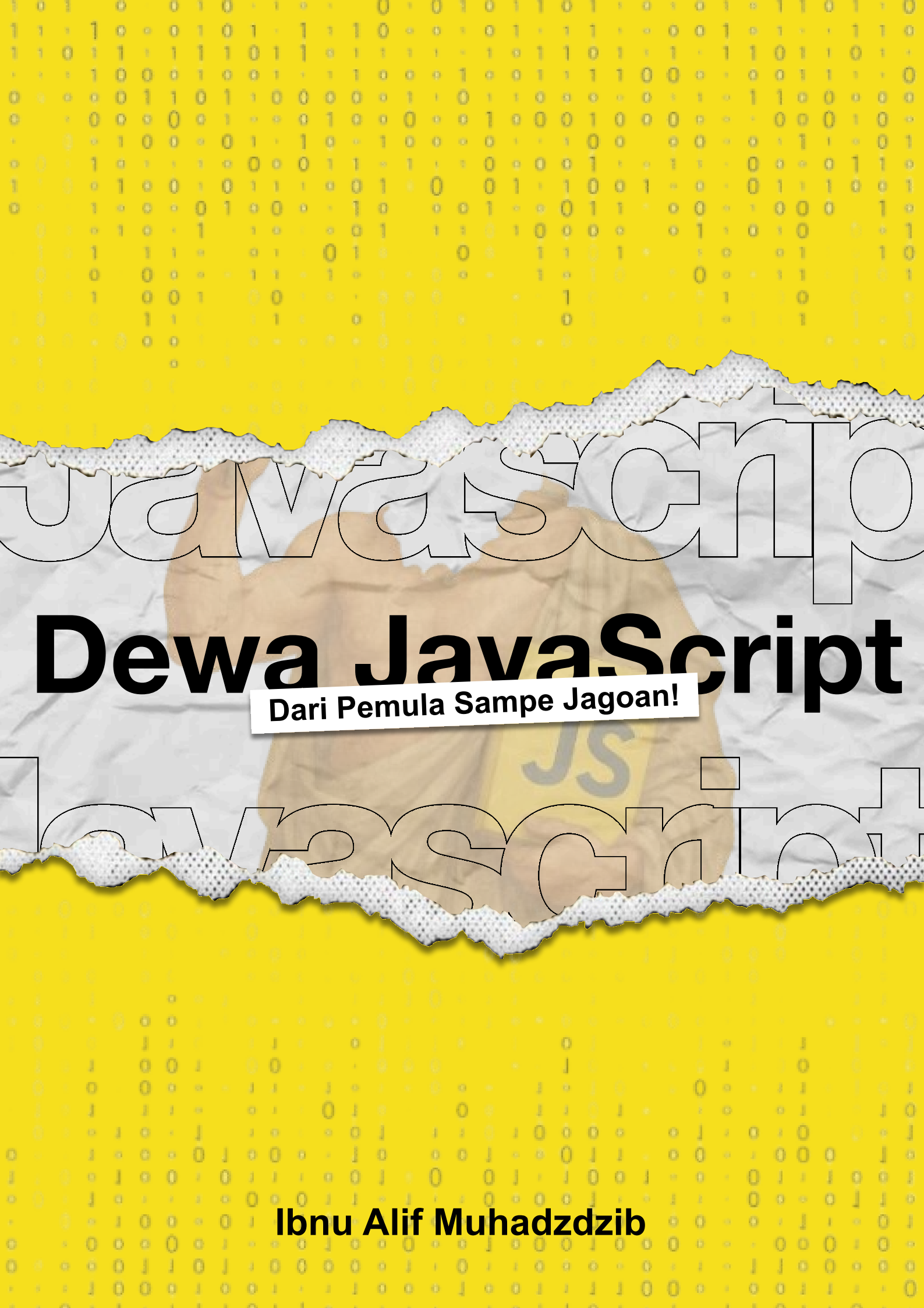

Ibnu Alif Muhadzdzib

KATA PENGANTAR

Bismillah…. Pertama-tama rasanya ada yang kurang jika kita belum bersyukur kepada rabb kita yang tidak ada illah yang berhak disembah selainnya. Yaitu Allah Subahanahu Wata'ala. Sholawat dan salam selalu tercurahkan kepada Rasulullah Shalallahu Alaihi Wassalam.

Selamat datang di buku Pemrograman JavaScript. Buku ini saya buat khusus untuk kamu yang masih pemula atau belum pernah ngoding sebelumnya, karena buku ini bakal ngajak kamu belajar dari dasar banget sampai ke topik yang lebih menantang.

Buku ini juga dilengkapi dengan banyak contoh kode yang gampang dipahami dan langsung bisa kamu coba sendiri. Tujuannya? Biar kamu nggak cuma baca teori, tapi juga langsung praktek. Karena belajar itu nggak cuma soal ngerti, tapi juga soal ngerasain sendiri.

Kami tahu belajar coding kadang bisa bikin mumet, tapi percayalah, proses ini worth it banget. Dengan dasar yang kamu dapat dari buku ini, kamu bakal lebih percaya diri untuk eksplorasi hal-hal baru dan bahkan mungkin bikin proyek keren kamu sendiri.

Semangat ngoding, Temen-temen!

Ibnu Alif Muhadzdzib
10 Desember 202

DAFTAR ISI

Javascript merupakan salah satu Bahasa pemrograman untuk komputer. Kamu tau gak apa itu pemrograman? Nah sebelum kita masuk ke pembelajaran dasar javascript, kita harus mengetahui dulu arti dari istilah tersebut.

Apa Itu Pemrograman?

Pemrograman adalah proses menulis, menguji, memperbaiki, dan memelihara kode instruksi untuk mengendalikan komputer. Initnya dari pengertian tersebut kita bisa menyimpulkan bahwa pemrograman adalah proses dimana kita mengendalikan komputer dengan cara menulis kode, mengujinya, dan memperbaikinya serta memeliharanya. Melalui pemrograman, ada sangat banyak permasalahan yang dapat diatasi sehingga dapat memudahkan kehidupan manusia, contohnya di bawah ini.

Melakukan pemesanan/pembelian tiket liburan, nonton film, makanan & minuman, dan sebagainya melalui aplikasi perangkat mobile.

Membayar tagihan belanjaan di e-commerce dengan mengisi formulir informasi kartu kredit/debit dan klik tombol “Bayar” melalui browser.

Melakukan pertemuan dengan banyak orang secara daring.

Nah sekarang kita sudah tau apa sih pengertian dari pemrograman itu sendiri. Nah sekarang mari kita mulai masuk ke dalam materi JavaScript

Pengenalan JavaScript

Sejarah JavaScript

Javascript atau kerap kali disebut JS adalah bahasa pemrograman tingkat tinggi dan juga dinamis yang ditemukan olehh ***Brendan Eich*** pada tahun ***1995*** saat ia bekerja di perusahaan bernama Netscape Communication Corporation.

Pada awalnya JS bernama Mocha kemudian Brendan Eich mengganti namanya menjadi LiveScript pada saat menyerahkannya kepada Netscape Corporation dan pada akhirnya berubah kembali menjadi JavaScript hingga sekarang.

Pengertian JavaScript

JS adalah sebuah bahasa pemrograman untuk membuat sebuah aplikasi dan halaman website agar lebih dinamis dan memiliki fungsi yang lebih banyak dan variatif, JS juga biasanya seringkali dipakai berdampingan dengan bahasa pemrograman HTML *(Hypertext Markup Language)* dan juga CSS *(Cascading Stylesheet)*. Bahasa ini bisa di jalankan di banyak aplikasi browser seperti contoh; Chrome, Firefox, Opera, Edge, dll.

Materi Dasar JavaScript

Variabel

Sebelum kita mempelajari bagaimana cara membuat variabel, kita harus mengetahui dulu apa sih variabel itu?.

Variabel adalah suatu struktur data yang memiliki nama dan di dalamnya terdapat suatu nilai. Sebagai contoh kasus dalam dunia nyata, anda sedang berbelanja di sebuah supermarket, kemudian di supermarket tersebut anda memiliki sebuah keranjang dan anda mengisi keranjang belanja anda dengan barang barang di supermarket. Barang yang ada dalam keranjang ini bisa berubah seiring waktu anda bisa menambah barang, mengurangi barang, atau mengganti barang dengan yang lain. Nah keranjang belanja ini adalah sebuah analogi dari variabel.

Keranjang belanja adalah sebuah contoh dari variabel. Sedangkan barang yang ada di dalam keranjang belanja adalah sebuah isi dari variabel anda. Dan terkadang barang yang ada di dalam variabel anda bisa berubah, misalnya anda menambah barang atau menghapus barang, seperti halnya variabel yang nilainya bisa berubah selama program berjalan.

Dalam dunia nyata, variabel bisa dipahami sebagai tempat untuk menyimpan sesuatu yang dapat berubah. Seperti halnya sebuah keranjang belanja yang dapat diisi dengan barang yang berbeda-beda, variabel dalam pemrograman menyimpan data yang nilainya bisa diubah sesuai dengan kebutuhan aplikasi atau program yang sedang berjalan.

A. Cara membuat variabel

Sebelum kita membuat variabel di Js kita harus mengetahui dulu kata kunci untuk membuat variabel di Js. Sejauh ini ada 3 kata kunci untuk membuat variabel di Js, yaitu:

- var

Digunakan sebelum ES6 (ECMAScript 2015). Sekarang penggunaannya cenderung diminimalkan karena memiliki keterbatasan seperti:

1. *Scope global atau function:* Variabel yang dideklarasikan dengan var dapat diakses di seluruh fungsi atau bahkan di luar blok (misalnya dalam loop atau conditional statements).
2. *Hoisting:* Variabel yang menggunakan var akan diangkat ke atas sebelum eksekusi kode, tetapi nilainya tetap undefined hingga diinisialisasi.

Contoh penggunaan:

```
//////////////////////////////////////////////
  var nama = 'Ibnu';
  console.log(nama) // Output: Ibnu
//////////////////////////////////////////////
```

- let
  Diperkenalkan di ES6 dan lebih sering digunakan dibandingkan var. Memiliki keunggulan:

1. *Scope blok:* Hanya dapat diakses dalam blok tempat variabel itu dideklarasikan, seperti dalam {…}.
2. *Tidak dapat dideklarasikan ulang:* Dalam scope yang sama, Anda tidak dapat mendeklarasikan variabel let dengan nama yang sama.

Contoh penggunaan:

```
//////////////////////////////////////////////
  let jurusan = 'RPL';
  console.log(jurusan) // Output: RPL
//////////////////////////////////////////////
```

Dan yang ketiga adalah

- const
  Digunakan untuk mendeklarasikan variabel yang nilainya tidak akan berubah (konstanta).

1. *Immutable (nilai tetap)*: Nilai dari const tidak dapat diubah setelah dideklarasikan.
2. *Scope blok:* Sama seperti let.

```
//////////////////////////////////////////////
  const namaLengkap = 'Jajang Julpikar';
  console.log(namaLengkap) // Output: Jajang Julpikar
//////////////////////////////////////////////
```

Note:

Namun terdapat 2 pengecualian dalam penggunaan kata kunci const. Variabel yang memiliki tipe data array atau objek **bisa** diubah nilai element atau propertinya meskipun menggunakan kata kunci const saat dideklarasikan. Kita akan belajar ini di topik yang akan datang.

B. Aturan penamaan variabel

Setelah tau cara membuat variabel sekarang kita akan belajar tentang aturan dalam penulisan variabel. Karena variabel tidak bisa dideklarasikan secara asal asalan teman teman. Nah aturannya adalah sebagai berikut:

- Nama variabel hanya boleh berisi huruf (A-Z, a-z), angka (0-9), underscore (_), atau simbol dolar ($).
- Nama variabel tidak boleh dimulai dengan angka.
- Tidak boleh menggunakan **kata kunci JavaScript** (seperti if, while, true, dll.) sebagai nama variabel.
- Gunakan **penamaan yang deskriptif** untuk mempermudah pemahaman kode.

Contoh nama variabel yang valid

```
////////////////////////////////////////////////
  let namaDepan = ‘Ibnu’;
  let _umur = 15;
  let $saldo = 1000;
////////////////////////////////////////////////
```

Contoh variabel yang tidak valid:

```
////////////////////////////////////////////////
  let 1angka = 5; // Tidak valid karena diawali angka
  let if = true; // Tidak valid karena "if" adalah kata kunci
////////////////////////////////////////////////
```

C. Inisialisasi variabel

Kamu sekarang sudah mengerti peraturan dalam mendeklarasikan sebuah variabel. Sekarang kita akan belajar bagaimana cara menginisialisasi / mengisi sebuah variabel.

