**.

Dengan kata lain, kita menginformasikan bahwa **variabel "a" bernilai "1"**.



```
let x = 2 * 2 + 1;  
console.log(x); // Output: 5
```

Selain assignment seperti tadi, kita juga bisa banget nih bikin **assignment berantai** seperti gambar di bawah. Gimana, keren, kan?!



```
let a, b, c;
a = b = c = 2 + 2;

/* Assignment berantai mengevaluasi dari kanan ke kiri. Pertama, ekspresi paling kanan 2 + 2 dievaluasi kemudian ditetapkan ke variabel di sebelah kiri: c, b dan a. Pada akhirnya, semua variabel berbagi nilai tunggal. */
console.log(a); // Output: 4
console.log(b); // Output: 4
console.log(c); // Output: 4
```

Eksponensial (Perpangkatan)~

Operator ini berfungsi untuk membuat suatu perpangkatan seperti di matematika. Tapi, kalau kita ingin menulis perpangkatan/eksponen di dalam Javascript, kita bisa menulisnya seperti gambar di bawah.



```
2**4 // Ini cara menulis perpangkatan di Javascript
console.log(2**4) // Output: 16
```

Increment dan Decrement (Peningkatan atau Penurunan)~

Keduanya berfungsi untuk menaikkan atau menurunkan satu angka dari suatu variabel. FYI, operator ini cuma berlaku buat variabel aja ya sob.

Kalau kamu ingin menambahkan operator ini di sebuah angka secara langsung, nantinya bakal menyebabkan error. Tetep hati-hati ya, gengs!



```
//Increment
let a = 2;
a++; // Nilai a akan menjadi a + 1
console.log(a); // Output: 3

//Decrement
let b = 3;
b--; // Nilai b akan menjadi nilai b - 1
console.log(b); // Output: 2
```

Lebih lanjut, operator increment dan decrement bisa ditaruh **sebelum (prefix)** ataupun **sesudah (postfix)** variabel.

Seperti yang kita praktikkan tadi, setiap operasi pasti mengembalikan suatu nilai.

Nantinya, akan ada perbedaan antara menggunakan prefix dengan postfix dalam pengembalian nilai dari operasi tersebut.

Lebih jelasnya bisa kamu cermati pada gambar di samping ya sob~



Prefix

```
let a = 2;
console.log(a++); // Operasi a++ akan mengembalikan nilai a yang asli
// Output: 2
console.log(a);
// Output: 3
```



Postfix

```
a = 2; // Mengatur ulang nilai a
console.log(++a); // Operasi ++a akan mengembalikan hasil dari operasi
// Output: 3
console.log(a);
// Output: 3
```

Bitwise~

Operator ini memperlakukan argumen sebagai angka **integer 32-bit** dan bekerja di level representasi binernya. Nggak cuma untuk Javascript, bitwise juga mendukung sebagian besar bahasa pemrograman.

Berhubung operator ini jarang banget dipakai, kita nggak bakal membahasnya lebih lanjut di materi ini.

Tapi, kalau kamu penasaran, coba deh baca artikel dari web Mozilla langsung. Bakal lebih praktis kalau sambil praktik sekalian.

- **AND (&)**
- **OR (|)**
- **XOR (^)**
- **NOT (~)**
- **LEFT SHIFT (<<)**
- **RIGHT SHIFT (>>)**
- **ZERO-FILL RIGHT SHIFT (>>>)**

Modify-in-place~

Ketika ingin menyunting nilai dari suatu variabel (misalnya kita butuh nilai awal dari variabel tersebut), kita bisa memakai cara **modify in place** ini lho sob~

Agar lebih jelas, coba deh perhatikan contoh gambar di samping.

```
let n = 2;
console.log(n) // Output: 2

/* Kita pengen rubah nilai n,
   tapi kita butuh nilai n yang sebelumnya */
n = n + 5; // n = 2 + 5
console.log(n); // Output: 7

n = n ** 2; // n = 7 ** 2
console.log(n); // Output: 49
```

Ada contoh lagi nih!

Kita punya variabel "**n**" yang ingin kita tambahkan dan pangkatkan dengan beberapa angka.

Di baris kedua ada **2 + 5**, di mana hasil dari baris kedua adalah **7**. Itu bakal **disimpan kembali ke variabel "n"**.



```
/* Atau bisa ditulis lebih simpel kayak gini */
let n = 2;
n += 5; // n = 7 (n = n + 5)
n **= 2; // n = 49 (n = n ** 2)
```

Operator **"modify-and-assign"** berlaku juga buat semua operator aritmatika dan bitwise, kayak **/=**, **-=**, dan lain-lain.

Operator tersebut punya prioritas yang sama dengan prioritas normal.

Jadi, operator ini bakal dijalankan sebagian perhitungan lainnya.

```
let n = 2;
n *= 3 + 5;
console.log(n); // 16 (Bagian kanan akan dievaluasi terlebih dahulu)
```

Comma~

Operator ini merupakan salah satu operator yang paling jarang dipakai dan dihitung sebagai kasus yang nggak biasa. Malahan, kadang dipakai buat menulis kode yang lebih pendek.

