



# Tutorial Lengkap Modern Javascript

PEMATERI : RANGGO PATO



# PENGENALAN JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Apa itu JavaScript?



Secara Definisi JavaScript adalah bahasa pemrograman untuk web yang digunakan untuk membuat halaman web menjadi interaktif.

Secara Fungsi JavaScript memiliki tugas utama.

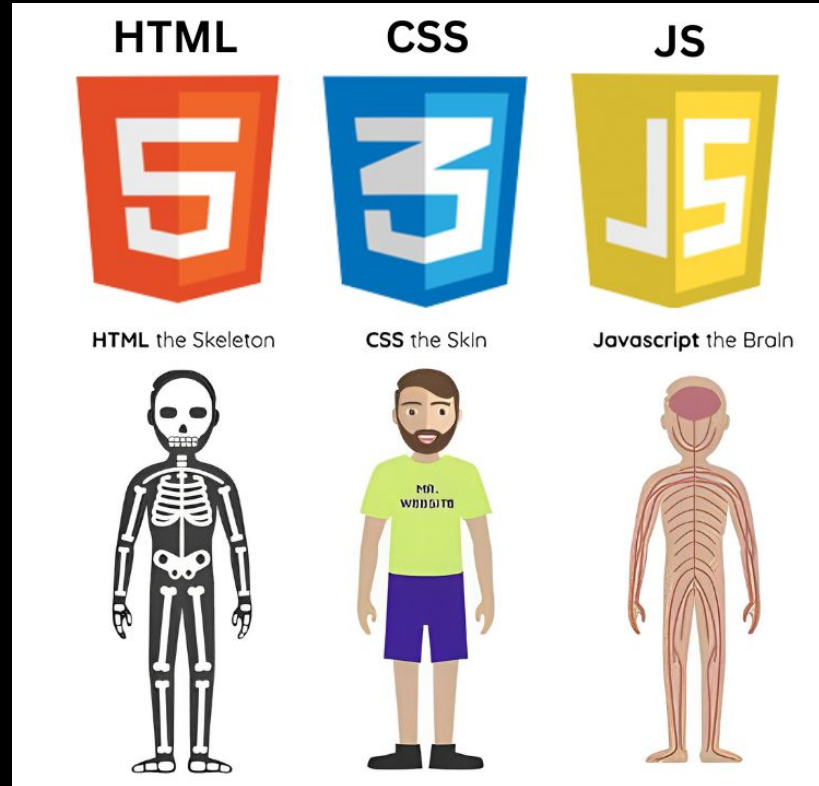
- Dinamis: Memungkinkan perubahan konten halaman tanpa memuat ulang.
- Interaktif: Digunakan untuk menangani event, animasi, dan validasi form.
- Versatil: Dapat digunakan di front-end dan back-end (Node.js).

# Sejarah dan Evolusi JavaScript



- Diciptakan oleh Brendan Eich pada tahun 1995.
- Awalnya disebut "Mocha," kemudian "LiveScript," dan akhirnya "JavaScript."
- 1996: JavaScript diadopsi oleh Netscape Navigator dan Internet Explorer.
- 1997: ECMAScript diciptakan sebagai standar JavaScript.
- Evolusi ECMAScript (ES5 pada 2009, ES6 pada 2015 dengan fitur baru seperti arrow functions, promises, dan lainnya).
- Munculnya framework dan library seperti React, Angular, dan Vue.js.
- Node.js memungkinkan JavaScript digunakan di sisi server.

# Penggambaran Sederhana Konsep JS



# Cara Kerja JavaScript di Browser



Setiap browser memiliki mesin JavaScript (contoh: V8 di Chrome, SpiderMonkey di Firefox).

Mesin ini yang mengeksekusi kode JavaScript dan berinteraksi dengan DOM.



# Langkah Kerja JavaScript:



- **Parsing:** Browser membaca HTML dan membentuk DOM (Document Object Model).
- **Eksekusi:** Mesin JavaScript mengeksekusi kode JavaScript sesuai urutan, berinteraksi dengan DOM untuk manipulasi elemen.
- **Event Loop:** JavaScript menangani event secara asinkron melalui event loop, memungkinkan interaksi dinamis tanpa memblokir eksekusi kode lainnya.

Contoh sederhananya : Bagaimana script sederhana di HTML dapat mengubah konten atau gaya elemen di halaman web.

# RANGKUMAN



JavaScript adalah bahasa pemrograman esensial untuk web.

Memiliki sejarah yang kaya dan terus berkembang.

Memungkinkan pengembangan halaman web interaktif dan dinamis.







# ENVIRONMENT SETUP

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**



# **HOMEWORK**

**SUPPORT CHANNEL INI KE :  
SAWERIA.CO/KENAPACODING**



# JAVASCRIPT DI BROWSER

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)



# VARIABEL DI JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Apa itu Variabel?



- Variabel adalah wadah yang digunakan untuk menyimpan data yang dapat berubah sepanjang program berjalan.
- Analogi: Variabel seperti kotak yang menyimpan nilai, dan kita bisa mengganti isi kotak tersebut kapan saja.
- Variabel di Javascript itu ada 3 yaitu, var, let, const





# TIPE DATA DI JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Tipe Data Primitif



- String, urutan karakter yang digunakan untuk menyimpan dan memanipulasi teks.
- Number digunakan untuk menyimpan nilai numerik, termasuk bilangan bulat (integer) dan desimal (float).
- Boolean hanya memiliki dua nilai: true atau false.
- Undefined, Nilai yang secara otomatis diberikan kepada variabel yang belum dideklarasikan atau belum diinisialisasi.
- Null, Nilai yang secara eksplisit diberikan untuk menunjukkan bahwa variabel tidak memiliki nilai
- Symbol (ES6+), Symbol adalah tipe data primitif yang unik dan immutable, digunakan sebagai identifier unik untuk properti objek.
- BigInt adalah tipe data yang digunakan untuk merepresentasikan angka yang sangat besar melebihi batas tipe data Number.

# Type Data Reference



- Object adalah tipe data kompleks yang digunakan untuk menyimpan koleksi data dalam pasangan key-value.
- Array adalah jenis Object yang menyimpan koleksi data berurutan dan dapat diakses melalui indeks.
- Function adalah blok kode yang dapat digunakan kembali, yang juga dianggap sebagai Object di JavaScript.



# Perbedaan Utama Tipe Primitive vs Reference



- **Penyimpanan:** Tipe data primitif disimpan di stack memory dengan nilai langsungnya, sedangkan reference types disimpan di heap memory dan variabel hanya menyimpan referensi ke data tersebut.
- **Perubahan Nilai:** Pada tipe data primitif, mengubah nilai variabel tidak mempengaruhi variabel lain yang menyimpan salinan nilai tersebut. Sedangkan pada reference types, mengubah data melalui satu variabel akan mempengaruhi variabel lain yang merujuk ke objek yang sama.
- **Kinerja:** Operasi pada tipe data primitif biasanya lebih cepat dan menggunakan lebih sedikit memori dibandingkan reference types karena penyimpanan dan akses langsung ke nilai.



# TYPE CONVERSION

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**

# Apa itu Type Conversion?



- Type Conversion adalah proses mengubah satu tipe data menjadi tipe data lain.
- Terdapat dua jenis konversi: **Implicit Conversion** (Coercion) dan **Explicit Conversion**.
- Implicit Conversion adalah konversi tipe data yang dilakukan secara otomatis oleh JavaScript.
- Terjadi ketika JavaScript mencoba menyesuaikan tipe data secara otomatis selama operasi tertentu.
- Explicit Conversion adalah konversi tipe data yang dilakukan secara eksplisit oleh programmer menggunakan metode atau fungsi tertentu.



# OPERATOR DI JAVASCRIPT

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**

# Apa itu Operator



- Operator adalah simbol atau kata kunci yang digunakan untuk melakukan operasi pada satu atau lebih operand (nilai atau variabel).
- Contoh umum termasuk penjumlahan, pengurangan, dan penggabungan string.

# Jenis-Jenis Operator di JavaScript



- Operator Aritmatika
- Operator Assignment
- Operator Perbandingan
- Operator Logika
- Operator Ternary

# Operator Aritmatika



Digunakan untuk melakukan operasi matematika.

- + (Penjumlahan)
- - (Pengurangan)
- \* (Perkalian)
- / (Pembagian)
- % (Modulus: Sisa bagi)
- \*\* (Eksponen: Pangkat)
- ++ (Increment: Menambah satu nilai)
- -- (Decrement: Mengurangi satu nilai)

# Operator Assignment



Digunakan untuk menetapkan nilai ke variabel.

- = (Assignment dasar)
- += (Penjumlahan dan assignment)
- -= (Pengurangan dan assignment)
- \*= (Perkalian dan assignment)
- /= (Pembagian dan assignment)
- %= (Modulus dan assignment)



# Operator Perbandingan [ Comparison ]



Digunakan untuk membandingkan dua nilai.

Menghasilkan nilai boolean (`true` atau `false`).

- `==` (Sama dengan, mengecek nilai)
- `===` (Sama dengan secara ketat, mengecek nilai dan tipe data)
- `!=` (Tidak sama dengan)
- `!==` (Tidak sama dengan secara ketat)
- `>` (Lebih besar dari)
- `<` (Lebih kecil dari)
- `>=` (Lebih besar atau sama dengan)
- `<=` (Lebih kecil atau sama dengan)

# Operator Logika



Digunakan untuk menggabungkan ekspresi logika dan menghasilkan nilai boolean.