Kita bisa menginisialisasi variabel tanpa memberikan nilai awal. Sebagai contoh:

```
//////////////////////////////////////////////////
  let nama; // Deklarasi variabel
nama = "Ibnu"; // Inisialisasi nilai
console.log(nama); // Output: Ibnu
//////////////////////////////////////////////////
```

Tetapi untuk const kita harus menginisialisasi langsung dan tidak bisa dideklarasikan tanpa memberikan nilai awal. Jadi jika kita ingin mendeklarasikan const kita harus langsung mengisi nilainya dari awal

Contoh dengan nilai awal:

```
//////////////////////////////////////////////////
const pi = 3.14; // Harus langsung diinisialisasi
//////////////////////////////////////////////////
```

Tipe Data

Nah kalo kamu sudah bisa mendeklarasikan sebuah variabel maka kamu juga harus bisa menginisialisasi nilai dari variabel dengan bermacam macam tipe data. Sebelumnya apasih tipe data itu?.

Tipe data adalah konsep fundamental dalam JavaScript yang mengacu pada jenis nilai yang dapat disimpan dan dimanipulasi dalam sebuah variabel. Pemahaman tipe data sangat penting karena memengaruhi cara Anda memanipulasi data dan membuat program bekerja dengan benar.

JavaScript memiliki dua kategori tipe data utama:

1. Tipe Data Primitif
2. Tipe Data Referensi

A. Tipe data primitif

Tipe data primitif adalah tipe data dasar di JavaScript yang tidak memiliki metode atau properti (kecuali melalui *wrapper object*). Tipe data ini bersifat **immutable**, artinya nilainya tidak dapat diubah setelah didefinisikan. Sebagai contoh, jika Anda mengubah sebuah string, JavaScript sebenarnya membuat string baru dan tidak mengubah string lama.

Nilai tipe data primitif **langsung disimpan di memori stack**, sehingga lebih cepat diakses dibandingkan tipe data referensi.

JavaScript memiliki **7 tipe data primitif**. Berikut adalah penjelasan rinci masing-masing:

- String

String adalah tipe data untuk menyimpan teks (rangkaian karakter). String diapit dengan tanda kutip tunggal ('), kutip ganda ("), atau backticks (`).

Karakteristik String:

1. String dapat kosong ("") atau memiliki panjang tertentu.
2. Immutable: Jika Anda memanipulasi string, JavaScript sebenarnya membuat string baru dan tidak memodifikasi string asli.

Contoh string:

```
//////////////////////////////////////////////
  let teks1 = "Halo Dunia"; // Kutip ganda
  let teks2 = 'Selamat Pagi'; // Kutip tunggal
  let nama = "Ibnu";
  let sapaan = `Hai, ${nama}!`; // Template literal (menggunakan backticks)
console.log(teks1); // Output: Halo Dunia
console.log(teks2); // Output: Selamat Pagi
console.log(sapaan); // Output: Hai, Ibnu!
//////////////////////////////////////////////
```

String juga memiliki metode dan properti:

Properti length

```
//////////////////////////////////////////////
let teks1 = "Halo Dunia";
console.log(teks1.length); // Output: 10
//////////////////////////////////////////////
```

Metode seperti .toUpperCase(), .toLowerCase(), .substring(), dll

```
//////////////////////////////////////////
let teks1 = "Halo Dunia";
console.log(teks1.toUpperCase());
// Output: HALO DUNIA
//////////////////////////////////////////
```

- Number

Number digunakan untuk menyimpan angka, baik bilangan bulat maupun bilangan desimal.

Karakteristik Number:

1. Mendukung operasi matematika seperti penjumlahan, pengurangan, perkalian, da
2. Dapat menyimpan nilai khusus:
    - Infinity: Hasil dari pembagian angka dengan nol.
    - -Infinity: Nilai negatif tak hingga.
    - NaN (Not-a-Number): Hasil dari operasi matematika yang tidak valid.

Contoh dari tipe data number:

```
//////////////////////////////////////////
let bilanganBulat = 42;
let bilanganDesimal = 3.14;
let hasilPembagian = 10 / 0; // Infinity
let operasiTidakValid = "halo" / 2; // NaN
console.log(bilanganBulat); // Output: 42
console.log(bilanganDesimal); // Output: 3.14
```

```
console.log(hasilPembagian); // Output: Infinity
console.log(operasiTidakValid); // Output: NaN
//////////////////////////////////////////
```

- BigInt

BigInt adalah tipe data baru yang diperkenalkan di ES2020. Tipe ini digunakan untuk menyimpan angka yang sangat besar melebihi batas Number, yaitu ±(2^53 - 1).

Karakteristik BigInt:

1. Ditandai dengan huruf n di akhir angka.
2. Tidak dapat dicampur langsung dengan tipe Number dalam operasi matematika.

Contoh dari BigInt:

```
//////////////////////////////////////////
  let angkaBesar=1234567890123456789012n;
  let angkaKecil = BigInt(12345);
  console.log(angkaBesar); // Output:
1234567890123456789012n
console.log(angkaKecil); // Output: 12345n
//////////////////////////////////////////
```

- Boolean

Bolean adalah tipe data logika dengan hanya dua nilai:

1. true: Untuk menyatakan benar.
2. false: Untuk menyatakan salah.

Karakteristik Boolean: Sangat berguna dalam pengambilan keputusan (misalnya dalam kondisi if-else).

Contoh dari tipe data boolean:

```
//////////////////////////////////////////
  let isOnline = true;
  let isError = false;
console.log(isOnline); // Output: true
console.log(isError); // Output: false
let angka = 10;
console.log(angka > 5); // Output: true
console.log(angka < 5); // Output: false
 //////////////////////////////////////////
```

- Undefined

Undefined adalah nilai default dari variabel yang telah dideklarasikan tetapi belum diberi nilai.

Karakteristik Undefined: Mengindikasikan bahwa variabel belum diinisialisasi.

Contoh dari undefined:

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

- Null

Null adalah tipe data khusus yang menunjukkan bahwa variabel tidak memiliki nilai atau kosong secara sengaja.

Karakteristik Null:

- Berbeda dengan undefined, yang berarti variabel belum diinisialisasi.
- Digunakan untuk "mengosongkan" variabel.

Contoh dari null:

```
//////////////////////////////////////////////
let kosong = null;
console.log(kosong);
//////////////////////////////////////////////
```

- Symbol

Symbol adalah tipe data yang digunakan untuk membuat identifier unik. Karakteristik Symbol:

- Tidak pernah sama meskipun memiliki deskripsi yang sama.
- Biasanya digunakan untuk membuat properti objek yang unik.

Contoh dari symbol:

```
//////////////////////////////////////////////
let id1 = Symbol("id");
let id2 = Symbol("id");
console.log(id1 === id2); // Output: false
//////////////////////////////////////////////
```

B. Tipe data referensi

Tipe data referensi di JavaScript digunakan untuk menyimpan data yang kompleks atau berbentuk koleksi. Berbeda dengan tipe data primitif yang disimpan di memori **stack**, tipe data referensi disimpan di memori **heap**, dan variabel hanya menyimpan **referensi** (alamat) ke lokasi memori tersebut.

Tipe data referensi memungkinkan kita untuk bekerja dengan objek, array, fungsi, dan struktur data yang lebih kompleks.

Karakteristik Tipe Data Referensi:

1. *Disimpan di Heap:* Data sebenarnya berada di memori heap, sedangkan variabel hanya menyimpan referensinya.
2. *Mutable:* Nilainya dapat diubah setelah didefinisikan.
3. *Perbandingan Berdasarkan Referensi:* Dua objek dianggap berbeda meskipun memiliki nilai yang sama, karena referensinya berbeda.

Jenis jenis tipe data referensi:

- Object

Object / Objek dalam bahasa Indonesia adalah tipe data referensi utama di JavaScript. Ini adalah koleksi pasangan kunci-nilai. Objek dapat menyimpan berbagai tipe data, termasuk tipe primitif dan tipe referensi lainnya.

Contoh dari tipe data objek :

```
//////////////////////////////////////////
const person = {
name: "Dodi",
age: 30,
isEmployed: true
};
//////////////////////////////////////////
```

Dalam contoh ini:

- *person* adalah sebuah objek.
- *name, age*, dan *isEmployed* adalah properti objek.

- Array

Array adalah struktur data yang digunakan untuk menyimpan kumpulan nilai dalam satu variabel. Nilai-nilai ini bisa berupa tipe data primitif (seperti number, string, boolean) atau tipe data referensi lainnya (seperti objek atau fungsi).

Contoh dari array:

```
//////////////////////////////////////////
const numbers = [1, 2, 3, 4, 5];
console.log(numbers); // Output:[1,2,3,4,5]
//////////////////////////////////////////
```

Dalam contoh array tadi:

- *fruits* adalah array yang menyimpan tiga elemen bertipe string.
- Elemen array diakses menggunakan indeks, ***dimulai dari 0.***

- Function

Fungsi (function) di JavaScript adalah blok kode yang dirancang untuk melakukan tugas tertentu atau menghitung suatu nilai. Fungsi digunakan untuk mengorganisasi, mengulang, dan menyederhanakan kode dengan cara mengelompokkan logika yang dapat digunakan kembali.

Contoh dari function:

```
//////////////////////////////////////////
function greet(name) {
 return `Hello, ${name}!`;
}
//////////////////////////////////////////
```

Dalam contoh ini *greet* adalah fungsi yang menerima parameter name.

- Date

Tipe data Date di JavaScript adalah objek bawaan yang digunakan untuk menangani tanggal dan waktu. Dengan Date, Anda dapat membuat objek yang merepresentasikan tanggal dan waktu tertentu, mengambil informasi tentang waktu (seperti tahun, bulan, atau hari), serta melakukan operasi seperti menambah atau mengurangi waktu.

Contoh dari tipe data *Date* (Tanpa argumen):

```
//////////////////////////////////////////////
const now = new Date();
console.log(now); // Output: Contoh: Sat Dec 07 2024 15:23:11 GMT+0700 (WIB)
//////////////////////////////////////////////
```

Kita bisa membuat objek tidak hanya dengan tanpa argumen seperti contoh diatas, tetapi kita juga bisa membuatnya dengan *Timestamp*

```
//////////////////////////////////////////////
const dateFromTimestamp = new Date(0);
// 1 Januari 1970
console.log(dateFromTimestamp);
// Output: Thu Jan 01 1970 07:00:00 GMT+0700 (WIB)
//////////////////////////////////////////////
```

Dengan string:

```
//////////////////////////////////////////
const specificDate = new Date("2024-12-07T12:00:00Z");
console.log(specificDate);
// Output: Sat Dec 07 2024 19:00:00 GMT+0700 (WIB)
//////////////////////////////////////////
```

Dan juga dengan Nilai Tahun, Bulan, dan Lainnya:

```
//////////////////////////////////////////
const customDate = new Date(2024, 11, 7, 15, 30);
// 7 Desember 2024, 15:30
console.log(customDate);
// Output: Sat Dec 07 2024 15:30:00 GMT+0700 (WIB)
//////////////////////////////////////////
```

- Null

Tipe data **null** di JavaScript adalah salah satu tipe data primitif yang digunakan untuk merepresentasikan nilai "tidak ada" atau "kosong." null secara eksplisit menunjukkan bahwa sebuah variabel tidak memiliki nilai yang valid.

Nah setelah kalian membaca pengertian diatas pasti kalian agak sedikit bingung, apasih perbedaan tipe data ***Null*** dan juga tipe data ***Undefined***.

Perbedaan Antara null dan undefined

Aspek	null	undefined
Pengertian	Menunjukkan bahwa nilai variabel kosong atau tidak ada secara eksplisit.	Menunjukkan bahwa variabel belum diberi nilai.
Penetapan	Ditentukan secara eksplisit oleh programmer.	Biasanya diberikan oleh JavaScript secara default.
Penggunaan	Digunakan untuk mengosongkan objek atau variabel.	Digunakan sebagai default untuk variabel yang tidak diinisialisasi atau properti yang tidak ada.
Pemeriksaan	typeof null menghasilkan "object".	typeof undefined menghasilkan "undefined".

Contoh:

```
//////////////////////////////////////////
let a = null; // Secara eksplisit diatur ke null
let b; // Tidak diinisialisasi, nilainya undefined
console.log(a); // Output: null
console.log(b); // Output: undefined
//////////////////////////////////////////
```

Operator JavaScript

Kamu sudah melewati materi tentang tipe data di Js. Pelajaran sebelumnya bisa kamu hafalkan dan dipraktikan agar selalu ingat.