Dengan operator ini, kita dapat mengevaluasi beberapa ekspresi dan membaginya dengan koma. Meskipun masing-masing dievaluasi, tapi cuma hasil terakhir yang bakal dikembalikan.



```
let a = (1 + 2, 3 + 4);
console.log(a);
// Output: 7, hasil dari 3 + 4
```

Biar lebih paham, latihan dulu yuk!

1. Buatlah variabel **kendaraan** dengan tipe data **array**
2. Tambahkan 5 data ke dalam variabel **kendaraan** dengan tipe data **object** yang berisikan property **tipe** dan **roda**
3. Cetak setiap data di dalam variabel kendaraan dengan format “[**tipe**] memiliki [**roda**] roda”



Setelah mempelajari Chapter 1 Topic 3 ini, kita jadi paham tentang...

Comments

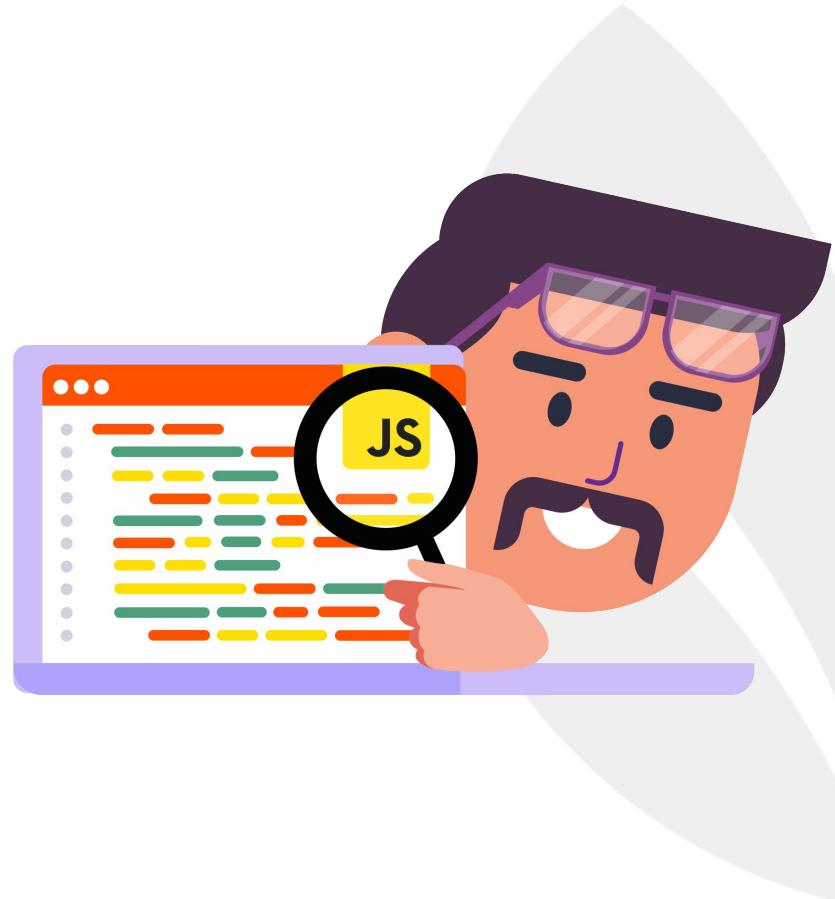
Penjelasan atau catatan dalam kode yang tidak dieksekusi, digunakan untuk dokumentasi.

Variable

Tempat penyimpanan data dalam program yang dapat diubah nilainya selama eksekusi.

Data Types

Jenis nilai yang dapat disimpan, seperti string, number, boolean, dan lain-lain.



Operator

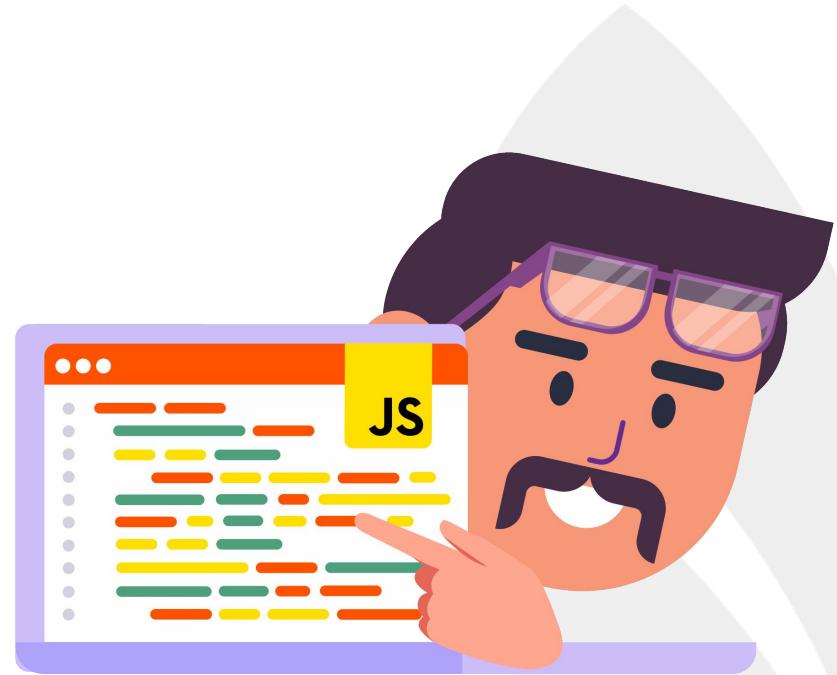
Simbol atau tanda yang digunakan untuk melakukan operasi pada nilai atau variabel.