- && (AND: Menghasilkan true jika kedua operand true)
- || (OR: Menghasilkan true jika salah satu operand true)
- ! (NOT: Membalikkan nilai boolean operand)

# Operator Ternary



- Operator ternary adalah operator kondisional yang merupakan cara singkat untuk menulis if-else.
- `condition ? expressionIfTrue : expressionIfFalse;`

javascript

```
let age = 18;  
let canVote = (age >= 18) ? "Yes" : "No";  
console.log(canVote); // Yes
```



# STRINGS DI JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Apa itu String?



- String adalah tipe data primitif di JavaScript yang digunakan untuk merepresentasikan teks.
- String diapit oleh tanda kutip tunggal ('...'), tanda kutip ganda ("..."), atau backticks (`...`).

javascript

```
let singleQuote = 'Hello';  
let doubleQuote = "World";  
let templateLiteral = `Hello, World!`;
```

# Mengakses Karakter dalam String



- Setiap karakter dalam string memiliki indeks, dimulai dari 0.
- Karakter dapat diakses menggunakan notasi bracket ([]).

javascript

```
let str = "JavaScript";  
console.log(str[0]); // Output: "J"  
console.log(str[4]); // Output: "s"
```

# Property dan Metode di String



- **length**, mengetahui panjang string

javascript

```
let txt = "Hello";  
console.log(txt.length); // Output: 5
```

- **toLowerCase()**, menjadikan huruf kecil

javascript

```
let mixedCase = "HeLlO";  
let lowerCase = mixedCase.toLowerCase();  
console.log(lowerCase); // Output: "hello"
```

- **toUpperCase()**: menjadikan huruf besar.
- **trim**, menghapus spasi di dalam string

javascript

```
let lowerCase = "hello";  
let upperCase = lowerCase.toUpperCase();  
console.log(upperCase); // Output: "HELLO"
```

javascript

```
let spaced = "  JavaScript  ";  
let trimmed = spaced.trim();  
console.log(trimmed); // Output: "JavaScript"
```

# Manipulasi String



- Menggabungkan string (concat)

javascript

```
let firstName = "John";
let lastName = "Doe";
let fullName = firstName + " " + lastName;
console.log(fullName); // Output: "John Doe"
```

javascript

```
let firstName = "John";
let lastName = "Doe";
let fullName = `${firstName} ${lastName}`;
console.log(fullName); // Output: "John Doe"
```

- Mengambil bagian dari string berdasarkan indeks (slice)

javascript

```
let text = "JavaScript";
let part = text.slice(0, 4);
console.log(part); // Output: "Java"
```

javascript

```
let text = "JavaScript";
let part = text.substring(4, 10);
console.log(part); // Output: "Script"
```

- Mengganti bagian dari string dengan string baru.

javascript

```
let str = "Hello, World!";
let newStr = str.replace("World", "JavaScript");
console.log(newStr); // Output: "Hello, JavaScript!"
```



# Manipulasi String (2)



- Membagi sebuah string menjadi array berdasarkan separator yang diberikan (split)

```
javascript
```

```
string.split(separator, limit);
```

- Menggabungkan semua elemen dalam array menjadi sebuah string (join)

```
javascript
```

```
array.join(separator);
```

# Pencarian dalam string



- **indexOf():** Mengembalikan indeks dari kemunculan pertama substring dalam string, atau -1 jika tidak ditemukan.

javascript

```
let sentence = "The quick brown fox";  
let index = sentence.indexOf("quick");  
console.log(index); // Output: 4
```

- **lastIndexOf():** Mengembalikan indeks dari kemunculan terakhir substring dalam string.

javascript

```
let sentence = "The quick brown fox jumps over the lazy dog";  
let index = sentence.lastIndexOf("the");  
console.log(index); // Output: 31
```

- **includes():** Mengembalikan true jika string mengandung substring yang ditentukan, sebaliknya false.

javascript

```
let sentence = "The quick brown fox";  
let exists = sentence.includes("brown");  
console.log(exists); // Output: true
```

# Capitalize Challenge



- Pastikan huruf pertama dari variable `stringsAsli` menjadi huruf kapital, sementara huruf-huruf lainnya tetap dalam huruf kecil. Dan simpan hasilnya ke variable

Hasil yang Diharapkan:

- ``const stringsAsli = 'javascript';``
- ``console.log(stringsBaru); // 'Javascript'``



# NUMBER DI JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)



# MATH OBJECT JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)



# DATE OBJECT JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)



# ARRAY DI JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Apa itu Array



- Array adalah struktur data yang digunakan untuk menyimpan koleksi elemen, seperti angka atau string dalam satu variabel. Di JavaScript, array dapat menyimpan berbagai tipe data sekaligus dan memiliki indeks yang dimulai dari 0.





# Membuat Array di Javascript



Ada beberapa cara untuk membuat array di JavaScript:

- Menggunakan notasi array literal

javascript

```
let fruits = ["Apple", "Banana", "Mango"];
```

- Menggunakan new Array()

javascript

```
let numbers = new Array(1, 2, 3, 4, 5);
```

# Mengakses Elemen Array



Kita bisa mengakses elemen dalam array menggunakan indeks. Ingat, indeks array dimulai dari 0

javascript

```
let fruits = ["Apple", "Banana", "Mango"];
```

```
console.log(fruits[0]); // Apple
```

```
console.log(fruits[2]); // Mango
```

# Menambah atau Mengubah Elemen Array



Kita dapat menambah elemen baru atau mengubah elemen yang sudah ada di array

javascript

```
let fruits = ["Apple", "Banana"];  
fruits[2] = "Mango"; // Menambah elemen baru  
fruits[0] = "Orange"; // Mengubah elemen yang ada  
  
console.log(fruits); // ["Orange", "Banana", "Mango"]
```

# Manipulasi Array



JavaScript menyediakan berbagai metode bawaan untuk memanipulasi array:

- **push()**: Menambah elemen ke akhir array.
- **pop()**: Menghapus elemen terakhir array.
- **shift()**: Menghapus elemen pertama array.
- **unshift()**: Menambah elemen ke awal array.
- **length**: Mendapatkan panjang (jumlah elemen) array.

javascript

```
let numbers = [1, 2, 3];

numbers.push(4); // Menambah elemen di akhir: [1, 2, 3, 4]
numbers.pop(); // Menghapus elemen terakhir: [1, 2, 3]
numbers.shift(); // Menghapus elemen pertama: [2, 3]
numbers.unshift(0); // Menambah elemen di awal: [0, 2, 3]


console.log(numbers.length); // 3
```

# Manipulasi Array (2)



- **concat()**: Menggabungkan dua atau lebih array.
- **slice()**: Mengambil bagian tertentu dari array.
- **includes()**: Mengecek apakah suatu elemen ada di dalam array.
- **splice()**: Menambah atau menghapus elemen dari array.
- **indexOf()**: Mencari indeks dari elemen tertentu.

javascript

 Copy code

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

let newNumbers = numbers.concat([6, 7]); // Menggabungkan array: [1, 2, 3, 4, 5, 6, 7]
let slicedNumbers = numbers.slice(1, 3); // Mengambil elemen dari indeks 1 hingga 2: [2, 3]
numbers.splice(2, 1, 10); // Menghapus elemen di indeks 2 dan menggantinya dengan 10: [1, 2, 10, 4, 5]

console.log(numbers.indexOf(4)); // 3 (indeks elemen 4)
console.log(numbers.includes(10)); // true
```

# Multidimensional Array (Array of Arrays)



Kita juga bisa membuat array dengan lebih dari satu dimensi, misalnya array dua dimensi:

javascript

```
let matrix = [  
  [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9]  
];
```

```
console.log(matrix[1][2]); // Mengakses elemen pada baris 2, kolom 3: 6
```



# OBJECT DI JAVASCRIPT

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**

# Pengertian Object



- Object di JavaScript adalah tipe data yang digunakan untuk menyimpan koleksi data dan entitas yang lebih kompleks. Sebuah object adalah pasangan antara *key* dan *value* (disebut juga properti).

javascript

```
let mahasiswa = {  
  nama: "Budi",  
  umur: 21,  
  jurusan: "Teknik Informatika"  
};
```

- Pada contoh di atas, mahasiswa adalah sebuah object yang memiliki tiga properti: nama, umur, dan jurusan.



# Membuat Object



Object bisa dibuat dengan dua cara utama:

- **Object Literal**, Cara yang paling umum dan sederhana :

javascript

```
let mobil = {  
  merk: "Toyota",  
  model: "Avanza",  
  tahun: 2021  
};
```

- Menggunakan **Constructor** `new Object()` :

javascript

```
let buku = new Object();  
buku.judul = "Belajar JavaScript";  
buku.penulis = "John Doe";  
buku.tahun = 2023;
```

# Mengakses Properti Object



Ada dua cara untuk mengakses properti dalam object:

- **Menggunakan Notasi Titik (Dot Notation):**

```
javascript
```

```
console.log(mahasiswa.nama); // Output: Budi
```

- **Menggunakan Notasi Kurung Petak (Bracket Notation):**

```
javascript
```

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

**Note :** Notasi kurung petak sangat berguna jika nama properti mengandung spasi atau karakter khusus.

# Menambah dan Mengubah Properti



- **Menambah Properti:**

```
javascript
```

```
mahasiswa.alamat = "Jakarta";
```

- **Mengubah Properti:**

```
javascript
```

```
mahasiswa.umur = 22;
```

# Menghapus Properti



- Untuk menghapus properti dari object, gunakan operator `delete`.

javascript

```
delete mahasiswa.alamat;  
console.log(mahasiswa.alamat); // Output: undefined
```

# Destructuring Object



Destructuring adalah fitur ES6 yang memungkinkan Anda untuk mengambil nilai dari object dan menempatkannya dalam variabel.

javascript

```
let { nama, umur } = mahasiswa;  
console.log(nama); // Output: Budi  
console.log(umur); // Output: 22
```

# Menghapus Properti



- Untuk menghapus properti dari object, gunakan operator `delete`.

javascript

```
delete mahasiswa.alamat;  
console.log(mahasiswa.alamat); // Output: undefined
```

# Nested Object



Object bisa berisi object lain sebagai properti. Ini disebut *nested object*.

javascript

```
let universitas = {  
  nama: "Universitas ABC",  
  fakultas: {  
    nama: "Fakultas Teknik",  
    jurusan: "Teknik Informatika"  
  }  
};  
  
console.log(universitas.fakultas.nama); // Output: Fakultas Teknik
```



# IF ELSE STATEMENT

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**



# Pengertian If-Else Statement



- `if-else` statement adalah salah satu struktur kontrol yang paling dasar di JavaScript. Ini digunakan untuk membuat keputusan berdasarkan kondisi tertentu. Jika kondisi tersebut benar (`true`), blok kode tertentu akan dijalankan. Jika salah (`false`), kode lain dapat dijalankan.

javascript

```
if (kondisi) {  
    // Kode yang dijalankan jika kondisi benar (true)  
} else {  
    // Kode yang dijalankan jika kondisi salah (false)  
}
```

javascript

```
let age = 18;  
  
if (age >= 18) {  
    console.log("You are eligible to vote.");  
} else {  
    console.log("You are not eligible to vote.");  
}
```

# If- else If statement



- if-else if digunakan untuk memeriksa beberapa kondisi secara berurutan. Blok kode pertama yang kondisinya benar (true) akan dijalankan, dan eksekusi akan berhenti setelah itu.

javascript

```
if (kondisi1) {  
    // Kode jika kondisi1 benar (true)  
} else if (kondisi2) {  
    // Kode jika kondisi2 benar (true)  
} else {  
    // Kode jika semua kondisi salah (false)  
}
```

javascript