A. Operator Aritmatika

Nah sekarang kita akan belajar tentang *operator aritmatika* di Js. Operator-operator tersebut bisa kamu lihat di tabel yang ada dibawah:

Operator	Deskripsi
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
**	Eksponen (pangkat)
%	Modulus (menghasilkan sisa hasil pembagian)
++	Increment (menambah 1)
--	Decrement (mengurangi 1)

Operator yang digunakan dalam Js sama dengan operator yang kita gunakan di dunia nyata untuk mengerjakan kasus aritmatika

Sekarang kau bisa melihat contoh penggunaan operator aritmatika dalam js:

Penjumlahan

```
//////////////////////////////////////////////
let bilangan1 = 10;
let bilangan2 = 3;
console.log(bilangan1 + bilangan2); // Output: 13
//////////////////////////////////////////////
```

Pengurangan

```
//////////////////////////////////////////////
let bilangan1 = 10;
let bilangan2 = 3;

console.log(bilangan1 - bilangan2); // Output: 7
//////////////////////////////////////////////
```

Perkalian

```
//////////////////////////////////////////////
let bilangan1 = 10;
let bilangan2 = 3;
console.log(bilangan1 * bilangan2); // Output: 30
//////////////////////////////////////////////
```

Eksponen (pangkat)

```
//////////////////////////////////////////////
let bilangan1 = 10;
let bilangan2 = 3;

console.log(bilangan1 ** bilangan2); // Output: 1000
//////////////////////////////////////////////
```

Modulus

```
//////////////////////////////////////////////
let bilangan1 = 10;
let bilangan2 = 3;

console.log(bilangan1 % bilangan2); // Output: 1
//////////////////////////////////////////////
```

Increment (tambah 1)

```
//////////////////////////////////////////////
let bilangan = 10;
bilangan++;

console.log(bilangan); // Output: 11
//////////////////////////////////////////////
```

Decrement (kurang 1)

```
//////////////////////////////////////////////
let bilangan = 10;
bilangan--;

console.log(bilangan); // output: 9
//////////////////////////////////////////////
```

B. Assignment Operator

Assignment operator digunakan untuk memberikan atau menetapkan nilai ke variabel dalam JavaScript. Operator ini memungkinkan programmer untuk mengatur, memperbarui, atau mengkombinasikan nilai dengan cara yang efisien.

Semua operatornya bisa kamu lihat pada tabel dibawah ini:

Assignment	Operator	Contoh Penggunaan	Setara Dengan
=		x=y	x=y
=	+	x += y	x = x + y
=	-	x -= y	x = x - y
=	*	x *= y	x = x * y
=	/	x /= y	x = x / y
=	%	x %= y	x = x % y
=	**	x **= y	x = x ** y

Contoh penggunaannya di Js:

Assignment =

```
//////////////////////////////////////////////
let nama = "Deni";
let umur = 20;
let apakahSudahMenikah = false;
let jumlahAnak;

console.log(nama); // Output: Deni
console.log(umur); // Output: 20
console.log(apakahSudahMenikah); // Output: false
console.log(jumlahAnak); // Output: undefined
//////////////////////////////////////////////
```

Adittion Assignment +=

```
//////////////////////////////////////////////
let bilangan = 10;
bilangan += 5;

console.log(bilangan); // Output: 15
//////////////////////////////////////////////
```

Subtraction Assignment -=

```
//////////////////////////////////////////////
let bilangan = 10;
bilangan -= 5;

console.log(bilangan); // Output: 5
//////////////////////////////////////////////
```

Multiplication Assignment *=

```
//////////////////////////////////////////////
let bilangan = 10;
bilangan *= 5;

console.log(bilangan); // Output: 50
//////////////////////////////////////////////
```

Division Assignment /=

```
//////////////////////////////////////////////
let bilangan = 10;
bilangan /= 5;

console.log(bilangan); // Output: 2
//////////////////////////////////////////////
```

Modulus Assignment %=

```
//////////////////////////////////////////
let bilangan = 10;
bilangan %= 5;

console.log(bilangan); // Output: 0
//////////////////////////////////////////
```

Exponent Assignment **=

```
//////////////////////////////////////////
let bilangan = 10;
bilangan **= 5;

console.log(bilangan); // Output: 100000
//////////////////////////////////////////
```

C. String Operator

Nah setelah kamu memahami **Assignment Operator** dan sudah mempraktikannya sekarang kita akan lanjut mempelajari **String Operator**.

String Operator adalah operator yang digunakan untuk melakukan operasi pada *string* (teks) dalam bahasa pemrograman. String adalah urutan karakter yang digunakan untuk menyimpan teks, seperti kata, kalimat, atau bahkan angka yang ditulis sebagai teks. Operator string memungkinkan manipulasi atau pengolahan string,

seperti penggabungan (*concatenation*), pengulangan, atau membandingkan nilai string.

String operator biasanya digunakan untuk menggabungkan dua atau lebih data string. Operasi ini biasa dikenal dengan sebutan **string concatenation**.

Ada 2 macam String Operator:

- +
- +=

Contoh penggunaan:

Operator +

```
////////////////////////////////////////////
let namaDepan = "Gugum";
let namaBelakang = "Gumelar";

console.log(namaDepan + " " + namaBelakang); // Output: Gugum Gumelar
////////////////////////////////////////////
```

Operator +=

```
////////////////////////////////////////////
let kata = "Halo ";
kata += "Dunia";

console.log(kata); // Output: Halo Dunia
////////////////////////////////////////////
```

Note:

Oh iya, di JavaScript, jika kita menggabungkan data string dengan data number, maka hasil akhirnya juga akan menjadi string. Contoh:

```
//////////////////////////////////////////////
let kata = "Kambing ada ";
let bilangan = 5;

console.log(kata + bilangan); // Output: Kambing ada 5
//////////////////////////////////////////////
```

D. Operator Perbandingan

Nah selanjutnya kita akan belajar operator yang pasti akan banyak di pakai di sytax kode JavaScript, karena mereka adalah inti dari pengambilan keputusan dan logika program. Dalam pemrograman, pengambilan keputusan sangat penting untuk mengontrol alur program, seperti menjalankan perintah tertentu berdasarkan kondisi tertentu. Yaitu **Operator Perbandingan**

Operator perbandingan (*comparison operators*) di JavaScript adalah jenis operator yang digunakan untuk membandingkan dua nilai. Operator ini mengembalikan hasil berupa nilai **boolean**:

- **true** jika kondisi perbandingan terpenuhi.
- **false** jika tidak terpenuhi.

Operator perbandingan sangat penting dalam pengambilan keputusan, seperti yang dilakukan dalam struktur kontrol (if, while, dll. [kita akan belajar nanti]).

Beberapa operatornya bisa dilihat pada tabel dibawah ini:

Operator	Deskripsi
==	Sama dengan (cek nilai)
===	Sama dengan (cek nilai dan tipe data)
!=	Tidak sama dengan (cek nilai)
!==	Tidak sama dengan (cek nilai dan tipe data)
>	Lebih dari
<	Kurang dari
>=	Lebih dari atau sama dengan
<=	Kurang dari atau sama dengan
? :	Ternary operator

Kita lihat contoh penggunaannya dalam Js:

Operator == (Loose Equality)

```
//////////////////////////////////////////////
let bilangan = 10;

console.log(bilangan == 10); // Output: true
console.log(bilangan == 8); // Output: false
console.log(bilangan == "10"); // Output: true
//////////////////////////////////////////////
```

Mari kita coba perhatikan bilangan == "10". Kenapa outputnya bisa true? Karena di dalam JavaScript jika operand-nya berbeda tipe data, maka salah satunya akan dipaksa berubah tipe datanya (**type coercion**) agar mereka sama dan bisa dibandingkan

Note:
Yang dimaksud dengan **operand** adalah nilai yang digunakan di dalam sebuah proses operasi. Misalnya dalam operasi penjumlahan 1 + 2, yang tergolong operand adalah angka 1 dan 2.

Agar lebih paham, mari kita coba perhatikan contoh di bawah ini:

```
//////////////////////////////////////////////
console.log("3.14" == 3.14) // Output: true
//////////////////////////////////////////////
```

Kenapa kode di atas menghasilkan output true?

Sebab "3.14" diubah dulu menjadi tipe data number agar bisa dibandingkan nilainya. Lalu baru dibandingkan 3.14 === 3.14.

Jadi pada dasarnya == akan berusaha mengubah tipe data jika tipe dari kedua berbeda. Lalu bagaimana kalau kita ingin membandingkan nilai dan tipe datanya juga? Mari kita lihat contoh selanjutnya.

Operator === (Strict Equality)

```
//////////////////////////////////////////////
let bilangan = 10;
console.log(bilangan === 10); // Output: true --> nilai dan tipe data sama
console.log(bilangan === "10"); // Output: false --> nilai sama tetapi tipe data tidak sama
//////////////////////////////////////////////
```

Kali ini, outputnya akan berupa true apabila ***kedua operand memiliki nilai dan tipe data yang sama***. Makanya pada baris kedua contoh kode di atas menghasilkan output false sebab kita membandingkan bilangan 10 dengan string "10" yang berbeda tipe datanya.

Operator !=

```
//////////////////////////////////////////
let bilangan = 10;

console.log(bilangan != 8); // Output: true --> nilai tidak sama tetapi tipe data sama
console.log(bilangan != "8"); // Output: true --> nilai dan tipe data tidak sama
console.log(bilangan != 10); // Output: false --> nilai dan tipe data sama
console.log(bilangan != "10"); // Output: false --> nilai sama tetapi tipe data tidak sama
//////////////////////////////////////////
```

Mungkin kalian agak bingung kalau melihat contoh tadi. Yang perlu kalian ketahui adalah operator != akan menghasilkan output false apabila **kedua operand memiliki nilai yang sama, tidak terpengaruh sama tipe datanya**.

Operator !==

```
//////////////////////////////////////////////
let bilangan = 10;
console.log(bilangan !== 8); // Output: true --> nilai tidak sama tetapi tipe data sama
console.log(bilangan !== "8"); // Output: true --> nilai dan tipe data tidak sama
console.log(bilangan !== 10); // Output: false --> nilai dan tipe data sama
console.log(bilangan !== "10"); // Output: true --> nilai sama tetapi tipe data tidak sama
//////////////////////////////////////////////
```

Operator >(Lebih Dari)

```
//////////////////////////////////////////////
let bilangan = 10;
console.log(bilangan > 10); // Output: false
console.log(bilangan > 11); // Output: false
console.log(bilangan > 8); // Output: true
//////////////////////////////////////////////
```

Operator < (Kurang Dari)

```
//////////////////////////////////////////////
let bilangan = 10;
console.log(bilangan < 10); // Output: false
console.log(bilangan < 8); // Output: false
console.log(bilangan < 11); // Output: true
//////////////////////////////////////////////
```

Operator >=(Lebih Dari atau Sama Dengan)

```
//////////////////////////////////////////////
let bilangan = 10;

console.log(bilangan >= 10); // Output: true
console.log(bilangan >= 8); // Output: true
console.log(bilangan >= 11); // Output: false
//////////////////////////////////////////////
```

Operator <= (Kurang Dari atau Sama Dengan)

```
//////////////////////////////////////////////
let bilangan = 10;

console.log(bilangan <= 10); // Output: true
console.log(bilangan <= 11); // Output: true
console.log(bilangan <= 8); // Output: false
//////////////////////////////////////////////
```

Operator Ternary (? :)

Digunakan untuk memberikan nilai pada variabel sesuai dengan kondisi yang ditentukan.

Syntaxnya adalah:

```
//////////////////////////////////////////////////////////////
variabel = (kondisi true) ? nilai1 : nilai2;
// Artinya apabila kondisi true, maka variabel akan diberi nilai1
// Apabila kondisi false, maka variabel akan diberi nilai2
//////////////////////////////////////////////////////////////
```

Contoh:

```
//////////////////////////////////////////////
let makanan = "daging";
let jenisHewan = makanan === "daging"  ? "karnivora" :
"herbivora";

console.log(jenisHewan); // Output: "karnivora"
//////////////////////////////////////////////
```

Pada kode di atas, kondisi yang diberikan adalah makanan === "daging". Karena kondisi tersebut bernilai true, maka yang dikembalikan oleh operator ini adalah "karnivora".

Seandainya variabel makanan tidak bernilai "daging", maka kondisi makanan === "daging" akan bernilai false dan nilai yang akan dikembalikan adalah "herbivora".

Function JavaScript

Pada pembelajaran sebelumnya pasti kamu sudah tahu bahwa **Function** atau Fungsi termasuk ke dalam tipe data. Lebih tepatnya lagi tipe data referensi, nah sekarang kita akan belajar tentang fungsi.