Array

Struktur data untuk menyimpan kumpulan nilai dalam satu variabel.

Object

Struktur data untuk menyimpan properti dan nilai terkait.



Aritmatika

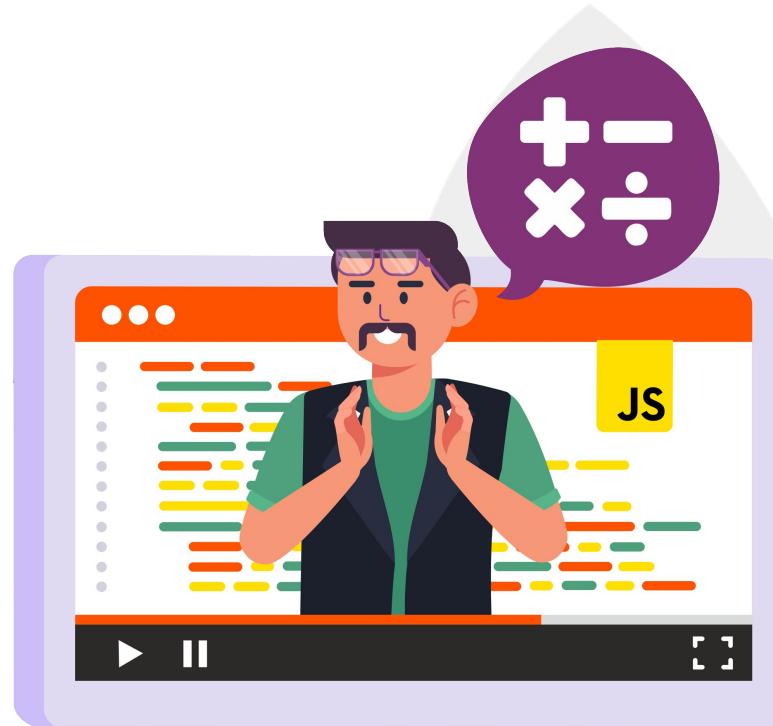
Operasi matematika seperti penambahan, pengurangan, perkalian, dan pembagian.

Comparison Operators

Membandingkan dua nilai atau variabel untuk menentukan hubungan di antara mereka.

Logical Operator

Operasi yang melibatkan nilai kebenaran (boolean) seperti AND, OR, NOT.

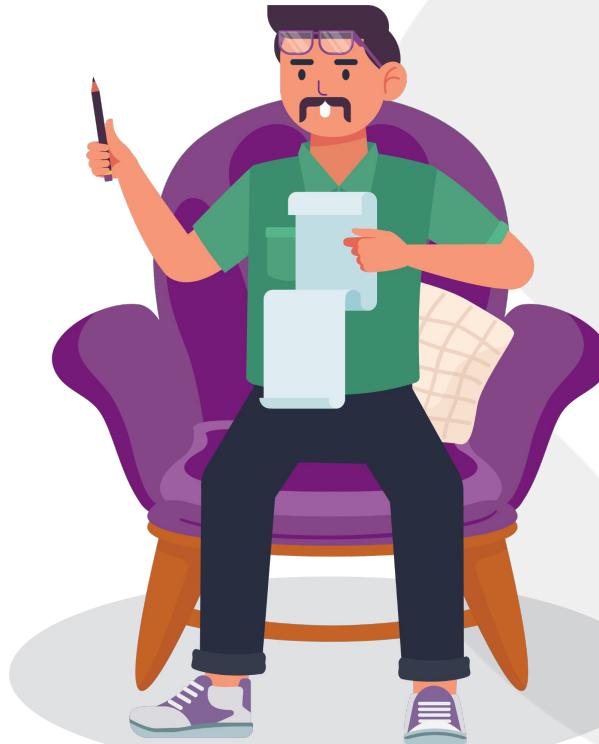


Assignment

Memberikan nilai pada variabel dengan operator seperti
=, +=, -=.

Ternary Operator

Operator kondisional singkat yang memiliki tiga bagian:
kondisi, nilai jika benar, nilai jika salah.



Referensi

1. <https://www.w3schools.com/js/default.asp>
2. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
3. <https://www.petanikode.com/tutorial/javascript/>



Mantap! Selamat ya karena kamu udah berhasil menamatkan Chapter 1 Topic 3



Next, kita bakal lanjut ke Topic 4 yang mana banyak materi baru dan seru.

See you later, learners!



**Selamat
Anda 😊
BERHASIL**