```
let score = 75;  
  
if (score >= 90) {  
    console.log("Grade: A");  
} else if (score >= 80) {  
    console.log("Grade: B");  
} else if (score >= 70) {  
    console.log("Grade: C");  
} else if (score >= 60) {  
    console.log("Grade: D");  
} else {  
    console.log("Grade: F");  
}
```

# Nested If-Else (If-Else Bertingkat)



javascript

```
if (kondisi1) {  
  if (kondisi2) {  
    // Kode jika kondisi1 dan kondisi2 benar  
  } else {  
    // Kode jika kondisi1 benar tapi kondisi2 salah  
  }  
} else {  
  // Kode jika kondisi1 salah  
}
```

```
let num = 10;  
  
if (num > 0) {  
  if (num % 2 === 0) {  
    console.log("The number is positive and even.");  
  } else {  
    console.log("The number is positive but odd.");  
  }  
} else {  
  console.log("The number is non-positive.");  
}
```



# SWITCH CASE DI JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Pengertian switch-case



- `switch-case` adalah salah satu struktur kontrol di JavaScript yang digunakan untuk membuat keputusan berdasarkan nilai yang cocok dengan berbagai kasus. Ini sering digunakan saat ada banyak kemungkinan hasil dan Anda ingin menghindari menggunakan terlalu banyak `if-else`.



# Sintaks Dasar



- **ekspresi:** Biasanya berupa variabel yang nilainya akan dicek terhadap berbagai kasus.
- **case:** Setiap case merepresentasikan nilai yang akan dibandingkan dengan ekspresi.
- **break:** Digunakan untuk menghentikan eksekusi setelah satu kasus cocok. Jika tidak ada break, kode akan terus berjalan ke kasus berikutnya (fall-through).
- **default:** Opsional, digunakan jika tidak ada kasus yang cocok. Biasanya diletakkan di akhir

javascript

```
switch (ekspresi) {  
  case nilai1:  
    // kode yang dijalankan jika ekspresi sama dengan nilai1  
    break;  
  case nilai2:  
    // kode yang dijalankan jika ekspresi sama dengan nilai2  
    break;  
  default:  
    // kode yang dijalankan jika tidak ada nilai yang cocok  
}
```

# Contoh Penggunaan Sederhana



```
let hari = 3;
let namaHari;

switch (hari) {
  case 1:
    namaHari = "Senin";
    break;
  case 2:
    namaHari = "Selasa";
    break;
  case 3:
    namaHari = "Rabu";
    break;
  case 4:
    namaHari = "Kamis";
    break;
```

```
    case 5:
      namaHari = "Jumat";
      break;
    case 6:
      namaHari = "Sabtu";
      break;
    case 7:
      namaHari = "Minggu";
      break;
    default:
      namaHari = "Hari tidak valid";
  }

  console.log(namaHari); // Output: Rabu
```

# Penggunaan break



break sangat penting untuk menghentikan eksekusi setelah satu kasus cocok. Jika break tidak dituliskan, JavaScript akan terus mengeksekusi kode di kasus berikutnya (fall-through).

javascript

```
let warna = "biru";

switch (warna) {
  case "merah":
    console.log("Warna adalah merah");
  case "biru":
    console.log("Warna adalah biru");
  case "hijau":
    console.log("Warna adalah hijau");
  default:
    console.log("Warna tidak dikenali");
}
```

Warna adalah biru  
Warna adalah hijau  
Warna tidak dikenali

Note : Tanpa break, semua kasus setelah case "biru"



# Contoh Menggunakan `default`



default digunakan jika tidak ada kasus yang cocok dengan ekspresi.

javascript

```
let angka = 10;

switch (angka) {
  case 1:
    console.log("Angka adalah 1");
    break;
  case 5:
    console.log("Angka adalah 5");
    break;
  default:
    console.log("Angka tidak dikenali");
}
```

Angka tidak dikenali

# Switch-Case dengan Ekspresi



Switch-case juga bisa digunakan dengan ekspresi atau operasi.

javascript

```
let nilai = 85;

switch (true) {
  case nilai >= 90:
    console.log("Grade: A");
    break;
  case nilai >= 80:
    console.log("Grade: B");
    break;
  case nilai >= 70:
    console.log("Grade: C");
    break;
  default:
    console.log("Grade: D");
}
```

# Kelebihan dan Kekurangan Switch-Case



## Kelebihan:

- Lebih mudah dibaca dibandingkan dengan rantai `if-else` panjang.
- Cocok digunakan saat ada banyak kemungkinan nilai yang harus diperiksa.

## Kekurangan:

- Kurang fleksibel dibandingkan `if-else` saat membandingkan dengan kondisi kompleks.
- Rentan terhadap kesalahan jika `break` terlewat.



# LOOP DI JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Apa itu Loop



- Loop digunakan untuk menjalankan blok kode berulang kali selama kondisi tertentu terpenuhi.
- JavaScript mendukung beberapa jenis loop: **for**, **while**, dan **do-while**. Setiap jenis loop memiliki kegunaannya sendiri, tergantung pada kebutuhan.



# For Loop



for loop digunakan ketika jumlah iterasi sudah diketahui.

sintaks :

javascript

```
for (inisialisasi; kondisi; perubahan) {  
  // kode yang dijalankan selama kondisi benar  
}
```

contoh penggunaan for loop

javascript

```
for (let i = 1; i <= 5; i++) {  
  console.log(`Iterasi ke-${i}`);  
}
```

**Inisialisasi:** Nilai awal variabel loop.

**Kondisi:** Kondisi yang harus benar agar loop terus berjalan.

**Perubahan:** Perubahan yang terjadi pada variabel loop setiap iterasi.

# While Loop



while loop menjalankan kode selama kondisi tetap benar. Loop ini berguna ketika jumlah iterasi tidak pasti dan bergantung pada kondisi tertentu. sintaks :

javascript

```
while (kondisi) {  
    // kode yang dijalankan selama kondisi benar  
}
```

contoh penggunaan while loop

javascript

```
let i = 1;  
  
while (i <= 5) {  
    console.log(`Iterasi ke-${i}`);  
    i++;  
}
```

# Do While Loop



do-while loop selalu menjalankan kode setidaknya satu kali, karena kondisi diperiksa setelah blok kode dieksekusi. sintaks :

javascript

```
do {  
  // kode yang dijalankan  
} while (kondisi);
```

contoh penggunaan do while loop

javascript

```
let i = 1;  
  
do {  
  console.log(`Iterasi ke-${i}`);  
  i++;  
} while (i <= 5);
```



# For-In dan For-Of Loop



For-In Loop, Digunakan untuk mengiterasi properti objek atau indeks array

```
javascript

const objek = { nama: "John", umur: 30 };

for (let properti in objek) {
  console.log(`${properti}: ${objek[properti]}`);
}
```

For-Of Loop, Digunakan untuk mengiterasi elemen dalam iterable seperti array atau string.

```
javascript

const array = [1, 2, 3, 4, 5];

for (let nilai of array) {
  console.log(nilai);
}
```

# Kapan Menggunakan Jenis Loop yang Berbeda?



- **For Loop:** Ketika jumlah iterasi sudah diketahui.
- **While Loop:** Ketika iterasi bergantung pada kondisi yang mungkin tidak diketahui sebelumnya.
- **Do-While Loop:** Ketika Anda perlu menjalankan kode setidaknya satu kali sebelum mengecek kondisi.



# **FUNCTION DI JAVASCRIPT**

**SUPPORT CHANNEL INI KE :  
SAWERIA.CO/KENAPACODING**

# Pengertian Function



**Function** di JavaScript adalah blok kode yang dirancang untuk melakukan tugas tertentu dan dapat digunakan berulang kali. Function mempermudah pengorganisasian kode dengan membagi tugas besar menjadi tugas-tugas kecil yang lebih terstruktur.



# Membuat Function



Untuk membuat function, gunakan kata kunci function diikuti dengan nama function, parameter (jika ada), dan blok kode yang akan dijalankan.

javascript

```
function namaFunction(parameter1, parameter2, ...) {  
  // Blok kode yang akan dijalankan  
}
```

SINTAKS

javascript

```
function sapaPengguna(nama) {  
  console.log("Halo, " + nama + "!");  
}
```

CONTOH

# Memanggil Function



Untuk menjalankan function, cukup panggil dengan menyebutkan nama function diikuti dengan tanda kurung, serta masukkan nilai untuk parameternya (jika ada).

javascript

```
sapaPengguna("Budi"); // Output: Halo, Budi!
```

# Parameter dan Argumen



- **Parameter** adalah variabel yang didefinisikan dalam deklarasi function.
- **Argumen** adalah nilai yang diberikan saat function dipanggil.

javascript

```
function tambah(a, b) {  
  return a + b;  
}
```

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

Di sini, a dan b adalah parameter, sedangkan 5 dan 3 adalah argumen.

# Return Statement



Function dapat mengembalikan nilai menggunakan return. Setelah return dieksekusi, eksekusi function akan berhenti.