Jadi apasih **Function** dalam Js itu?, Di konsep JavaScript, fungsi bisa diilustrasikan sebagai berikut:

Bayangkan kamu sedang membuat burger di dapur. Kamu punya resep andalan (function) yang bisa dipakai kapan saja untuk membuat burger favoritmu. Resep ini sudah tersimpan rapi di kepala, jadi kamu tinggal menggunakannya setiap kali ingin memasak tanpa perlu menuliskannya lagi dari awal. Begini penjelasannya:

- **Resep Burger**
  Resep ini seperti sebuah function. Misalnya, kita beri nama buatBurger.
  Resep ini membutuhkan beberapa bahan (parameter), seperti roti, daging, selada, dan saus.
- **Eksekusi Resep**
  Ketika kamu ingin makan burger, cukup panggil resepnya, masukkan bahan-bahannya, dan burgermu siap disajikan.

Analogi halusnya Function itu seperti memiliki seorang koki pribadi yang selalu siap membantu. Kamu cukup memberi tahu apa yang ingin dibuat dan bahan yang dibutuhkan, maka hasilnya langsung jadi. Praktis dan efisien, bukan?

Kamu bisa lihat ilustrasinya dalam sintaks Js:

```
////////////////////////////////////////////
function buatBurger(roti, daging, selada, saus) {
 return `Burger dengan ${roti}, ${daging}, ${selada}, dan ${saus}`;
}

// Panggil resepnya
let burgerFavorit = buatBurger
("roti gandum", "patty sapi", "selada segar", "saus BBQ");
```

```
console.log(burgerFavorit);
// Output: Burger dengan roti gandum, patty sapi, selada segar, dan saus BBQ
//////////////////////////////////////////////
```

Secara sederhana, **fungsi (function) dalam JavaScript** adalah blok kode yang dirancang untuk menjalankan tugas tertentu. Fungsi memungkinkan kita menulis kode sekali dan menggunakannya berkali-kali, sehingga mempermudah pengelolaan dan efisiensi kode.

Ciri utama fungsi dalam JavaScript:

1. **Nama Fungsi:** Fungsi biasanya diberi nama agar mudah digunakan kembali.
2. **Parameter:** Input yang diberikan ke fungsi untuk diproses.
3. **Return Value:** Hasil atau keluaran yang dikembalikan oleh fungsi (opsional).
4. **Reusable:** Bisa dipanggil kapan saja tanpa perlu menulis ulang logika.

Note:

Dalam JavaScript, kita bisa menggunakan **//** untuk memberikan komentar pada satu baris kode dan /* untuk lebih dari satu baris kode agar tidak dijalankan oleh program. Ini sering digunakan untuk **"menyembunyikan"** (atau lebih tepatnya "mengabaikan") kode tertentu, memberi catatan, atau menjelaskan bagian kode. Contohnya:

```
//////////////////////////////////////////
/*
console.log("Ini tidak akan muncul");
console.log("Karena dikomentari!");
*/
 console.log("Hanya baris ini yang dieksekusi.");

Hasil Output: Hanya baris ini yang dieksekusi.
//////////////////////////////////////////
```

A. Deklarasi Function

Pembuatan function sebetulnya mirip dengan pembuatan variabel. Dalam variabel, nilai dapat kita akses dengan menyebutkan nama variabelnya. Hal ini karena memang kita menyimpan atau mengikat nilai dalam variabel tersebut. Sebuah function juga diikat dalam sebuah nama function. Dengan kata lain, nama function tersebut adalah identifier.

Dalam mendefinisikan function, kita perlu memahami anatominya lebih dahulu

- **Keyword function:**
  Kata kunci yang digunakan untuk mendefinisikan sebuah function.
- **Nama Function (Identifier):**
  Nama unik untuk mengidentifikasi fungsi. Sama seperti nama variabel, nama function harus deskriptif agar mudah dipahami.

- **Parameter:**
  Input yang diberikan ke function, ditulis di dalam tanda kurung ().
- **Badan Function (Block Code):**
  Kode yang berada di dalam kurung kurawal {}. Di sinilah logika fungsi ditulis.
- **Return Statement (Opsional):**
  Mengembalikan hasil dari function ke pemanggilnya.

Setelah kamu sudah tau teori dari anatotmi *function* di JavaScript sekarang kamu bisa langsung melihat ilustrasi dari sintaks kode JavaScript

Ilustrasi sintaks kodenya bisa kamu lihat dibawah:

```
//////////////////////////////////////////////
function hitungLuasPersegi(panjang, lebar) {
  let luas = panjang * lebar;
  return luas;
}

// Memanggil function
let hasil = hitungLuasPersegi(5, 10);
console.log(`Luas persegi: ${hasil}`);
// Output: Luas persegi: 50
//////////////////////////////////////////////
```

Penjelasan Kode Diatas

- function hitungLuasPersegi**(panjang, lebar)**:
  - function: Kata kunci untuk membuat fungsi.
  - hitungLuasPersegi: Nama fungsi.
  - panjang, lebar: Parameter yang akan menerima input saat fungsi dipanggil.
- let **luas = panjang * lebar;**: Kode di dalam fungsi untuk menghitung luas persegi panjang.
- return **luas;**: Mengembalikan hasil perhitungan ke pemanggil fungsi.

- **Memanggil Fungsi:** Ketika hitungLuasPersegi(5, 10) dipanggil, nilai panjang = 5 dan lebar = 10 diberikan sebagai parameter. Hasil 50 dikembalikan dan disimpan dalam variabel hasil.

Parameter dan Argument

Nah, dari tadi kan kita sudah belajar untuk mendeklarasikan. Nah, kita belum berkenalan secara resmi nih dengan si function. Sekarang kita akan mencoba untuk berkenalan lebih dalam dengan si function ini

Parameter adalah *syarat* input yang harus dimasukkan ke dalam suatu fungsi dan dideklarasikan bersama dengan deklarasi fungsi. Sementara

Argument adalah *nilai* yang dimasukan ke dalam suatu fungsi, sesuai dengan persyaratan parameter, di mana

argument dituliskan bersamaan dengan pemanggilan fungsi.

Hati-hati, istilah parameter dan argument suka dianggap sama, jadi pemakaian kedua kata ini suka terbalik-balik. Bisa diilustrasikan seperti ini:

```
////////////////////////////////////////////
function operasiPerkalian(angka1, angka2){
 return angka1 * angka2;
}

console.log(operasiPerkalian(2, 6)) // Output: 12
////////////////////////////////////////////
```

Penjelasan kode ilustrasi:

- angka1 & angka2 adalah **parameter**. Pada contoh di atas, parameter harus bertipe number, agar bisa diolah oleh fungsi, yaitu perkalian kedua parameter.
- 2 & 6 adalah **argument**. Sesuai kan dengan *syarat* parameter? Yap, mereka bertipe number.

B. Memanggil Function

Setelah kamu bisa membuat / mendeklarasikan sebuah function, nah kamu juga harus bisa untuk memanggil sebuah function yang sudah kamu buat

Melakukan deklarasi function tidak meminta JavaScript untuk menjalankannya. Lagi-lagi ini mirip dengan variabel, kita perlu menyebutkan identifier untuk memanggilnya.

Contoh cara memanggilnya:

```
//////////////////////////////////////////////
function buatKopi(jenisKopi, gula) {
 return `Kopi ${jenisKopi} dengan ${gula} gula siap disajikan!`;
}

// Memanggil fungsi
console.log(buatKopi("Espresso", "2 sendok"));
// Output: Kopi Espresso dengan 2 sendok gula siap disajikan!
console.log(buatKopi("Latte", "tanpa"));
// Output: Kopi Latte dengan tanpa gula siap disajikan!
//////////////////////////////////////////////
```

Kamu juga bisa coba kode tadi di kode editor kamu
Penjelasan dari kode yang tadi

1. Fungsi buatKopi menerima dua parameter: jenisKopi dan gula.
2. Fungsi mengembalikan string yang berisi detail kopi yang dibuat.
3. Fungsi dipanggil dengan memberikan nilai untuk jenisKopi dan gula, lalu hasilnya ditampilkan dengan console.log.

Function hoisting

Pernah nggak sih, kamu kepikiran gimana JavaScript bisa paham kalau kita panggil sebuah fungsi sebelum fungsinya didefinisikan? Nah, itu karena ada fitur keren yang namanya ***Function Hoisting.*** Ibaratnya, JavaScript itu fleksibel banget dan suka "ngertiin" kita, jadi dia otomatis "mengangkat" deklarasi fungsi ke bagian atas kode. Keren, kan?

"Apa kerennya? Perasaan biasa saja" mungkin kamu bakal mikir begitu. Contoh:

```
//////////////////////////////////////////////
console.log(operasiPerkalian(5, 5)); // Output: 25
function operasiPerkalian(angka1, angka2) {
  return angka1 * angka2;
}
//////////////////////////////////////////////
```

Secara logika kan, harusnya kita mendeklarasikan fungsi terlebih dahulu, setelah itu barulah kita menggunakan fungsi tersebut (pada contoh di atas, fungsi operasiPerkalian dipanggil di dalam console.log padahal deklarasinya ditulis setelah console.log). Nah, keistimewaan inilah yang disebut dengan *Function Hoisting.*

Tapi, ini tidak berlaku jika fungsi tersebut dideklarasi di dalam sebuah variabel. Contoh:

```
////////////////////////////////////////////
console.log(operasiPerkalian(5, 5));
 // Output: Uncaught ReferenceError: Cannot access
'operasiPerkalian' before initialization

const operasiPerkalian = function(angka1, angka2) {
 return angka1 * angka2;
};
//Keren kan!

////////////////////////////////////////////
```

Lingkup dalam JavaScript

Tadi kita sudah belajar tentang **function** di JavaScript, yang merupakan salah satu fitur paling penting dan fleksibel. Dengan function, kita bisa membuat potongan kode yang dapat digunakan kembali, mempermudah pengelolaan program, dan bikin kode kita jadi lebih rapi.

Nah, sekarang kita akan belajar **lingkup** dalam JavaScript—atau yang sering disebut **scope**. Pernah nggak sih, kamu bingung kenapa ada variabel yang hanya bisa diakses di bagian tertentu dari kode, sementara yang lain bisa dipakai di mana aja? Nah, jawabannya ada di konsep scope ini! Ibaratnya, setiap variabel atau fungsi itu punya "wilayah kekuasaan" sendiri, dan JavaScript tahu betul di mana mereka bisa bergerak.

Lingkup adalah konsep fundamental yang bikin JavaScript jadi lebih terstruktur dan efisien. Dengan memahami scope, kamu bisa lebih pintar dalam mengatur variabel dan fungsi agar nggak saling tumpang tindih. Mau variabel yang

cuma bisa diakses di dalam fungsi? Bisa! Atau yang bisa digunakan di mana saja? Juga bisa!. Lingkup dalam Js sendiri terbagi menjadi 2, yaitu:

- Global Scope

Variabel yang dideklarasikan di luar fungsi atau blok kode masuk ke dalam global scope. Artinya, variabel ini bisa diakses di mana saja dalam kode kamu. Tapi hati-hati, karena kalau kebanyakan variabel global, kode bisa jadi susah diatur!

Contoh dari Scope Global:

```
//////////////////////////////////////////
const game = 'Brawlhalla';

function namaPemain() {
 let pemain = 'Pavelski';
 const rankPavelski = 'Valhalla';
 console.log(game); // Output: Brawlhalla
 if (game === 'Brawlhalla') {
   pemain = 'Sandstorm';
   const rankSandstorm = 'World Champion';
   console.log(game); // Output: Brawlhalla
 }
 return pemain;
}

console.log(namaPemain()); // Output: Sandstorm
console.log(game); // Output: Brawlhalla
//////////////////////////////////////////
```

Penjelasan Global Scope Diatas:

- Variabel game dideklarasikan di luar fungsi namaPemain(), yang berarti game memiliki **global scope**. Ini artinya, variabel game bisa diakses dari mana saja dalam kode, termasuk di dalam fungsi namaPemain().
- Nilai dari game adalah 'Brawlhalla', dan ini dapat diakses di dalam fungsi atau di luar fungsi. Di dalam kode ini, kita melihat variabel game digunakan di dalam fungsi namaPemain() untuk memeriksa apakah nilai dari game adalah 'Brawlhalla'.

- Local Scope

Variabel yang dideklarasikan di dalam sebuah fungsi hanya bisa diakses dalam fungsi tersebut. Lingkup ini membantu menjaga agar variabel tidak "terlihat" di luar fungsi.

Contoh kodenya:

```
//////////////////////////////////////////
function namaAtlet() {
 const olahraga = 'basketball';
 let atlet = 'Lionel Messi';
 const noMessi = 10;
 console.log(olahraga); // Output: basketball

 if (olahraga === 'basketball') {
   atlet = 'Kobe Bryant';
   const noKobe = 24;
   console.log(olahraga); // Output: basketball
 }
```

```
 return atlet;
}

console.log(namaAtlet()); // Output: Kobe Bryant
// console.log(olahraga); // Error: olahraga is not defined
//////////////////////////////////////////////
```

- Fungsi namaAtlet() mendefinisikan variabel-variabel yang hanya dapat diakses di dalam fungsi tersebut. Variabel-variabel seperti olahraga, atlet, dan noMessi berada dalam **local scope** dari fungsi namaAtlet(), artinya hanya bisa diakses selama eksekusi dalam fungsi tersebut.
- Variabel olahraga adalah variabel **local** yang hanya bisa diakses dalam fungsi namaAtlet(). Ketika kita memanggil console.log(olahraga) di dalam fungsi, hasilnya adalah 'basketball'.

***

Object dan Array

A. Pengenalan Object

Dalam dunia nyata, kita sering menghadapi data yang kompleks dan beragam. Misalnya, ketika kita bekerja di sebuah kampus dan ingin mendata seorang mahasiswa, tentu kita perlu mencatat nama, usia, jurusan, serta alamat mahasiswa tersebut. Jika menggunakan variabel biasa, kita harus membuat satu variabel untuk setiap data.

```
//////////////////////////////////////////////
let nama = "Rina";
let usia = 21;
let jurusan = "Informatika";
let alamat = "Bandung";
//////////////////////////////////////////////
```

Namun, jika ada ratusan mahasiswa, pendekatan seperti ini akan sangat tidak efisien dan membingungkan.

Object hadir sebagai solusi dalam pengelolaan data yang kompleks. Dalam JavaScript, object adalah struktur data yang memungkinkan kita menyimpan banyak informasi dalam satu variabel secara rapi.

```
//////////////////////////////////////////////
let mahasiswa = {
  nama: "Rina",
  usia: 21,
  jurusan: "Informatika",
  alamat: "Bandung"
};
//////////////////////////////////////////////
```

Bayangkan sebuah **formulir biodata**. Setiap kolom seperti "Nama", "Usia", atau "Alamat" mewakili **key**, dan isian dari kolom tersebut adalah **value**. Object bekerja seperti formulir itu: menyimpan semua data terkait dalam satu paket.

B. Mengakses dan Mengubah Data dalam Object

Terdapat dua cara umum untuk mengakses data dalam object:

- **Dot notation**: object.key
- **Bracket notation**: object["key"]

Contoh:

```
//////////////////////////////////////////////
console.log(mahasiswa.nama);      // Output: Rina
console.log(mahasiswa["jurusan"]);
// Output: Informatika
//////////////////////////////////////////////
```

Untuk menambah atau mengubah properti:

```
//////////////////////////////////////////////
mahasiswa.ipk = 3.8;
mahasiswa.usia = 22;
//////////////////////////////////////////////
```

Untuk menghapus properti:

```
//////////////////////////////////////////////
delete mahasiswa.alamat;
//////////////////////////////////////////////
```

C. Object Bersarang (Nested Object)

Sering kali, satu properti bisa menyimpan data yang juga kompleks. Misalnya, alamat bisa terdiri dari kota, provinsi, dan kode pos:

```
////////////////////////////////////////////////
let mahasiswa = {
 nama: "Rina",
 alamat: {
  kota: "Bandung",
  provinsi: "Jawa Barat",
  kodePos: 40123
 }
};
////////////////////////////////////////////////
```

Untuk mengakses kota:

```
////////////////////////////////////////////////
console.log(mahasiswa.alamat.kota);
// Output: Bandung
////////////////////////////////////////////////
```

D. Method pada Object

Object tidak hanya menyimpan data, tapi juga bisa menyimpan fungsi sebagai method.

```
//////////////////////////////////////////////
let kalkulator = {
  tambah: function(a, b) {
    return a + b;
  },
  kurang: function(a, b) {
    return a - b;
  }
};

console.log(kalkulator.tambah(5, 3)); // Output: 8
//////////////////////////////////////////////
```

Analogi Dunia Nyata:
Jika object adalah mobil, maka method adalah fitur-fiturnya seperti nyalakanMesin(), jalan(), atau rem(). Mereka melakukan tindakan terhadap data dalam mobil itu.

E. Pengenalan Array

Array adalah struktur data yang digunakan untuk menyimpan kumpulan data dalam satu variabel. Berbeda dengan object, array lebih cocok digunakan jika datanya bersifat daftar atau urutan.

```
//////////////////////////////////////////////
let buah = ["Apel", "Jeruk", "Mangga"];
//////////////////////////////////////////////
```

Array diakses dengan index:

```
//////////////////////////////////////////////
console.log(buah[0]); // Output: Apel
//////////////////////////////////////////////
```

F. Method Pada Array

Berikut beberapa method penting pada array:

Method	Fungsi
push()	Menambah elemen ke akhir array
pop()	Menghapus elemen terakhir
shift()	Menghapus elemen pertama
unshift()	Menambahkan elemen ke awal array
splice()	Menyisipkan atau menghapus elemen tertentu
slice()	Mengambil sebagian elemen
map()	Memetakan array ke array baru
filter()	Menyaring elemen berdasarkan kondisi

Contoh:

```
//////////////////////////////////////////////
let angka = [1, 2, 3, 4, 5];

let ganda = angka.map(function(n) {
  return n * 2;
});
// Output: [2, 4, 6, 8, 10]

let genap = angka.filter(function(n) {
  return n % 2 === 0;
});
// Output: [2, 4]
//////////////////////////////////////////////
```

Analogi Dunia Nyata:
Bayangkan kamu punya daftar belanja: ["Susu", "Roti", "Gula"]

- push("Telur") = Tambah telur ke daftar
- pop() = Hapus item terakhir (Telur)
- filter() = Hanya ambil item yang harganya di bawah Rp10.000
- map() = Ubah semua item jadi huruf kapital

G. Array of Object

Kadang kita ingin menyimpan banyak object ke dalam satu array. Contohnya data mahasiswa lebih dari satu:

Contoh:

```
////////////////////////////////////////////
let dataMahasiswa = [
 { nama: "Rina", jurusan: "Informatika" },
 { nama: "Budi", jurusan: "Sistem Informasi" }
];
////////////////////////////////////////////
```

Akses data:

```
////////////////////////////////////////////
console.log(dataMahasiswa[0].nama); // Output: Rina
////////////////////////////////////////////
```

***

Percabangan dan Pengulangan (Control Flow)

A. Percabangan (Conditional Statements)

Dalam kehidupan sehari-hari, kita sering dihadapkan pada keputusan. Misalnya:

"Kalau hari hujan, saya akan membawa payung. Kalau tidak hujan, saya tidak akan membawanya."

Logika seperti ini dapat diterapkan dalam program menggunakan **percabangan**. Di JavaScript, kita memiliki beberapa cara untuk membuat percabangan:

- if, else if, else

```
////////////////////////////////////////////
let cuaca = "hujan";

if (cuaca === "hujan") {
  console.log("Bawa payung!");
} else if (cuaca === "panas") {
  console.log("Pakai kacamata hitam!");
} else {
  console.log("Cuaca tidak menentu.");
}
////////////////////////////////////////////
```

- switch statement

```
////////////////////////////////////////////
let hari = "Senin";

switch (hari) {
  case "Senin":
    console.log("Mulai kerja!");
    break;
  case "Sabtu":
    console.log("Saatnya bersantai.");
    break;
  default:
    console.log("Hari biasa.");
////////////////////////////////////////////
```

- Ternary Operator

```
//////////////////////////////////////////////
let umur = 18;
let status = (umur >= 17) ? "Dewasa" : "Anak-anak";

console.log(status); // Output: Dewasa
//////////////////////////////////////////////
```

Analogi Dunia Nyata:
"Jika kamu punya uang cukup, beli. Jika tidak, jangan beli."
Dalam kode: let beli = (uang >= harga) ? "Beli" : "Tunda dulu";

B. Pengulangan (Looping)

Pengulangan digunakan saat kita ingin menjalankan kode **berulang kali**. Misalnya, mencetak daftar nama murid, menghitung rata-rata nilai, atau memproses data dalam array.

- for loop

```
//////////////////////////////////////////////
for (let i = 1; i <= 5; i++) {
 console.log("Nomor ke-" + i);
//////////////////////////////////////////////
```

Akan mencetak:

```
////////////////////////////////////////////
Nomor ke-1
Nomor ke-2
Nomor ke-3
Nomor ke-4
Nomor ke-5
////////////////////////////////////////////
```

- while loop

```
////////////////////////////////////////////
let angka = 1;

while (angka <= 5) {
  console.log("Angka " + angka);
  angka++;
}
////////////////////////////////////////////
```

- do...while loop

```
////////////////////////////////////////////
let angka = 1;

do {
  console.log("Angka ke-" + angka);
  angka++;
} while (angka <= 3);
////////////////////////////////////////////
```

Catatan:

*do...while akan mengeksekusi blok kode setidaknya **sekali**, walau kondisi tidak terpenuhi.*

C. Loop Khusus untuk Array dan Object

- for...of → untuk Array

```
//////////////////////////////////////////////
let buah = ["Apel", "Mangga", "Jeruk"];

for (let item of buah) {
  console.log(item);
}
//////////////////////////////////////////////
```

- for...in → untuk Object

```
//////////////////////////////////////////////
let mahasiswa = {
  nama: "Budi",
  umur: 20,
  jurusan: "TI"
};

for (let key in mahasiswa) {
  console.log(key + ": " + mahasiswa[key]);
}
//////////////////////////////////////////////
```

D. Break dan Continue

Kadang kita butuh menghentikan atau melewatkan iterasi dalam loop.

```
////////////////////////////////////////////////
for (let i = 1; i <= 5; i++) {
 if (i === 3) {
  continue; // Lewatkan angka 3
 }
 console.log(i);
}
////////////////////////////////////////////////
```

```
////////////////////////////////////////////////
for (let i = 1; i <= 5; i++) {
 if (i === 4) {
  break; // Hentikan loop saat i = 4
 }
 console.log(i);
}
////////////////////////////////////////////////
```

***

JavaScript Intermediate

Setelah memahami dasar-dasar JavaScript, kini saatnya kita melangkah ke tingkat yang lebih lanjut. Di bagian **JavaScript Intermediate** ini, kita akan mempelajari konsep-konsep penting seperti manipulasi DOM, event handling, asynchronous programming, hingga komunikasi dengan server menggunakan API.

Materi-materi ini akan menjadi bekal utama untuk membangun aplikasi web yang dinamis dan interaktif. Dengan pendekatan berbasis analogi dan studi kasus, diharapkan setiap konsep dapat dipahami lebih dalam dan aplikatif.

Mari lanjutkan perjalanan kita menuju penguasaan JavaScript yang lebih profesional.

Error Handling dan Debugging

Nah, setelah kita belajar tentang operator, sekarang saatnya kita bahas hal yang nggak kalah penting, yaitu *error handling* alias penanganan error. Dalam dunia pemrograman, error itu hal yang wajar banget. Yang penting bukan *"bagaimana cara menghindari error"*, tapi *"gimana cara menanganinya dengan elegan"*

A. Jenis-Jenis Error di JavaScript

Sebelum kita bahas gimana cara menanganinya, yuk kenalan dulu sama jenis-jenis error yang biasa muncul di JavaScript:

Jenis Error	Penjelasan
SyntaxError	Terjadi karena ada kesalahan dalam penulisan sintaks (tanda kurung kurang, dll)
ReferenceError	Mengakses variabel yang belum dideklarasikan
TypeError	Melakukan operasi yang salah terhadap tipe data (misal: angka dikali string)
RangeError	Biasanya karena nilai di luar jangkauan yang bisa diterima
EvalError	Error yang berkaitan dengan fungsi eval() (jarang dipakai)
URIError	Error pada fungsi decodeURI / encodeURI dengan parameter yang tidak valid

B. Menangani Error dengan try...catch

Nah, cara yang paling umum untuk menangani error adalah dengan **try...catch**. Gampang banget syntax-nya:

```
////////////////////////////////////////////
try {
  // kode yang mungkin error
  let hasil = 10 / 0;
  console.log("Hasil:", hasil);
} catch (error) {
  // kode yang dijalankan kalau terjadi error
```

```
  console.log("Terjadi error:", error.message);
}
//////////////////////////////////////////////
```

Penjelasan:

- Blok try berisi kode yang berpotensi menimbulkan error.
- Kalau error beneran terjadi, program langsung lompat ke blok catch.
- error.message akan kasih tahu pesan error-nya secara jelas.