```
function kuadrat(x) {  
  return x * x;  
}  
  
console.log(kuadrat(4)); // Output: 16
```



# Function dengan Parameter Default



Parameter default adalah nilai yang akan digunakan jika tidak ada argumen yang diberikan saat function dipanggil.

javascript

```
function perkenalan(nama = "Anonim") {  
  console.log("Halo, " + nama + "!");  
}
```

```
perkenalan(); // Output: Halo, Anonim!
```



# **FUNCTION EXPRESSIONS**

**SUPPORT CHANNEL INI KE :  
SAWERIA.CO/KENAPACODING**

# Function Expressions



Selain membuat function menggunakan deklarasi (function declaration), Anda juga bisa membuat function menggunakan ekspresi (function expression), di mana function disimpan dalam variabel.

javascript

```
let kali = function(a, b) {  
  return a * b;  
};
```

```
console.log(kali(4, 5)); // Output: 20
```



# ARROW FUNCTIONS

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**

# Arrow Functions



Arrow function adalah cara singkat untuk menulis function expression, diperkenalkan di ES6.

javascript

```
let kali = function(a, b) {  
  return a * b;  
};  
  
console.log(kali(4, 5)); // Output: 20
```

javascript

```
let kali = (a, b) => {  
  return a * b;  
};  
  
console.log(kali(4, 5)); // Output: 20
```

Note : Jika function hanya memiliki satu pernyataan, Anda bisa menghilangkan tanda kurung kurawal {} dan kata kunci return.

javascript

```
let kali = (a, b) => a * b;  
  
console.log(kali(4, 5)); // Output: 20
```



# IFFE

**SUPPORT CHANNEL INI KE :  
SAWERIA.CO/KENAPACODING**

# IIFE (Immediately Invoked Function Expression)



IIFE adalah function yang dipanggil langsung setelah dibuat. Ini berguna untuk mengisolasi variabel dan mencegahnya mengganggu kode lain.

javascript

```
(function() {  
    console.log("IIFE dipanggil!");  
})(); // Output: IIFE dipanggil!
```

javascript

```
const appConfig = (function() {  
    const apiKey = "12345";  
    const apiUrl = "https://api.example.com";  
  
    return {  
        getApiKey: function() {  
            return apiKey;  
        },  
        getApiUrl: function() {  
            return apiUrl;  
        }  
    };  
})();  
  
console.log(appConfig.getApiKey()); // Output: 12345  
console.log(appConfig.getApiUrl()); // Output: https://api.example.com
```



# HIGH ORDER DAN CALLBACK FUNCTION

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)



# Higher-Order Functions



Higher-order functions adalah function yang menerima function lain sebagai argumen atau mengembalikan function lain sebagai hasil.

javascript

```
function manipulasiArray(arr, callback) {  
  let hasil = [];  
  for (let i = 0; i < arr.length; i++) {  
    hasil.push(callback(arr[i]));  
  }  
  return hasil;  
}  
  
function kaliDua(x) {  
  return x * 2;  
}  
  
let angka = [1, 2, 3, 4];  
let hasil = manipulasiArray(angka, kaliDua);  
  
console.log(hasil); // Output: [2, 4, 6, 8]
```

# Callback Functions



Callback function adalah function yang dikirim sebagai argumen ke function lain dan dipanggil di dalam function tersebut.

javascript

```
function selesaikanTugas(tugas, callback) {  
  console.log("Menyelesaikan tugas: " + tugas);  
  callback();  
}  
  
function tugasSelesai() {  
  console.log("Tugas selesai!");  
}  
  
selesaikanTugas("Belajar JavaScript", tugasSelesai);
```



# SPREAD OPERATOR

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Apa Itu Spread Operator?



- Spread Operator di JavaScript dilambangkan dengan tiga titik (...). Spread digunakan untuk “menyebarkan” elemen dari sebuah array atau objek ke dalam elemen-elemen individual. Ini sangat berguna untuk menggabungkan, menyalin, atau memisahkan data.



# Spread Operator pada Array



Menggabungkan Array:

javascript

```
let fruits = ["Apple", "Banana"];  
let moreFruits = ["Mango", "Orange"];  
let allFruits = [...fruits, ...moreFruits];  
  
console.log(allFruits); // ["Apple", "Banana", "Mango", "Orange"]
```

Menyalin Array:

javascript

```
let originalArray = [1, 2, 3];  
let copiedArray = [...originalArray];  
  
console.log(copiedArray); // [1, 2, 3]
```

**Note:** Spread Operator membuat salinan "shallow copy", sehingga perubahan pada objek dalam array yang disalin tetap dapat memengaruhi array aslinya.

# Spread Operator pada Objek



## Menggabungkan Objek:

javascript

```
let person = { name: "John", age: 30 };  
let job = { title: "Developer", company: "Tech Corp" };  
  
let fullProfile = { ...person, ...job };  
  
console.log(fullProfile); // { name: "John", age: 30, title:
```

## Menyalin Objek:

javascript

```
let originalObject = { a: 1, b: 2 };  
let copiedObject = { ...originalObject };  
  
console.log(copiedObject); // { a: 1, b: 2 }
```

# Spread Operator dengan Fungsi



Spread Operator dapat digunakan untuk memisahkan elemen array menjadi argumen individu dalam sebuah fungsi.

javascript

```
function sum(x, y, z) {  
  return x + y + z;  
}  
  
let numbers = [1, 2, 3];  
console.log(sum(...numbers)); // 6
```

# Spread Operator pada String



Spread Operator dapat digunakan untuk memisahkan karakter dalam string menjadi elemen individual.

javascript

```
let str = "Hello";  
let chars = [...str];  
  
console.log(chars); // ["H", "e", "l", "l", "o"]
```



# Kombinasi Spread Operator dengan Array dan Objek



Spread Operator memungkinkan penggabungan fleksibel antara array atau objek:

javascript

```
let array = [1, 2, ...[3, 4], 5];  
console.log(array); // [1, 2, 3, 4, 5]  
  
let obj = { ...{ a: 1, b: 2 }, c: 3 };  
console.log(obj); // { a: 1, b: 2, c: 3 }
```



# RECURSION DI JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Pengertian Recursion



Recursion terjadi ketika sebuah function memanggil dirinya sendiri secara langsung atau tidak langsung untuk menyelesaikan sebagian dari masalah yang diberikan. Setiap panggilan recursion membawa masalah tersebut lebih dekat ke kondisi dasar (base case), yaitu kondisi di mana recursion berhenti.



# Struktur Recursion



Recursion biasanya terdiri dari dua bagian penting:

1. **Base Case (Kondisi Dasar):** Bagian dari function yang menghentikan recursion. Ini adalah kondisi di mana masalah tidak lagi memerlukan pemanggilan function secara rekursif.
2. **Recursive Case:** Bagian dari function yang memecah masalah menjadi sub-masalah yang lebih kecil dan memanggil dirinya sendiri.

# Contoh Recursion: Faktorial



Faktorial dari sebuah bilangan adalah hasil kali dari bilangan tersebut dengan semua bilangan bulat positif di bawahnya. Faktorial dari  $n$  ditulis sebagai  $n!$ . Sebagai contoh,  $5!$  (dibaca "5 faktorial") adalah  $5 * 4 * 3 * 2 * 1$ , yang sama dengan 120.

javascript

```
function faktorial(n) {  
  // Base case  
  if (n === 0) {  
    return 1;  
  }  
  // Recursive case  
  return n * faktorial(n - 1);  
}  
  
console.log(faktorial(5)); // Output: 120
```

## Penjelasan:

- **Base Case:** Jika  $n$  sama dengan 0, function mengembalikan 1. Ini adalah kondisi dasar yang menghentikan recursion.
- **Recursive Case:** Jika  $n$  tidak sama dengan 0, function mengalikan  $n$  dengan hasil pemanggilan dirinya sendiri ( $\text{faktorial}(n - 1)$ ). Dengan setiap pemanggilan, nilai  $n$  berkurang 1 hingga mencapai 0.

# Keuntungan dan Kekurangan Recursion



## Keuntungan:

- Recursion dapat menyederhanakan penyelesaian masalah yang kompleks dengan membaginya menjadi masalah yang lebih kecil.
- Beberapa masalah yang bersifat rekursif secara alami, seperti traversal pohon atau algoritma pembagian, lebih mudah dipecahkan dengan recursion.

## Kekurangan:

- Recursion dapat menyebabkan penggunaan memori yang lebih besar karena setiap panggilan recursive menambah frame baru ke stack call.
- Jika base case tidak tercapai atau salah diimplementasikan, function bisa masuk ke dalam loop infinite dan menyebabkan stack overflow.

Dengan memahami konsep recursion, Anda dapat memecahkan berbagai masalah komputasi yang berulang dengan cara yang lebih elegan dan efisien.



# SCOPE DAN HOISTING

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**

# Pengertian Scope



Scope mengacu pada aksesibilitas variabel dan fungsi dalam berbagai bagian kode Anda.

JavaScript memiliki dua jenis scope utama:

- **Global Scope**
- **Local Scope**



# Global Scope



Variabel yang dideklarasikan di luar fungsi atau block apa pun memiliki **global scope**. Variabel ini dapat diakses dari mana saja dalam kode Anda, baik di dalam fungsi maupun di luar fungsi.

javascript

```
let globalVar = "Saya global";

function contohGlobal() {
  console.log(globalVar); // Akses variabel global
}

contohGlobal(); // Output: Saya global
console.log(globalVar); // Output: Saya global
```

Note : Dalam contoh ini, `globalVar` dapat diakses baik di dalam fungsi `contohGlobal` maupun di luar fungsi tersebut.

# Lokal Scope



- Variabel yang dideklarasikan di dalam fungsi atau block hanya dapat diakses di dalam fungsi atau block tersebut. Variabel ini memiliki **local scope**.
- Ada dua jenis local scope dalam JavaScript:
  1. **Function Scope**: Variabel yang dideklarasikan dengan var, let, atau const di dalam fungsi.
  2. **Block Scope**: Variabel yang dideklarasikan dengan let atau const di dalam block ({}).

# Lokal Scope (2)



## Contoh Function Scope:

javascript

```
function contohLocal() {  
  let localVar = "Saya lokal";  
  console.log(localVar); // Output: Saya lokal  
}  
  
contohLocal();  
console.log(localVar); // Error: localVar is not defined
```

## Contoh Block Scope:

javascript

```
if (true) {  
  let blockVar = "Saya block-scoped";  
  console.log(blockVar); // Output: Saya block-scoped  
}  
  
console.log(blockVar); // Error: blockVar is not defined
```

# Hoisting



Hoisting adalah mekanisme di mana deklarasi variabel dan fungsi "diangkat" (hoisted) ke bagian atas scope-nya sebelum kode dieksekusi. Ini berarti Anda dapat menggunakan variabel dan fungsi sebelum dideklarasikan dalam kode, meskipun nilai variabel tidak diangkat.



# Hoisting pada Variabel



JavaScript hanya mengangkat **deklarasi** variabel, bukan inisialisasinya. Jadi, variabel yang diakses sebelum dideklarasikan akan menghasilkan undefined jika menggunakan var, dan error jika menggunakan let atau const.

Contoh hoisting pada var

javascript

```
console.log(hoistedVar); // Output: undefined
var hoistedVar = "Saya terhoist";
console.log(hoistedVar); // Output: Saya terhoist
```

Contoh hoisting pada let dan const

javascript

```
console.log(hoistedLet); // Error: Cannot access 'hoistedLet' before declaration
let hoistedLet = "Saya tidak terhoist";

console.log(hoistedConst); // Error: Cannot access 'hoistedConst' before declaration
const hoistedConst = "Saya tidak terhoist";
```

# Hoisting pada Fungsi



Fungsi yang dideklarasikan dengan cara **function declaration** akan diangkat sepenuhnya (baik deklarasi maupun definisi), sehingga dapat dipanggil sebelum dideklarasikan.

javascript