Analogi dunia nyata:
Bayangin kamu lagi nyetir mobil ke kantor. Tiba-tiba bannya kempes.

- Kalau kamu **nggak punya plan**, ya mogok di pinggir jalan. 🛑
- Tapi kalau kamu **punya ban serep dan dongkrak**, kamu bisa lanjut perjalanan setelah ganti ban. ✅

Nah, try...catch itu ibarat ban serep dan dongkrak dalam kode kita. Kita tetap bisa "jalan" meskipun terjadi masalah di tengah jalan.

C. Debugging Tools

Debugging adalah proses mencari dan memperbaiki error di dalam kode program.

Kadang, error nggak selalu langsung kelihatan. Makanya kita perlu alat bantu debugging. Beberapa tools yang sering dipakai:

- **console.log()**
  Teman sejuta umat. Dipakai untuk menampilkan nilai variabel atau ngecek alur kode.
- **Debugger Statement**
  JavaScript punya keyword debugger; yang bisa kamu sisipkan di kode. Kalau dibuka di browser dengan Developer Tools, eksekusi akan berhenti di situ, dan kamu bisa "ngintip" isi variabel satu-satu.
- **Developer Tools di Browser**
  Tekan F12 atau Ctrl + Shift + I di browser. Di tab "Console" dan "Sources", kamu bisa lihat log, error, dan bahkan set breakpoint.

Salah satu cara paling sederhana dan sering digunakan programmer buat mencari tahu jalannya program adalah dengan console.log().

Contoh:

```
////////////////////////////////////////////////
let nama = "Budi";
console.log("Nama: ", nama);

let nilai = 80;
console.log("Nilai: ", nilai);




if (nilai >= 75) {
  console.log("Lulus!");
} else {
  console.log("Tidak lulus.");
```

```
}

Output:

Nama: Budi
Nilai: 80
Lulus!
//////////////////////////////////////////////
```

Biar lebih gampang, bayangin kalau kita bikin teh manis:

1. **Tanpa gula:** Rasanya hambar → kayak **ReferenceError**, ada yang hilang.
2. **Pakai garam:** Rasanya aneh → kayak **TypeError**, bahan salah.
3. **Menuang tanpa gelas:** Tumpah semua → kayak **SyntaxError**, prosedur salah.

Makanya, kita perlu cek dulu bahan dan alatnya sebelum bikin, sama kayak ngecek kode sebelum dan selama dijalankan.

***

DOM (Document Object Model)

A. Pengertian DOM

Bayangkan kamu sedang melihat bangunan rumah dari luar. Kamu bisa melihat pintu, jendela, atap, bahkan taman. Sekarang bayangkan kalau kamu punya **remote** yang bisa membuka pintu, menutup jendela, menyalakan lampu, dan mengubah warna cat secara langsung **itulah yang dilakukan DOM untuk JavaScript terhadap HTML.**

DOM adalah singkatan dari **Document Object Model**, yaitu representasi struktur halaman web dalam bentuk **pohon (tree)**. Setiap elemen dalam HTML seperti paragraf <p>, gambar <img>, tombol <button> akan direpresentasikan sebagai **node** di dalam pohon DOM. JavaScript dapat mengakses node-node ini dan melakukan **manipulasi** terhadapnya: menambah, menghapus, atau mengubah kontennya.

Analogi dunia nyata:
Bayangkan halaman HTML sebagai tubuh manusia. DOM adalah "kerangkanya", dan JavaScript adalah tangan kita yang bisa menggerakkan bagian-bagian tubuh itu sesuai keinginan.

B. Mengakses Elemen DOM

Untuk memanipulasi HTML, pertama-tama kita harus **mengakses elemen-nya** lewat JavaScript. Berikut adalah cara umum:

- getElementById()

```
/////// HTML ////////////////////////////////
<p id="judul">Halo Dunia</p>
/////////////////////////////////////////////
```

```
/////// JavaScript //////////////////////////
let elemen = document.getElementById("judul");
console.log(elemen.innerText); // Output: Halo Dunia
/////////////////////////////////////////////
```

- getElementsByClassName()

```
/////// HTML ////////////////////////////////
<p class="teks">Paragraf 1</p>
<p class="teks">Paragraf 2</p>
/////////////////////////////////////////////
```

```
/////// JavaScript //////////////////////////
let elemen =
document.getElementsByClassName("teks");
console.log(elemen[0].innerText); // Output: Paragraf
1
/////////////////////////////////////////////
```

- querySelector() dan querySelectorAll()

```
/////// HTML ////////////////////////////////
<div class="container">
  <p id="teks-utama">Selamat Datang</p>
</div>
/////////////////////////////////////////////
```

```
/////// JavaScript //////////////////////////
let teks = document.querySelector("#teks-utama");
console.log(teks.innerText); // Output: Selamat Datang
/////////////////////////////////////////////
```

Note:
querySelector() hanya mengambil elemen pertama yang cocok, sedangkan querySelectorAll() mengambil semua elemen yang cocok dalam bentuk NodeList.

C. Mengubah Konten dan Properti Elemen

Setelah elemen didapat, kita bisa:

- Mengubah Teks dan HTML

```
/////// JavaScript ////////////////////////////////
let paragraf = document.getElementById("judul");
paragraf.innerText = "Judul Baru"; // Ganti teks
paragraf.innerHTML = "<strong>Judul
Tebal</strong>"; // Ganti HTML
////////////////////////////////////////////////////
```

- Mengubah Atribut

```
/////// HTML ////////////////////////////////////
<img id="gambar" src="lama.jpg">
////////////////////////////////////////////////////
```

```
/////// JavaScript ////////////////////////////////
let gambar = document.getElementById("gambar");
gambar.src = "baru.jpg";
gambar.alt = "Gambar baru";
////////////////////////////////////////////////////
```

- Mengubah Style (CSS)

```
/////// JavaScript ////////////////////////////////
let elemen = document.getElementById("judul");
elemen.style.color = "blue";
```

```
elemen.style.fontSize = "24px";
//////////////////////////////////////////////
```

D. Menambahkan dan Menghapus Elemen

DOM juga memungkinkan kita menambahkan atau menghapus elemen dari halaman.

- Membuat Elemen Baru

```
/////// JavaScript ////////////////////////////////
let pBaru = document.createElement("p");
pBaru.innerText = "Ini paragraf baru";
document.body.appendChild(pBaru); // Tambahkan ke akhir body
//////////////////////////////////////////////////
```

- Menghapus Elemen

```
/////// JavaScript ////////////////////////////////
let target = document.getElementById("judul");
target.remove();
//////////////////////////////////////////////////
```

Studi Kasus Ringan: Ubah Warna Tema

```
/////// HTML ////////////////////////////////////
<button onclick="ubahTema()">Ubah Tema</button>
<script>
function ubahTema() {
```

```
 document.body.style.backgroundColor = "black";
 document.body.style.color = "white";
}
</script>
////////////////////////////////////////////////
```

Dengan satu tombol, DOM memungkinkan kita **mengubah seluruh suasana tampilan halaman** seperti mengganti cat rumah hanya dengan satu klik.

E. Struktur DOM Tree (Pohon DOM)

Untuk membayangkan DOM secara visual, bayangkan seperti pohon keluarga:

- document
  - html
    - head
      - title
    - body
      - h1
      - p
      - div
        - img
        - a

Setiap elemen adalah **node**, dan punya relasi seperti:

- **parentNode** (induk),
- **childNodes** (anak),
- **nextSibling / previousSibling** (saudara).

F. Event DOM dan Interaktifitas Dasar

```
/////// JavaScript ////////////////////////////
let tombol = document.getElementById("klikSaya");
tombol.addEventListener("click", function() {
  alert("Tombol diklik!");
});
////////////////////////////////////////////////
```

***

Event Handling

A. Pengertian Event Handling

Dalam dunia nyata, sebuah peristiwa bisa terjadi kapan saja: seseorang menekan tombol, membuka pintu, atau menyalakan lampu. Dalam dunia pemrograman web, **peristiwa** seperti ini disebut **event**, dan JavaScript memiliki kemampuan untuk **mendengarkan** dan **menanggapi** peristiwa-peristiwa tersebut. Inilah yang disebut dengan **Event Handling**.

Event adalah segala aksi atau interaksi pengguna terhadap halaman web, seperti klik, scroll, ketikan di keyboard, submit form, dan lainnya.

Event Handling adalah proses menangkap event tersebut dan memberikan respon tertentu melalui kode JavaScript.

Bayangkan kamu memiliki bel pintu. Ketika seseorang menekan tombol bel (event), bel tersebut berbunyi (response). JavaScript bekerja seperti sistem bel itu — mendeteksi event dan menjalankan tindakan sebagai responnya.

B. Cara Menangani Event

- Menggunakan atribut HTML:

Cara paling dasar adalah menulis event langsung di HTML.

```
//////////////////////////////////////////////
<button onclick="sapa()">Klik Saya</button>
<script>
function sapa() {
  alert("Halo, Selamat datang!");
}
</script>
//////////////////////////////////////////////
```

- Menggunakan addEventListener()

Cara yang lebih fleksibel dan disarankan adalah menggunakan addEventListener() di JavaScript.

```
/////// HTML ////////////////////////////////
<button id="tombol">Klik Saya</button>
//////////////////////////////////////////////
```

```
/////// JavaScript ////////////////////////////
let btn = document.getElementById("tombol");
btn.addEventListener("click", function() {
  alert("Halo juga!");
});
////////////////////////////////////////////////
```

C. Jenis-Jenis Event Populer

Berikut adalah beberapa event yang sering digunakan:

Event	Deskripsi
click	Ketika elemen diklik
mouseover	Saat kursor diarahkan ke elemen
mouseout	Saat kursor keluar dari elemen
keydown	Saat tombol keyboard ditekan
keyup	Saat tombol keyboard dilepas
submit	Ketika form disubmit
change	Ketika nilai input berubah
load	Ketika halaman selesai dimuat

Studi Kasus: Ubah Warna Saat Diklik

```
/////// HTML ////////////////////////////////
<button id="gantiWarna">Ganti Warna</button>
////////////////////////////////////////////////
```

```
/////// JavaScript ////////////////////////////
let tombol = document.getElementById("gantiWarna");
tombol.addEventListener("click", function() {
 document.body.style.backgroundColor = "lightblue";
});
////////////////////////////////////////////////
```

D. Event Object (Objek Event)

Saat event terjadi, JavaScript sebenarnya menyimpan detail informasi tentang event tersebut dalam sebuah objek event. Objek ini bisa diakses lewat parameter di dalam fungsi.

```
/////// JavaScript ////////////////////////////
document.addEventListener("click", function(e) {
 console.log("Koordinat klik:", e.clientX, e.clientY);
});
////////////////////////////////////////////////
```

Informasi dari event object bisa sangat membantu, misalnya untuk mengetahui elemen mana yang diklik, posisi mouse, atau tombol keyboard mana yang ditekan.

Studi Kasus Tambahan: Validasi Form

```
/////// HTML ////////////////////////////////
<form id="formulir">
 <input type="text" id="email" placeholder="Email...">
 <button type="submit">Kirim</button>
</form>
<p id="error"></p>
////////////////////////////////////////////////
```

```
/////// JavaScript ////////////////////////////
let form = document.getElementById("formulir");
let email = document.getElementById("email");
let error = document.getElementById("error");

form.addEventListener("submit", function(e) {
 if (email.value === "") {
  e.preventDefault(); // Cegah form dikirim
  error.innerText = "Email tidak boleh kosong!";
 }
});
/////////////////////////////////////////////
```

Dalam dunia nyata, ini seperti satpam yang tidak membiarkan orang masuk jika mereka belum mengisi data dengan benar.

***

Form Validation

A. Pengertian Form Validation

Dalam kehidupan sehari-hari, kita sering diminta mengisi formulir — entah itu formulir pendaftaran sekolah, formulir pembelian barang, atau formulir registrasi acara. Biasanya, ada syarat atau aturan yang harus dipenuhi: misalnya, nama tidak boleh kosong, nomor telepon harus terdiri dari angka, atau email harus dalam format tertentu.

Nah, dalam web, kita juga menggunakan formulir (form) untuk mengumpulkan data dari pengguna. Form Validation adalah proses memeriksa apakah data yang dimasukkan pengguna sesuai dengan aturan yang

ditentukan sebelum formulir dikirimkan.