```
console.log(fungsiTerhoist()); // Output: Saya fungsi yang terhoist

function fungsiTerhoist() {
    return "Saya fungsi yang terhoist";
}
```

Namun, jika menggunakan **function expression** (termasuk dengan let, const, atau var), hanya variabel yang diangkat, bukan fungsinya.

javascript

```
console.log(fungsiTidakTerhoist); // Output: undefined
var fungsiTidakTerhoist = function() {
    return "Saya fungsi yang tidak terhoist";
};

console.log(fungsiTidakTerhoist()); // Output: Saya fungsi y
```

# Kesimpulan



- **Scope** menentukan di mana variabel dan fungsi dapat diakses dalam kode.
- **Global Scope** variabel dapat diakses dari mana saja, sedangkan **Local Scope** variabel hanya dapat diakses di dalam fungsi atau block tempat mereka dideklarasikan.
- **Hoisting** mengangkat deklarasi variabel dan fungsi ke atas scope mereka sebelum kode dieksekusi, tetapi inisialisasi variabel dan definisi fungsi (pada function expression) tidak diangkat.



# JAVASCRIPT DOM

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**



# Apa itu DOM



DOM (Document Object Model) adalah representasi struktur dokumen HTML atau XML dalam bentuk tree (pohon) yang memungkinkan program untuk mengakses dan memanipulasi konten, struktur, dan gaya halaman web secara dinamis.

- **Node:** Setiap elemen dalam tree DOM disebut "node". Ada beberapa jenis node, termasuk elemen, teks, dan atribut.
- **Root Node:** Dokumen HTML dimulai dari node akar (document), yang merupakan entry point untuk mengakses semua elemen lain di halaman.

# Mengakses DOM



JavaScript menyediakan beberapa metode untuk mengakses elemen di DOM:

- **getElementById**: Mengakses elemen berdasarkan ID.
- **getElementsByClassName**: Mengakses elemen berdasarkan class.
- **getElementsByTagName**: Mengakses elemen berdasarkan tag.
- **querySelector**: Mengakses elemen menggunakan selector CSS.
- **querySelectorAll**: Mengakses semua elemen yang sesuai dengan selector CSS.

# Memanipulasi Elemen



Setelah mengakses elemen, Anda bisa mengubah konten, atribut, dan gaya elemen tersebut.

- **Mengubah Konten:** `element.textContent` atau `element.innerHTML`
- **Mengubah Atribut:** `element.setAttribute(attribute, value)`
- **Mengubah Gaya:** `element.style.property = value`

# Menambahkan dan Menghapus Elemen DOM



Untuk menambahkan elemen baru ke DOM, Anda dapat menggunakan:

- **createElement(tagName)**: Membuat elemen baru.
- **appendChild(childNode)**: Menambahkan elemen sebagai anak dari elemen lain.
- **insertBefore(newNode, referenceNode)**: Menyisipkan elemen sebelum elemen referensi.

Untuk menghapus elemen dari DOM, gunakan:

- **removeChild(childNode)**: Menghapus elemen anak.
- **remove()**: Menghapus elemen secara langsung.

# DOM Traversal



DOM traversal adalah proses menjelajahi node di DOM tree untuk menemukan elemen tertentu. Anda dapat berpindah ke elemen anak, orang tua, atau saudara menggunakan berbagai properti dan metode.

## Properti DOM Traversal

- **parentNode**: Mendapatkan elemen orang tua.
- **childNodes**: Mendapatkan NodeList dari semua anak elemen.
- **firstChild/lastChild**: Mendapatkan anak pertama/terakhir.
- **nextSibling/previousSibling**: Mendapatkan elemen saudara berikutnya/sebelumnya.



# EVENT HANDLING DI JAVASCRIPT

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Event Handling



Event Handling adalah konsep yang sangat penting dalam pengembangan web. Ini memungkinkan Anda untuk menangkap dan merespons interaksi pengguna, seperti klik tombol, gerakan mouse, input keyboard, dan banyak lagi.



# Apa itu Event?



Event adalah sebuah tindakan atau kejadian yang terjadi di halaman web. Contohnya adalah klik pada tombol, pengisian formulir, atau pengguliran halaman. Event ini bisa ditangkap dan ditangani menggunakan JavaScript.





# Event Listener



Event listener adalah mekanisme untuk menangkap event dan menjalankan fungsi tertentu ketika event tersebut terjadi.

## Cara Menambahkan Event Listener:

### 1. Menggunakan onclick (Inline Event Handling):

html

```
<button onclick="alert('Button clicked!')">Click Me</button>
```

### 2. Menggunakan addEventListener (JavaScript):

javascript

```
const button = document.querySelector('button');  
button.addEventListener('click', () => {  
    alert('Button clicked!');  
});
```

# Jenis-Jenis Event yang Umum digunakan



## Mouse Events:

- `click` – Ketika elemen diklik.
- `dblclick` – Ketika elemen diklik dua kali.
- `mouseover` – Ketika mouse berada di atas elemen.
- `mouseout` – Ketika mouse meninggalkan elemen.

## Keyboard Events:

- `keydown` – Ketika sebuah tombol ditekan.
- `keyup` – Ketika sebuah tombol dilepaskan.
- `keypress` – Ketika sebuah tombol ditekan dan dilepaskan (umumnya tidak digunakan lagi).

## Form Events:

- `submit` – Ketika formulir dikirim.
- `change` – Ketika nilai elemen input berubah.
- `focus` – Ketika elemen form mendapatkan fokus.
- `blur` – Ketika elemen form kehilangan fokus.

## Window Events:

- `load` – Ketika halaman selesai dimuat.
- `resize` – Ketika ukuran jendela berubah.
- `scroll` – Ketika halaman digulir.



# Error Handling

**SUPPORT CHANNEL INI KE :  
SAWERIA.CO/KENAPACODING**

# Pendahuluan



**Error Handling** adalah proses menangani kesalahan dalam kode agar aplikasi tidak crash atau berhenti secara tiba-tiba. JavaScript menyediakan beberapa cara untuk menangani error, seperti menggunakan blok `try...catch`, `finally`, membuat error custom, serta menangani error dalam kode asynchronous.

# Try...Catch...Finally



`try...catch...finally` adalah cara yang paling umum untuk menangani error di JavaScript. Bagian `try` digunakan untuk mencoba menjalankan kode, dan jika terjadi error, bagian `catch` akan menangkap error tersebut dan memungkinkan kita menanganinya. Kita juga bisa menggunakan `finally` untuk menjalankan kode yang perlu dieksekusi terlepas dari apakah error terjadi atau tidak. **`finally` bersifat optional**

```
try {
  console.log("Mencoba menjalankan kode...");
  let number = parseInt("XYZ"); // Tidak bisa diubah menjadi angka
  if (isNaN(number)) {
    throw new Error("Nilai bukan angka yang valid!");
  }
} catch (error) {
  console.error("Terjadi kesalahan:", error.message);
} finally {
  console.log("Blok finally selalu dijalankan, terlepas dari berhasil atau tidaknya");
}
```

- Kode dalam blok `try` dieksekusi terlebih dahulu.
- Jika terjadi error, kontrol akan langsung beralih ke blok `catch`.
- Objek error berisi informasi tentang error yang terjadi, seperti `message`.
- Blok `finally` dijalankan setelah blok `try` dan `catch`, dan selalu dijalankan, bahkan jika ada `return` di blok `try` atau `catch`.

# Throwing Custom Errors



throw digunakan untuk membuat error kustom, yang bisa kita gunakan untuk menentukan sendiri kapan error harus dilempar.

javascript

```
function divide(a, b) {  
  if (b === 0) {  
    throw new Error("Pembagian dengan nol tidak diperbolehkan!");  
  }  
  return a / b;  
}  
  
try {  
  let result = divide(10, 0);  
  console.log(result);  
} catch (error) {  
  console.error("Error:", error.message);  
}
```

## Penjelasan:

- throw digunakan untuk melempar error secara manual jika suatu kondisi terpenuhi.
- Dalam contoh ini, kita melempar error jika mencoba membagi dengan angka 0.

# Beberapa Jenis-Jenis Error di JavaScript



## SyntaxError

Kesalahan ini terjadi jika ada kesalahan dalam sintaks kode. Contoh yang paling umum adalah tidak menutup tanda kurung atau kesalahan dalam penggunaan simbol tertentu.

```
javascript

try {
  console.log("Hello World" // Lupa menutup tanda kurung
} catch (error) {
  console.error("SyntaxError:", error.message);
}
```

## TypeError

Kesalahan ini muncul ketika kita mencoba menggunakan tipe nilai atau properti yang tidak sesuai dengan yang diharapkan, misalnya mengakses properti dari undefined atau null.

```
javascript

try {
  let num = null;
  console.log(num.toString()); // TypeError karena 'num' adalah null
} catch (error) {
  console.error("TypeError:", error.message);
}
```

## ReferenceError

Kesalahan ini muncul ketika kita mencoba mengakses variabel yang belum dideklarasikan.

```
javascript

try {
  console.log(myVar); // ReferenceError karena 'myVar' belum dideklarasikan
} catch (error) {
  console.error("ReferenceError:", error.message);
}
```

# Class Custom Errors



Kita dapat membuat error sendiri dengan mendefinisikan kelas yang mewarisi `Error`. Ini berguna untuk membuat pesan error khusus untuk aplikasi tertentu.

javascript

```
class CustomError extends Error {  
  constructor(message) {  
    super(message);  
    this.name = "CustomError";  
  }  
}  
  
try {  
  throw new CustomError("This is a custom error");  
} catch (error) {  
  console.error(`${error.name}: ${error.message}`);  
}
```

## Penjelasan:

- Kita membuat kelas `CustomError` yang mewarisi dari `Error`. Dengan cara ini, kita bisa membuat error khusus dengan nama dan pesan tertentu, yang membantu memberikan lebih banyak konteks saat terjadi kesalahan.





# Asynchronous JavaScript

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**

# Pendahuluan



JavaScript secara default adalah **single-threaded** dan berjalan secara **synchronous**, yang berarti kode dieksekusi satu per satu, dan setiap baris harus selesai sebelum baris berikutnya dapat dijalankan. Namun, dalam banyak situasi, kita tidak ingin program berhenti menunggu operasi selesai, seperti ketika mengambil data dari server atau membaca file besar. Untuk mengatasi ini, JavaScript menyediakan cara untuk menangani operasi asynchronous, sehingga tugas-tugas yang memakan waktu lama dapat diproses di latar belakang tanpa menghentikan eksekusi kode lainnya.

# Kenapa Asynchronous



**JavaScript** pada dasarnya adalah bahasa pemrograman **synchronous** dan **blocking**. Ini berarti bahwa JavaScript mengeksekusi kode baris per baris, dari atas ke bawah, dan tidak akan melanjutkan ke baris berikutnya sampai kode pada baris saat ini selesai dieksekusi. Dalam hal ini, setiap tugas atau operasi harus diselesaikan terlebih dahulu sebelum JavaScript dapat melanjutkan ke tugas berikutnya.

javascript

```
console.log("Start"); // Baris 1
for (let i = 0; i < 1000000000; i++) {}
console.log("End"); // Baris 3
```

Pada contoh di atas:

- Program akan mengeksekusi `console.log("Start")`, kemudian menjalankan loop (yang memakan waktu cukup lama), dan setelah loop selesai, baru mengeksekusi `console.log("End")`. Tidak ada operasi lain yang bisa dilakukan selama loop berlangsung.

**Synchronous** di sini berarti setiap operasi dilakukan satu per satu, secara berurutan. Ketika suatu operasi memakan waktu lama, ini disebut **blocking**.

# Blocking



**Blocking** terjadi ketika sebuah operasi menghalangi eksekusi kode lain sampai operasi tersebut selesai. Karena JavaScript secara default bekerja secara synchronous, operasi yang memakan waktu lama (seperti permintaan data dari server atau proses membaca file besar) bisa memblokir eksekusi kode lainnya. Ini bisa menyebabkan masalah dalam aplikasi web di mana proses seperti fetching data dari server, pengolahan data besar, atau operasi I/O memakan waktu dan menyebabkan aplikasi menjadi lambat atau tidak responsif.

javascript

```
console.log("Start"); // Baris 1
for (let i = 0; i < 10000000000; i++) {}
console.log("End"); // Baris 3
```

Jika loops kita adalah proses yang memakan waktu lama, seluruh program akan terhenti sampai loops tersebut selesai diambil. Ini akan memblokir semua kode lain dari eksekusi.

# Single-Threaded



JavaScript menggunakan model **single-threaded**. Ini berarti JavaScript memiliki satu "jalur" atau "thread" untuk menjalankan semua kode. Pada satu waktu, hanya satu tugas yang bisa dijalankan, dan JavaScript harus menyelesaikan tugas tersebut sebelum beralih ke tugas berikutnya. Ini berbeda dengan bahasa yang **multi-threaded**, di mana beberapa tugas bisa berjalan secara paralel.

## Apa itu Single Threaded?

- **Single-threaded** berarti hanya satu operasi atau satu bagian dari kode yang bisa berjalan pada satu waktu dalam event loop.
- Semua tugas, baik itu yang sederhana seperti menampilkan teks di konsol, atau yang kompleks seperti mengambil data dari server, dijalankan dalam satu thread.

Karena JavaScript hanya memiliki satu thread untuk mengeksekusi semua kode, tugas yang berat atau memakan waktu lama (seperti fetching data atau pengolahan data besar) dapat mengunci (block) thread tersebut. Hal ini akan mengakibatkan aplikasi menjadi tidak responsif, dan pengguna harus menunggu sampai tugas tersebut selesai.

## Masalah utama dari sifat single-threaded dan blocking ini adalah:

1. **UI menjadi tidak responsif:** Jika kode yang dijalankan memakan waktu lama (misalnya pengolahan data besar), antarmuka pengguna bisa menjadi "freeze" atau tidak bisa diakses sampai tugas tersebut selesai.
2. **Efisiensi rendah:** Semua tugas harus dijalankan satu per satu, tanpa adanya kemampuan untuk menjalankan beberapa tugas di background atau paralel.

# Mengapa Kita Membutuhkan Asynchronous JavaScript?

Untuk mengatasi keterbatasan dari single-threaded dan blocking code, **Asynchronous JavaScript** diperkenalkan. Dengan asynchronous code, JavaScript dapat melakukan tugas yang memakan waktu (seperti fetching data dari server) di latar belakang, tanpa memblokir eksekusi kode lainnya. Asynchronous memungkinkan kode lain untuk terus dijalankan sementara tugas yang memakan waktu selesai di background.

## Contoh Masalah:

Bayangkan aplikasi web yang harus mengambil data dari server. Jika dilakukan secara synchronous, browser akan "freeze" selama menunggu data dikirim kembali dari server, dan pengguna tidak akan bisa melakukan hal lain.

# Asynchronous



**Asynchronous** memungkinkan JavaScript untuk menjalankan operasi di background tanpa memblokir thread utama. Ini dilakukan dengan **callbacks**, **Promises**, dan **async/await**. Ketika operasi asynchronous dilakukan, JavaScript dapat mengeksekusi operasi lain terlebih dahulu, dan setelah operasi asynchronous selesai, JavaScript akan menangani hasil dari operasi tersebut.

Contoh Asynchronous :

javascript

```
console.log("Start");

setTimeout(() => {
  console.log("This is asynchronous code");
}, 2000); // Akan menunggu 2 detik, tetapi tidak akan memblokir kode lainnya

console.log("End");
```

Pada contoh di atas, meskipun `setTimeout` memakan waktu 2 detik, eksekusi `console.log("End")` tetap berjalan tanpa harus menunggu `setTimeout` selesai.

# Pengenalan `setTimeout` dan `setInterval`



`setTimeout` dan `setInterval` adalah dua fungsi penting dalam **Asynchronous JavaScript** yang memungkinkan kita untuk mengatur eksekusi kode secara asynchronous dengan jeda waktu tertentu atau secara berulang. Meskipun keduanya sangat berguna untuk tugas-tugas sederhana seperti penundaan dan eksekusi berulang, dalam aplikasi yang lebih kompleks, penggunaan **Promises** dan **Async/Await** sering lebih disarankan untuk menangani operasi asynchronous dengan cara yang lebih terstruktur dan mudah dibaca.



# setTimeout



Fungsi `setTimeout` sering digunakan untuk:

- Menunda eksekusi kode.
- Membuat efek animasi atau transisi.
- Mengatur timeout untuk operasi tertentu, seperti menunggu respon pengguna.

javascript

```
console.log('Mulai');

setTimeout(() => {
  console.log('Dijeda 3 detik');
}, 3000);

console.log('Akan muncul sebelum delay selesai');
```

# setInterval



Fungsi `setInterval` sering digunakan untuk:

- Membuat timer atau jam.
- Menjalankan polling terhadap server.
- Mengupdate UI secara berkala, seperti slideshow atau news ticker.

javascript

```
let timer = 0;

const intervalId = setInterval(() => {
  timer += 1;
  console.log(`Timer: ${timer} detik`);

  if (timer === 5) {
    clearInterval(intervalId);
    console.log('Timer dihentikan');
  }
}, 1000);
```

# Callback Functions



Callback function adalah function yang dikirim sebagai argumen ke function lain dan dipanggil di dalam function tersebut.

javascript

```
function selesaikanTugas(tugas, callback) {  
  console.log("Menyelesaikan tugas: " + tugas);  
  callback();  
}  
  
function tugasSelesai() {  
  console.log("Tugas selesai!");  
}  
  
selesaikanTugas("Belajar JavaScript", tugasSelesai);
```

# Contoh Asynchronous Callback



```
javascript

function getUserData(callback) {
  console.log("Fetching user data...");

  // Simulasi pengambilan data dari server yang membutuhkan waktu 2 detik
  setTimeout(() => {
    const userData = {
      id: 1,
      name: "John Doe",
      email: "john.doe@example.com"
    };

    // Setelah data "diperoleh", kita panggil callback
    callback(null, userData);
  }, 2000);
}

// Fungsi untuk memproses data pengguna setelah didapatkan
function displayUserData(error, data) {
  if (error) {
    console.error("Error fetching user data:", error);
  } else {
    console.log("User Data:", data);
  }
}

// Memanggil fungsi untuk mendapatkan data pengguna
getUserData(displayUserData);
```

Misalkan kita ingin mengambil data pengguna dari "server". Karena proses pengambilan data ini butuh waktu, kita akan mensimulasikannya menggunakan `setTimeout`. Callback akan digunakan untuk mengeksekusi kode setelah data "diperoleh".

- **Callback** adalah fungsi yang dipanggil setelah operasi asynchronous selesai dilakukan.
- Dalam contoh ini, **`setTimeout`** digunakan untuk mensimulasikan operasi asynchronous seperti pengambilan data dari server.
- Setelah data "didapatkan", **callback** dipanggil untuk memproses data tersebut.

Callback memungkinkan kita untuk memproses data hanya setelah data diambil dari API, tanpa memblokir eksekusi kode lainnya.

# Callback Hell



**Callback Hell** terjadi ketika kita memiliki banyak operasi asynchronous yang bergantung satu sama lain, sehingga callback dipanggil berulang kali di dalam callback sebelumnya. Ini sering kali membuat kode menjadi sulit dibaca dan dipelihara, karena indentasi yang terus bertambah. Berikut adalah contoh sederhana yang menunjukkan masalah ini.

Misalkan kita memiliki beberapa operasi yang harus dilakukan secara berurutan, misalnya:

1. Mengambil data pengguna.
2. Mengambil postingan pengguna berdasarkan data pengguna.
3. Mengambil komentar pada postingan tersebut.

# Callback Hell



```
function getUser(userId, callback) {
  setTimeout(() => {
    console.log("Fetching user...");
    callback(null, { id: userId, name: "John Doe" });
  }, 1000);
}
```

```
function getUserPosts(userId, callback) {
  setTimeout(() => {
    console.log(`Fetching posts for user ${userId}...`);
    callback(null, [
      { id: 101, title: "Post 1" },
      { id: 102, title: "Post 2" }
    ]);
  }, 1000);
}
```

```
function getPostComments(postId, callback) {
  setTimeout(() => {
    console.log(`Fetching comments for post ${postId}...`);
    callback(null, [
      { id: 1, comment: "Great post!" },
      { id: 2, comment: "Thanks for sharing!" }
    ]);
  }, 1000);
}
```

```
// Memulai dengan mengambil data pengguna
getUser(1, (userError, user) => {
  if (userError) {
    console.error("Error fetching user:", userError);
  } else {
    console.log("User:", user);

    // Mengambil postingan pengguna
    getUserPosts(user.id, (postsError, posts) => {
      if (postsError) {
        console.error("Error fetching posts:", postsError);
      } else {
        console.log("Posts:", posts);