Form Validation penting untuk mencegah data kosong, salah format, atau tidak valid masuk ke server. Ini seperti penjaga gerbang sebelum data masuk ke sistem.

B. Jenis Validasi

- Validasi Client-side
  Dilakukan di sisi pengguna menggunakan JavaScript sebelum data dikirim. Lebih cepat dan ramah pengguna.
- Validasi Server-side
  Dilakukan di sisi server (PHP, Node.js, dll). Lebih aman karena tidak bisa dimanipulasi oleh pengguna.

Studi Kasus: Validasi Formulir Pendaftaran

```
/////// HTML ////////////////////////////////
<form id="formDaftar">
 <label>Nama:</label>
 <input type="text" id="nama"><br><br>
 <label>Email:</label>
 <input type="text" id="email"><br><br>
 <button type="submit">Daftar</button>
 <p id="errorMsg" style="color:red;"></p>
</form>
//////////////////////////////////////////////
```

```
/////// JavaScript ////////////////////////////
let form = document.getElementById("formDaftar");
let nama = document.getElementById("nama");
let email = document.getElementById("email");
let errorMsg =
document.getElementById("errorMsg");

form.addEventListener("submit", function(e) {
 let pesan = "";

 if (nama.value === "") {
  pesan += "Nama tidak boleh kosong. ";
 }

 if (!email.value.includes("@")) {
  pesan += "Email tidak valid.";
 }

 if (pesan !== "") {
  e.preventDefault();
  errorMsg.innerText = pesan;
 }
});
//////////////////////////////////////////////
```

Analogi Dunia Nyata

Formulir yang divalidasi bisa diibaratkan seperti petugas keamanan di bandara. Sebelum masuk ke area keberangkatan, semua penumpang diperiksa apakah membawa dokumen yang sah. Jika tidak lengkap, mereka tidak diperbolehkan melanjutkan perjalanan. Demikian pula, form validation memastikan data yang masuk sudah benar dan layak diproses.

Validasi Lebih Canggih: Pola Regex (Regular Expression)

```
/////// JavaScript ////////////////////////////
let emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
if (!emailPattern.test(email.value)) {
  pesan += "Format email tidak sesuai.";
}
////////////////////////////////////////////////
```

Menampilkan Error Secara Real-Time

```
/////// JavaScript ////////////////////////////
email.addEventListener("input", function() {
  if (email.value.includes("@")) {
    errorMsg.innerText = "";
  } else {
    errorMsg.innerText = "Email belum valid.";
  }
});
////////////////////////////////////////////////
```

***

Web Storage: Local Storage dan Session Storage

A. Pengertian Web Storage

Bayangkan kamu sedang mengisi keranjang belanja di sebuah toko online. Saat kamu keluar dari website, lalu kembali lagi beberapa jam kemudian, kamu masih menemukan barang-barang yang kamu pilih sebelumnya tetap ada di keranjangmu. Hal ini bisa terjadi berkat **Web Storage**.

Web Storage adalah fitur yang disediakan oleh browser untuk menyimpan data secara lokal di perangkat pengguna. Tidak seperti cookie yang ikut terkirim ke server pada setiap request, data di Web Storage **hanya tersimpan di browser** dan **tidak secara otomatis dikirim ke server**.

Web Storage terbagi menjadi dua jenis:

- **Local Storage** – menyimpan data secara permanen (sampai pengguna menghapus atau browser cache dibersihkan).
- **Session Storage** – menyimpan data hanya selama sesi browser berlangsung (hilang saat tab ditutup).

B. Perbandingan Local Storage dan Session Storage

Fitur	Local Storage	Session Storage
Masa Penyimpanan	Permanen	Sementara (selama sesi)
Tersedia di Tab Baru	Ya	Tidak
Ukuran Maksimum	Sekitar 5MB	Sekitar 5MB
Tersimpan di Disk?	Ya	Tidak (hanya di memori)

C. Cara Menggunakan Local Storage

- Menyimpan Data

```
/////// JavaScript ////////////////////////////////
localStorage.setItem("nama", "Dian");
////////////////////////////////////////////////////
```

- Mengambil Data

```
/////// JavaScript ////////////////////////////////
let nama = localStorage.getItem("nama");
console.log(nama); // Output: Dian
////////////////////////////////////////////////////
```

- Menghapus Data

```
/////// JavaScript //////////////////////////////
localStorage.removeItem("nama");
////////////////////////////////////////////////
```

- Menghapus Semua Data

```
/////// JavaScript //////////////////////////////
localStorage.clear();
////////////////////////////////////////////////
```

D. Cara Menggunakan Session Storage

Mirip seperti Local Storage, hanya saja menggunakan sessionStorage:

```
/////// JavaScript //////////////////////////////
sessionStorage.setItem("login", "true");

let status = sessionStorage.getItem("login");
console.log(status); // Output: true

sessionStorage.removeItem("login");
////////////////////////////////////////////////
```

Studi Kasus: Menyimpan Nama Pengguna

```
/////// HTML ////////////////////////////
<input type="text" id="inputNama"
placeholder="Masukkan nama Anda">
<button onclick="simpanNama()">Simpan</button>
<p id="salam"></p>
//////////////////////////////////////////
```

```
/////// JavaScript ////////////////////////////
function simpanNama() {
 let nama =
document.getElementById("inputNama").value;
 localStorage.setItem("userNama", nama);
 tampilkanSalam();
}

function tampilkanSalam() {
 let nama = localStorage.getItem("userNama");
 if (nama) {
   document.getElementById("salam").innerText =
"Halo, " + nama + "!";
 }
}

tampilkanSalam(); // Panggil saat halaman dibuka
//////////////////////////////////////////////
```

Ketika halaman direfresh atau dibuka ulang, salam masih tetap muncul tanpa harus mengetik nama ulang. Inilah kekuatan Local Storage.

Analogi Dunia Nyata:

- **Local Storage** seperti menyimpan data di **loker pribadi** di pusat kebugaran. Kamu bisa keluar dan kembali kapan saja, dan barangmu masih ada di sana.
- **Session Storage** seperti **meja kerja hotel sementara** — saat kamu check-out (menutup tab), semua barangmu langsung dibersihkan.

E. Kelebihan dan Kekurangan Web Storage

Kelebihan:

- Mudah digunakan.
- Tidak membebani server.
- Cocok untuk menyimpan data ringan seperti preferensi pengguna.

Kekurangan:

- Tidak cocok untuk data sensitif (tidak terenkripsi).
- Kapasitas terbatas (sekitar 5MB).
- Tidak tersedia di browser lama.

Web Storage memungkinkan kita membuat aplikasi web yang lebih interaktif dan personal, seperti mengingat preferensi pengguna, menyimpan data sementara saat pengisian form, dan sebagainya. Dengan Local Storage dan Session Storage, JavaScript dapat menyimpan data tanpa perlu koneksi ke server.

***

Asynchronous JavaScript

A. Pengertian Asynchronous

Dalam dunia nyata, bayangkan kamu memesan kopi di kafe. Setelah memesan, kamu **tidak** berdiri di depan kasir menunggu kopimu selesai dibuat. Sebaliknya, kamu duduk, ngobrol, atau mengerjakan sesuatu sambil menunggu. Ketika kopi siap, barista akan memanggil namamu.

Inilah konsep **Asynchronous (asinkron)**: kita bisa **melanjutkan aktivitas lain** tanpa harus menunggu satu tugas selesai.

Di JavaScript, asynchronous artinya kode **tidak harus dijalankan berurutan**, melainkan bisa menunggu sesuatu (seperti data dari server) tanpa menghentikan program utama.

B. Callback

Callback adalah **fungsi yang dikirimkan sebagai argumen** ke fungsi lain, yang akan **dipanggil setelah tugas selesai**.

Contoh Callback:

```
////////////////////////////////////////////
function salam(callback) {
  console.log("Halo, selamat datang!");
  callback();
}
```

```
function lanjut() {
  console.log("Silakan menikmati layanan kami.");
}

salam(lanjut);
////////////////////////////////////////////////
```

```
////////////////////////////////////////////////
Output:

Halo, selamat datang!
Silakan menikmati layanan kami.
////////////////////////////////////////////////
```

C. Masalah Callback

Kalau tugasnya banyak dan bertingkat, callback menjadi **sangat rumit** dibaca:

```
////////////////////////////////////////////////
pesanKopi(function(order) {
  buatKopi(order, function(result) {
    minumKopi(result, function() {
      bayarKopi(function() {
        console.log("Semua selesai!");
      });
    });
  });
});
////////////////////////////////////////////////
```

Ini disebut **Callback Hell** — struktur bertingkat yang sulit dipahami.

D. Promise

Promise adalah objek di JavaScript yang mewakili **janji** bahwa operasi asinkron akan selesai di masa depan, dengan 2 kemungkinan:

- **Fulfilled** (berhasil)
- **Rejected** (gagal)

Membuat Promise:

```
////////////////////////////////////////////////
let janji = new Promise(function(resolve, reject) {
  let sukses = true;

  if (sukses) {
    resolve("Berhasil!");
  } else {
    reject("Gagal!");
  }
});

janji
  .then(function(result) {
    console.log(result); // Output: Berhasil!
  })
  .catch(function(error) {
    console.log(error); // Output jika gagal
  });
////////////////////////////////////////////////
```

Studi Kasus Promise: Pesan Kopi

```
//////////////////////////////////////////////
function pesanKopi() {
  return new Promise(function(resolve, reject) {
    let tersedia = true;
    setTimeout(() => {
      if (tersedia) {
        resolve("Kopi sudah jadi!");
      } else {
        reject("Kopi habis.");
      }
    }, 3000);
  });
}

pesanKopi()
  .then(res => console.log(res))
  .catch(err => console.log(err));
//////////////////////////////////////////////
```

Analogi Dunia Nyata:
Kamu pesan kopi → Kasir berjanji → Setelah 3 menit → Kamu diberi kabar.

E. Async / Await

Async/Await adalah cara modern menulis kode asynchronous yang **lebih sederhana** dan **lebih mirip kode sinkron biasa**.

- async digunakan untuk menandai fungsi yang mengandung await.
- await digunakan untuk menunggu Promise selesai.

Contoh Async Await:

```
////////////////////////////////////////////////
async function mulai() {
  try {
    let hasil = await pesanKopi();
    console.log(hasil);
  } catch (error) {
    console.log(error);
  }
}

mulai();
////////////////////////////////////////////////
```

Promise ibarat kamu diberikan **nomor antrian** saat mengantri di bank. Kamu bisa duduk santai sambil menunggu nomor dipanggil.
Async/Await membuat proses ini terasa seperti kamu ngobrol normal dengan petugas tanpa merasa sedang "menunggu" di antrian.

***

Fetch API dan AJAX

A. Pengertian Fetch API dan AJAX

Bayangkan kamu memesan makanan lewat aplikasi ojek online. Kamu tidak perlu pergi ke restoran, cukup tekan tombol, lalu informasi pesanan diambil dan tampil di layar. Dalam dunia web, **Fetch API** dan **AJAX** berfungsi **mengambil atau mengirim data ke server** tanpa harus me-reload seluruh halaman.

AJAX (Asynchronous JavaScript and XML) adalah teknik lama, sedangkan **Fetch API** adalah cara modern yang lebih sederhana dan berbasis Promise.

B. Perbedaan Fetch API dengan AJAX?

	AJAX (dengan XMLHttpRequest)	**Fetch API**
Sintaks	Kompleks dan panjang	Ringkas dan berbasis Promise
Dukungan Browser	Lebih luas (browser lama)	Modern (semua browser terbaru)
Mudah Dibaca	Kurang	Lebih mudah

C. Contoh Penggunaan Fetch API

- Mengambil Data dari Server

```
//////////////////////////////////////////////
fetch('https://jsonplaceholder.typicode.com/posts')
 .then(response => response.json())
 .then(data => console.log(data))
 .catch(error => console.log('Error:', error));
//////////////////////////////////////////////
```

- fetch mengirim permintaan ke server.
- .then(response => response.json()) mengubah respons menjadi objek JavaScript.
- .catch menangkap jika ada error.

Studi Kasus: Menampilkan Data Postingan

```
/////// HTML ////////////////////////////////
<button onclick="ambilData()">Ambil Data</button>
<ul id="listData"></ul>
//////////////////////////////////////////////
```

```
/////// JavaScript ////////////////////////////
function ambilData() {
 fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json())
  .then(posts => {
   let list = document.getElementById('listData');
   list.innerHTML = "";
   posts.slice(0, 5).forEach(post => {
```

```
      let item = document.createElement('li');
      item.innerText = post.title;
      list.appendChild(item);
    });
  })
  .catch(error => console.log('Gagal:', error));
}
////////////////////////////////////////////////
```

Saat tombol diklik, daftar postingan akan muncul **tanpa me-refresh halaman**.