        // Mengambil komentar dari posting pertama
        getPostComments(posts[0].id, (commentsError, comments) => {
          if (commentsError) {
            console.error("Error fetching comments:", commentsError);
          } else {
            console.log("Comments for Post 1:", comments);
          }
        });
      }
    });
  }
});
```

# Masalah Callback Hell



## Indentasi Berulang:

- Setiap callback berada di dalam callback sebelumnya, menyebabkan banyaknya indentasi, sehingga kode tampak berantakan.

## Sulit Dibaca dan Dipelihara:

- Jika terdapat lebih banyak operasi yang bergantung satu sama lain, maka tingkat kedalaman callback akan semakin dalam dan semakin sulit dipahami.

# Mengatasi Callback Hell:



## Promises:

- Kita dapat mengganti callback dengan Promise untuk membuat kode lebih bersih dan lebih mudah dibaca.

## Async/Await:

- Penggunaan `async` dan `await` memberikan cara yang lebih linear untuk menangani operasi asynchronous dan membuat kode lebih mudah dibaca seperti kode yang berjalan secara sinkron.



# Promises



**Promises** adalah cara untuk menangani operasi asynchronous secara lebih bersih daripada callback. Promise adalah objek yang mewakili **keberhasilan atau kegagalan** suatu operasi asynchronous.

## States of Promise:

1. **Pending:** Promise sedang berjalan.
2. **Fulfilled:** Promise berhasil diselesaikan.
3. **Rejected:** Promise mengalami kegagalan.

# Contoh Membuat Promise



```
function checkStock(product) {
  return new Promise((resolve, reject) => {
    console.log(`Checking stock for ${product}...`);

    // Simulasi waktu untuk mengecek stok (misalnya 2 detik)
    setTimeout(() => {
      const stockAvailable = true; // Ganti menjadi false untuk melihat hasil reject

      if (stockAvailable) {
        resolve(`${product} is available in stock.`);
      } else {
        reject(`${product} is out of stock.`);
      }
    }, 2000);
  });
}

// Memanggil fungsi checkStock
checkStock("Laptop")
  .then(message => {
    // Menangani jika Promise berhasil (resolved)
    console.log(message);
  })
  .catch(error => {
    // Menangani jika Promise gagal (rejected)
    console.error(error);
  });
```

# Async/Await



**Async/Await** adalah sintaks modern untuk menangani operasi asynchronous secara lebih sederhana dan terlihat seperti kode synchronous, namun tetap asynchronous.

- **Async:** Digunakan untuk mendeklarasikan fungsi agar dapat menggunakan await.
- **Await:** Digunakan untuk menunggu penyelesaian sebuah Promise, dan kode akan berhenti sejenak hingga promise selesai.

javascript

```
function getData() {  
  return new Promise((resolve) => {  
    setTimeout(() => {  
      resolve("Data fetched");  
    }, 2000);  
  });  
}  
  
async function fetchData() {  
  console.log("Fetching data...");  
  const data = await getData();  
  console.log(data);  
}  
  
fetchData();
```

- `getData` adalah sebuah fungsi yang mengembalikan promise dan akan selesai setelah dua detik.
- `await` digunakan untuk menunggu `getData()` menyelesaikan pekerjaannya sebelum melanjutkan eksekusi ke `console.log(data)`.



# WORKING WITH API

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Apa Itu API?



API (Application Programming Interface) memungkinkan pengembang untuk mengakses data atau layanan dari aplikasi lain tanpa perlu memahami bagaimana aplikasi tersebut bekerja di dalamnya. API menyediakan cara untuk melakukan hal-hal seperti:

- Mengambil data dari server.
- Mengirim data ke server.
- Mengautentikasi pengguna.
- Mengakses layanan pihak ketiga (misalnya Google Maps, OpenWeather, dsb.).

## API:

- REST API (Representational State Transfer): API berbasis HTTP yang digunakan untuk berkomunikasi antara klien dan server.
- JSON (JavaScript Object Notation): Format data yang sering digunakan dalam komunikasi API.

# HTTP Request Methods



API biasanya bekerja melalui HTTP dengan beberapa metode standar, yaitu:

1. **GET**: Mengambil data dari server.
2. **POST**: Mengirim data baru ke server.
3. **PUT**: Mengganti atau memperbarui data di server.
4. **DELETE**: Menghapus data dari server.

# HTTP Status Code



Setiap kali kita mengirimkan permintaan HTTP, kita akan menerima kode status sebagai respon, seperti:

- **200 OK:** Permintaan berhasil.
- **201 Created:** Sumber daya berhasil dibuat.
- **400 Bad Request:** Permintaan tidak valid.
- **404 Not Found:** Sumber daya tidak ditemukan.
- **500 Internal Server Error:** Server mengalami kesalahan.

# fetch() API



`fetch()` adalah cara modern dan lebih sederhana untuk mengirim permintaan HTTP ke server dan bekerja dengan API. `fetch()` adalah promise-based, sehingga lebih mudah dikelola.

javascript

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json();
  })
  .then(data => {
    console.log("Data fetched:", data);
  })
  .catch(error => {
    console.error("Error fetching data:", error);
  });
```

## Penjelasan:

- `fetch()` menerima URL API sebagai parameter dan mengembalikan sebuah Promise.
- `response.ok` memeriksa apakah permintaan berhasil.
- `response.json()` digunakan untuk mengonversi response menjadi format JSON.
- `.catch()` menangani kesalahan yang mungkin terjadi selama proses permintaan.



# fetch() untuk POST Data



javascript

```
const postData = {
  title: 'New Post',
  body: 'This is the body of the new post.',
  userId: 1
};

fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(postData)
})
  .then(response => response.json())
  .then(data => {
    console.log("Data posted:", data);
  })
  .catch(error => {
    console.error("Error posting data:", error);
  });
```

## Penjelasan:

- **method:** Menentukan metode HTTP (dalam hal ini, POST).
- **headers:** Menentukan tipe konten yang dikirim (Content-Type: application/json).
- **body:** Data yang akan dikirim dalam format JSON.

# Menggunakan `async/await` dengan `fetch()`



Dengan `async/await`, kode kita menjadi lebih mudah dibaca dan ditulis, terutama saat menangani operasi asynchronous.

javascript

Copy code

```
async function getPost() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/posts/1');
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json();
    console.log("Data fetched using async/await:", data);
  } catch (error) {
    console.error("Error fetching data:", error);
  }
}

getPost();
```

## Penjelasan:

- **async** digunakan untuk mendeklarasikan fungsi asynchronous.
- **await** menunggu `fetch()` selesai sebelum mengeksekusi baris kode berikutnya.
- Blok **try...catch** digunakan untuk menangani error.

# Menggunakan Library seperti Axios



**Axios** adalah library JavaScript yang mempermudah kerja dengan API karena lebih singkat dan mendukung fitur seperti transformasi request/response otomatis dan pembatalan request.

Instalasi Axios (menggunakan npm atau CDN):

```
axios.get('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    console.log("Data fetched with Axios:", response.data);
  })
  .catch(error => {
    console.error("Axios error:", error);
  });
```

```
// Fungsi untuk mengambil data (GET) menggunakan async/await
async function getData() {
  try {
    const response = await axios.get('https://jsonplaceholder.typicode.com/posts/1')
    // Menampilkan data di konsol
    console.log('Data:', response.data);
  } catch (error) {
    // Menangani error
    console.error('Error:', error.message);
  }
}

getData();
```



# OOP DI JAVASCRIPT

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**

# Apa itu OOP?



Pemrograman Berbasis Objek (OOP) adalah paradigma pemrograman yang berfokus pada penggunaan "objek" untuk mewakili data dan fungsionalitas dalam program. Konsep OOP memungkinkan Anda untuk memodelkan elemen-elemen dunia nyata dalam kode, membuatnya lebih mudah dipahami dan dikelola.



# Dasar-Dasar Objek di JavaScript



Objek adalah kumpulan pasangan kunci-nilai (key-value pairs) yang dapat mewakili data dan perilaku. Di JavaScript, objek bisa dibuat dengan cara yang sederhana.

javascript

```
let mobil = {  
  merk: "Toyota",  
  model: "Avanza",  
  tahun: 2021,  
  start: function() {  
    console.log("Mobil dimulai");  
  },  
  info: function() {  
    console.log(`Mobil: ${this.merk} ${this.model}, Tahun: ${this.tahun}`)  
  }  
};  
  
mobil.start(); // Output: Mobil dimulai  
mobil.info(); // Output: Mobil: Toyota Avanza, Tahun: 2021
```

## NOTE :

- merk, model, dan tahun adalah **properti** objek.
- start dan info adalah **method** (fungsi yang ada di dalam objek).
- this mengacu pada objek itu sendiri (dalam hal ini mobil).

# Constructor Function



Constructor function adalah cara lain untuk membuat objek yang memungkinkan Anda membuat banyak objek dengan properti dan method yang sama.

javascript

```
function Mobil(merk, model, tahun) {  
  this.merk = merk;  
  this.model = model;  
  this.tahun = tahun;  
  
  this.start = function() {  
    console.log(`${this.merk} dimulai`);  
  };  
  
  this.info = function() {  
    console.log(`Mobil: ${this.merk} ${this.model}, Tahun: ${this.tahun}`);  
  };  
}
```

```
let mobil1 = new Mobil("Toyota", "Avanza", 2021);
```

```
let mobil2 = new Mobil("Honda", "Civic", 2020);
```

```
mobil1.start(); // Output: Toyota dimulai
```

```
mobil2.info(); // Output: Mobil: Honda Civic, Tahun: 2020
```

## NOTE :

- **new** digunakan untuk membuat instance baru dari Mobil.

# Konsep Prototypal Inheritance



JavaScript menggunakan **prototypal inheritance**, yang berarti objek dapat mewarisi properti dan method dari objek lain.

javascript

```
function Hewan(nama, jenis) {  
  this.nama = nama;  
  this.jenis = jenis;  
}  
  
Hewan.prototype.makan = function() {  
  console.log(`${this.nama} sedang makan.`);  
};  
  
let kucing = new Hewan("Kitty", "Kucing");  
kucing.makan(); // Output: Kitty sedang makan.
```

## NOTE :

- **Prototype** adalah objek dari mana objek lain mewarisi properti dan method.



# ES6 Classes



Dengan ES6, JavaScript memperkenalkan sintaks **class** yang lebih mudah dipahami dan digunakan.

javascript

```
class Mobil {
  constructor(merk, model, tahun) {
    this.merk = merk;
    this.model = model;
    this.tahun = tahun;
  }

  start() {
    console.log(`${this.merk} dimulai`);
  }

  info() {
    console.log(`Mobil: ${this.merk} ${this.model}, Tahun: ${this.tahun}`);
  }
}

let mobil1 = new Mobil("Toyota", "Avanza", 2021);
let mobil2 = new Mobil("Honda", "Civic", 2020);

mobil1.start(); // Output: Toyota dimulai
mobil2.info(); // Output: Mobil: Honda Civic, Tahun: 2020
```

## NOTE :

- **Class** adalah blueprint untuk membuat objek.
- **Constructor** adalah method khusus untuk menginisialisasi objek baru.