D. Mengirim Data ke Server dengan Fetch

Untuk **mengirim data** (POST request), kita bisa menulis:

```
////////////////////////////////////////////////
fetch('https://jsonplaceholder.typicode.com/posts', {
 method: 'POST',
 headers: {
  'Content-Type': 'application/json'
 },
 body: JSON.stringify({
  title: 'Belajar Fetch',
  body: 'Ini adalah percobaan POST',
  userId: 1
 })
})
```

```
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.log('Error:', error));
//////////////////////////////////////////////
```

Analogi Dunia Nyata

- **Fetch** itu seperti **kamu menelepon restoran** untuk memesan makanan, lalu menunggu konfirmasi.
- Jika pesanan sukses, kamu mendapat makanan. Jika gagal (misal restoran tutup), kamu menerima pemberitahuan error.

Fetch API bekerja di latar belakang tanpa mengganggu halaman utama, sama seperti kamu bisa tetap ngobrol sambil menunggu konfirmasi dari restoran.

E. Perbedaan GET dan POST

Metode	Keterangan
GET	Mengambil data dari server.
POST	Mengirim data baru ke server.

- GET → Seperti meminta menu makanan.
- POST → Seperti mengirimkan pesanan ke dapur.

Fetch API memberikan cara yang lebih mudah, modern, dan powerful untuk berkomunikasi dengan server tanpa reload halaman.
Menguasai Fetch adalah kunci untuk membangun aplikasi web modern yang dinamis dan responsif.

***

JSON

A. Pengertian JSON

Bayangkan kamu menulis daftar belanja ke temanmu lewat pesan WhatsApp. Kamu ingin daftar itu:

- **Mudah dibaca**,
- **Tertata rapi**, dan
- **Bisa langsung dipahami** tanpa perlu penjelasan panjang.

Dalam dunia web, data juga perlu dikirim atau disimpan dalam format yang **ringan** dan **terstruktur**.
JSON (**JavaScript Object Notation**) adalah **format standar** untuk **menyimpan dan bertukar data** yang sangat mirip dengan cara kita menulis objek di JavaScript.

JSON digunakan di hampir semua aplikasi modern — dari komunikasi antara server dan browser, hingga penyimpanan konfigurasi aplikasi.

B. Bentuk JSON

Format dasar JSON terdiri dari:

- **Key** (kunci) dan **value** (nilai),
- Ditulis dalam tanda kurung kurawal {},
- Kunci **harus dalam tanda kutip dua** "".

Contoh format JSON:

```
/////// JSON ////////////////////////////////
{
 "nama": "Dian",
 "umur": 25,
 "kota": "Jakarta"
}
//////////////////////////////////////////////
```

Perhatikan:

- Nama properti "nama", "umur", "kota" wajib menggunakan tanda kutip ganda.
- Nilai bisa berupa string, angka, boolean, array, objek, atau null.

C. JSON vs JavaScript Object

	JavaScript Object	JSON
Kunci tanpa tanda kutip boleh	Wajib tanda kutip ganda	
Bisa menyimpan fungsi	Tidak bisa	
Khusus di JavaScript	Bisa di berbagai bahasa	

Contoh **JavaScript Object**:

```
/////// JSON ////////////////////////////////
let orang = {
 nama: "Dian",
 umur: 25
};
/////////////////////////////////////////////
```

Contoh **JSON**:

```
/////// JSON ////////////////////////////////
{
 "nama": "Dian",
 "umur": 25
}
/////////////////////////////////////////////
```

D. Mengubah Data JavaScript menjadi JSON

Gunakan JSON.stringify():

```
/////// JavaScript ////////////////////////////
let orang = {
 nama: "Dian",
 umur: 25
};

let jsonData = JSON.stringify(orang);
console.log(jsonData);
/////////////////////////////////////////////
```

Output:

```
/////// JSON ////////////////////////////////
{"nama":"Dian","umur":25}
/////////////////////////////////////////////////
```

E. Mengubah JSON menjadi JavaScript Object

Gunakan JSON.parse():

```
/////// JavaScript ////////////////////////////
let jsonData = '{"nama":"Dian","umur":25}';

let orang = JSON.parse(jsonData);
console.log(orang.nama); // Output: Dian
/////////////////////////////////////////////////
```

Studi Kasus: Simpan Data di Local Storage:

```
/////// JavaScript ////////////////////////////
let profil = {
  nama: "Dian",
  umur: 25,
  kota: "Jakarta"
};
// Simpan ke Local Storage
localStorage.setItem("profilPengguna",
JSON.stringify(profil));

// Ambil dari Local Storage
let ambilProfil =
JSON.parse(localStorage.getItem("profilPengguna"));
console.log(ambilProfil.kota); // Output: Jakarta
/////////////////////////////////////////////////
```

- **JavaScript Object** itu seperti catatan kecilmu sendiri bisa banyak gaya, santai.
- **JSON** itu seperti **formulir resmi** harus rapi, sesuai format, supaya orang lain bisa memahami tanpa bertanya lagi.

Sama seperti kamu mengirim daftar belanja, kamu harus memastikan formatnya jelas supaya temanmu tidak salah beli.

Kenapa JSON Sangat Penting?

- Ringan dan cepat
- Mudah dibaca manusia
- Dipahami oleh banyak bahasa pemrograman (Python, Java, PHP, C#, dll)
- Standar pertukaran data dalam API dan web service

JSON adalah bahasa universal dalam dunia pertukaran data modern.

Mengerti bagaimana mengubah JavaScript Object ke JSON, dan sebaliknya, adalah dasar untuk membangun aplikasi berbasis API, komunikasi server, dan data-driven application.

***

API dan REST API

A. Pengertian API

Bayangkan kamu pergi ke restoran. Kamu **tidak** langsung masuk ke dapur untuk mengambil makanan. Sebaliknya, kamu berbicara kepada **pelayan**. Pelayan ini mencatat pesananmu, menyampaikannya ke dapur, lalu membawa makanan yang sudah jadi ke mejamu.

Dalam dunia pemrograman, **API (Application Programming Interface)** berperan seperti **pelayan** tadi. API menghubungkan satu aplikasi dengan aplikasi lain, dengan cara yang aman dan terstruktur.

Secara sederhana:

API = Jembatan komunikasi antar aplikasi.

Kenapa API Penting?

- Membuat aplikasi lebih modular dan fleksibel
- Menghemat waktu pengembangan
- Memungkinkan aplikasi berinteraksi dengan sistem lain (seperti database, server, layanan cloud)

Contoh sehari-hari penggunaan API:

- Aplikasi ojek online mengambil data lokasi dari Google Maps API.
- Aplikasi cuaca mengambil data prakiraan dari API cuaca.
- Website e-commerce menggunakan API pembayaran dari Midtrans atau Stripe.

B. Rest API

REST adalah singkatan dari **Representational State Transfer**.

REST API adalah **jenis API** yang menggunakan aturan HTTP (GET, POST, PUT, DELETE) untuk berkomunikasi. REST sangat populer karena sederhana, ringan, dan mudah diintegrasikan.

Struktur Dasar REST API

Misalnya, kamu punya aplikasi toko online:

Method	URL	Fungsi
GET	/produk	Mengambil daftar produk
GET	/produk/1	Mengambil detail produk dengan ID 1
POST	/produk	Menambah produk baru
PUT	/produk/1	Mengedit produk ID 1
DELETE	/produk/1	Menghapus produk ID 1

C. Mengakses REST API dengan Fetch

```
////////////////////////////////////////////
// Ambil daftar produk
fetch('https://api.example.com/produk')
  .then(response => response.json())
  .then(data => console.log(data))
```

```
 .catch(error => console.log('Error:', error));
////////////////////////////////////////////////
```

- GET → Kamu **melihat** daftar menu.
- POST → Kamu **menambahkan** pesanan baru.
- PUT → Kamu **mengubah** pesanan (ganti topping pizza misalnya).
- DELETE → Kamu **membatalkan** pesanan.

Studi kasus dunia nyata

Kasus: Aplikasi Buku Digital

Kamu membuat aplikasi baca buku online. Data buku tidak kamu simpan di aplikasi langsung, melainkan kamu **mengambil data** dari REST API.

Langkah-langkah:

1. Aplikasi mengirim GET /buku ke API server.
2. API server mengembalikan daftar buku dalam format JSON.
3. Aplikasi menampilkan daftar buku tersebut ke pengguna.

Jika pengguna ingin menambahkan buku:

1. Aplikasi mengirim POST /buku dengan data buku baru.
2. API server menyimpan data tersebut ke database.

D. rinsip-Prinsip REST API

- **Stateless** → Setiap permintaan harus berisi semua informasi yang dibutuhkan.
- **Resource-based** → Data diakses lewat URL, bukan lewat prosedur.
- **HTTP Methods** → Gunakan GET, POST, PUT, DELETE dengan tepat.
- **Representasi data** → Biasanya berupa JSON.

***

Perjalanan kita dalam memahami JavaScript, dari dasar hingga ke konsep-konsep intermediate, merupakan fondasi penting dalam membangun aplikasi modern yang dinamis, interaktif, dan efisien.
Melalui buku ini, diharapkan pembaca tidak hanya memahami sintaks dan teori JavaScript, tetapi juga mampu mengembangkan logika pemrograman, menyusun struktur program yang baik, serta menerapkan konsep-konsep tersebut dalam proyek nyata.

Belajar pemrograman adalah perjalanan yang berkelanjutan. Teknologi terus berkembang, begitu juga dengan ekosistem JavaScript yang semakin luas — termasuk framework modern, teknik optimasi, dan integrasi dengan berbagai layanan web.
Oleh karena itu, pembaca diharapkan terus berlatih, mengeksplorasi teknologi baru, dan membangun berbagai proyek kecil hingga besar.

Ingat, kunci menjadi programmer yang andal bukan hanya memahami teori, tetapi juga konsisten berlatih dan belajar dari tantangan nyata.

Semoga buku ini menjadi salah satu bekal awal yang berharga dalam perjalanan panjang Anda di dunia pengembangan perangkat lunak.
Selamat berkarya dan terus semangat untuk menjadi bagian dari generasi pembangun masa depan digital!

Daftar Pustaka

1. Mozilla Developer Network (MDN). *JavaScript Documentation*. Diakses dari: https://developer.mozilla.org/en-US/docs/Web/JavaScript
2. Eloquent JavaScript. Marijn Haverbeke. (2018). *Eloquent JavaScript, 3rd Edition*. No Starch Press.
3. JavaScript.info. *The Modern JavaScript Tutorial*. Diakses dari: https://javascript.info/
4. W3Schools. *JavaScript Tutorial*. Diakses dari: https://www.w3schools.com/js/
5. FreeCodeCamp. *JavaScript Algorithms and Data Structures Certification*. Diakses dari: https://www.freecodecamp.org/learn/
6. Dicoding Indonesia. *Belajar Dasar Pemrograman JavaScript*. Diakses dari: https://www.dicoding.com/academies/256
7. Skilvul. *Belajar Pemrograman JavaScript untuk Pemula*. Diakses dari: https://skilvul.com/
8. OpenAI. *ChatGPT: Artificial Intelligence Language Model for Knowledge Assistance*. Diakses dari: https://chat.openai.com/

Ingin menguasai JavaScript dari nol sampai jadi Dewa?

Buku ini adalah teman setia yang bakal menemani perjalanan kamu mulai dari hal-hal dasar yang kadang bikin pusing, sampai trik-trik canggih yang bikin kamu merasa seperti penguasa kode. Ditulis dengan bahasa yang enak dibaca dan penuh analogi dunia nyata, buku ini cocok buat siapa saja: dari yang baru kenalan sama JavaScript, sampai yang ingin memperdalam skill dan jadi lebih percaya diri di dunia coding.

Di dalamnya kamu akan temukan:

- 👉 **Penjelasan konsep yang runut dan mudah dicerna**
- 👉 **Banyak analogi dunia nyata yang bikin konsep JavaScript jadi lebih ngeh**
- 👉 **Studi kasus ringan sampai yang lebih kompleks**
- 👉 **Tips & trik biar ngoding jadi lebih asik dan efisien**
- 👉 **Bonus: insight tentang error handling dan debugging yang sering disepelekan tapi krusial banget!**

Gak cuma ngajarin ngoding, buku ini bantu kamu nangkep logika di balik setiap baris kode, biar lebih paham, lebih siap, dan lebih pede.

Yuk, bersiaplah naik takhta jadi Dewa JavaScript!