# Inheritance dengan Class



Inheritance memungkinkan Anda membuat class baru yang mewarisi properti dan method dari class lain.

```
javascript

class Hewan {
  constructor(nama, jenis) {
    this.nama = nama;
    this.jenis = jenis;
  }

  makan() {
    console.log(`${this.nama} sedang makan.`);
  }
}

class Kucing extends Hewan {
  constructor(nama, warna) {
    super(nama, "Kucing");
    this.warna = warna;
  }

  bersuara() {
    console.log("Meong!");
  }
}

let kucing = new Kucing("Kitty", "Putih");
kucing.makan(); // Output: Kitty sedang makan.
kucing.bersuara(); // Output: Meong!
```

## NOTE :

- **extends** digunakan untuk membuat subclass.
- **super** memanggil constructor dari class induk.

# Encapsulation



Encapsulation adalah konsep untuk membatasi akses ke properti dan method dari objek. JavaScript mengimplementasikan ini dengan penggunaan simbol underscore \_ atau menggunakan closures.

javascript

```
class BankAccount {  
  constructor(owner, balance) {  
    this.owner = owner;  
    this._balance = balance;  
  }  
  
  deposit(amount) {  
    this._balance += amount;  
    console.log(`Deposit: ${amount}`);  
  }  
}
```

Penggunaan `_balance` mengindikasikan bahwa properti ini "pribadi", meskipun masih bisa diakses (JavaScript tidak mendukung encapsulation private secara ketat di ES6).

```
  withdraw(amount) {  
    if (amount > this._balance) {  
      console.log("Saldo tidak mencukupi.");  
    } else {  
      this._balance -= amount;  
      console.log(`Withdraw: ${amount}`);  
    }  
  }  
  
  getBalance() {  
    console.log(`Saldo: ${this._balance}`);  
  }  
}
```

```
let akun = new BankAccount("John Doe", 1000);  
akun.deposit(500); // Output: Deposit: 500  
akun.withdraw(200); // Output: Withdraw: 200  
akun.getBalance(); // Output: Saldo: 1300
```

# Polymorphism



Polymorphism memungkinkan Anda untuk menggunakan method dengan nama yang sama pada objek yang berbeda.

javascript

```
class Hewan {  
  bersuara() {  
    console.log("Hewan bersuara.");  
  }  
}  
  
class Kucing extends Hewan {  
  bersuara() {  
    console.log("Meong!");  
  }  
}
```

```
class Anjing extends Hewan {  
  bersuara() {  
    console.log("Guk Guk!");  
  }  
}  
  
let hewan = new Hewan();  
let kucing = new Kucing();  
let anjing = new Anjing();  
  
hewan.bersuara(); // Output: Hewan bersuara.  
kucing.bersuara(); // Output: Meong!  
anjing.bersuara(); // Output: Guk Guk!
```

Method `bersuara()` digunakan pada semua class, tetapi memberikan output yang berbeda sesuai dengan class-nya.

# Abstraction (Abstraksi)



Abstraction adalah proses menyembunyikan detail implementasi dari pengguna dan hanya menampilkan esensi atau fitur utama. Ini dilakukan dengan menggunakan abstract class atau interface (tidak sepenuhnya didukung dalam JavaScript, tetapi dapat disimulasikan).

javascript

```
class Shape {
  constructor(name) {
    if (this.constructor === Shape) {
      throw new Error("Cannot instantiate abstract class");
    }
    this.name = name;
  }

  calculateArea() {
    throw new Error("Abstract method must be implemented");
  }
}
```

```
class Rectangle extends Shape {
  constructor(width, height) {
    super("Rectangle");
    this.width = width;
    this.height = height;
  }

  calculateArea() {
    return this.width * this.height;
  }
}

let myRectangle = new Rectangle(10, 20);
console.log(myRectangle.calculateArea()); // Output: 200
```

- Shape adalah abstraksi yang tidak dapat diinstansiasi secara langsung.
- Subclass seperti Rectangle harus mengimplementasikan method calculateArea.

# Manfaat dan Keterbatasan OOP



OOP memberikan beberapa keuntungan utama:

- **Modularitas:** Kode dapat dipisahkan ke dalam kelas yang berbeda, membuatnya lebih mudah dikelola dan diperluas.
- **Reusability:** Kelas yang sudah ada dapat digunakan kembali dalam proyek lain.
- **Scalability:** Mudah untuk menambahkan fungsionalitas baru dengan membuat kelas baru atau mewarisi dari kelas yang sudah ada.
- **Maintainability:** Perubahan pada satu bagian kode tidak mempengaruhi bagian lain, berkat enkapsulasi dan modularitas.

Meskipun OOP memiliki banyak manfaat, ia juga memiliki beberapa keterbatasan:

- **Kompleksitas:** Struktur kelas dan objek yang kompleks bisa membuat program lebih sulit untuk dipahami, terutama bagi pemula.
- **Overhead:** Penggunaan objek dan inheritance bisa menambah overhead pada kinerja program.
- **Tidak selalu cocok:** Beberapa masalah atau proyek mungkin lebih mudah diselesaikan dengan paradigma pemrograman lain, seperti pemrograman fungsional.



# JAVASCRIPT MODULES

SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)

# Pengenalan JavaScript Modules



JavaScript modules adalah fitur yang memungkinkan kita untuk membagi kode JavaScript ke dalam file terpisah. Ini membuat kode lebih terstruktur, lebih mudah untuk dipelihara, dan meningkatkan keterbacaan. Dengan modules, Anda dapat mengelompokkan fungsionalitas tertentu dalam file yang terpisah, dan kemudian mengimpor file tersebut sesuai kebutuhan.



# Mengapa Menggunakan Modules?



- **Pemeliharaan yang Lebih Mudah:** Dengan memisahkan kode menjadi bagian-bagian yang lebih kecil, pemeliharaan dan debugging menjadi lebih mudah.
- **Penggunaan Ulang Kode:** Modules memudahkan penggunaan ulang kode di tempat lain tanpa perlu menulis ulang fungsi yang sama.
- **Meningkatkan Kejelasan:** Kode yang dibagi menjadi file yang lebih kecil akan lebih mudah dipahami dibandingkan dengan kode dalam satu file besar.

# Tipe Module JavaScript



1. **ES Modules (ECMAScript Modules):** ES Modules menggunakan sintaks `import` dan `export`. Ini adalah standar yang ditentukan oleh ECMAScript.
2. **CommonJS Modules:** Modul yang biasanya digunakan dalam lingkungan Node.js. Menggunakan `require` dan `module.exports`.

# ES Modules



Anda dapat menggunakan `export` untuk mengeksport fungsi, variabel, atau objek dari suatu file, yang kemudian dapat diimpor oleh file lain. Setelah Anda mengeksport, Anda dapat mengimpor nilai tersebut ke file lain.

## Export Named (Ekspor Bernama)

javascript

```
// file: math.js
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
```

## Export Default

javascript

```
// file: greet.js
export default function greet(name) {
  console.log(`Hello, ${name}!`);
}
```

## Import Named

javascript

```
// file: main.js
import { add, subtract } from './math.js';

console.log(add(5, 3)); // Output: 8
console.log(subtract(5, 3)); // Output: 2
```

## Import Default

javascript

```
// file: main.js
import greet from './greet.js';

greet('Alice'); // Output: Hello, Alice!
```

# CommonJS Modules (Node.js)



CommonJS adalah sistem modul yang umumnya digunakan di Node.js. Ini menggunakan `require` untuk mengimpor dan `module.exports` untuk mengekspor.

## Ekspor dengan `module.exports`

javascript

```
// file: math.js
const add = (a, b) => a + b;
const subtract = (a, b) => a - b;

module.exports = { add, subtract };
```

## Impor dengan `require`

javascript

```
// file: main.js
const { add, subtract } = require('./math');

console.log(add(5, 3));           // Output: 8
console.log(subtract(5, 3));     // Output: 2
```

# Dynamic Import



Dengan **dynamic import**, Anda dapat mengimpor modul hanya ketika dibutuhkan, yang dapat meningkatkan kinerja aplikasi.

javascript

```
document.getElementById('load').addEventListener('click', async () => {  
  const module = await import('./math.js');  
  console.log(module.add(4, 5)); // Output: 9  
});
```



# Pengenalan Testing JavaScript

**SUPPORT CHANNEL INI KE :  
[SAWERIA.CO/KENAPACODING](https://saweria.co/kenapacoding)**

# Apa itu Testing



Testing adalah proses penting dalam pengembangan perangkat lunak yang bertujuan untuk memastikan bahwa kode berfungsi sesuai harapan. Dalam JavaScript, terdapat beberapa cara dan alat untuk melakukan testing, termasuk menggunakan library seperti **Jest** atau **Mocha** untuk unit testing, serta pendekatan **Test-Driven Development (TDD)** dan penggunaan **mocking** dan **spying**. Berikut ini penjelasan lebih detail.

# Apa itu Testing



Testing adalah proses penting dalam pengembangan perangkat lunak yang bertujuan untuk memastikan bahwa kode berfungsi sesuai harapan. Dalam JavaScript, terdapat beberapa cara dan alat untuk melakukan testing, termasuk menggunakan library seperti **Jest** atau **Mocha** untuk unit testing, serta pendekatan **Test-Driven Development (TDD)** dan penggunaan **mocking** dan **spying**. Berikut ini penjelasan lebih detail.



# Unit Testing



**Unit testing** adalah proses pengujian bagian kecil dari perangkat lunak secara terisolasi, biasanya sebuah fungsi atau modul. Ini membantu mengidentifikasi kesalahan lebih awal, membuat kode lebih dapat diandalkan, dan memudahkan maintenance.



# Jest



**Jest** adalah library testing JavaScript yang dikembangkan oleh Facebook, sangat cocok digunakan untuk proyek React, namun dapat digunakan untuk pengujian aplikasi JavaScript pada umumnya.



# Jest



- Instalasi Jest

```
bash
```

```
npm install --save-dev jest
```

- Contoh Unit Test dengan Jest

```
javascript
```

```
// fungsi yang akan diuji
function add(a, b) {
  return a + b;
}

// unit test menggunakan Jest
test('Menambahkan 2 + 3 harusnya sama dengan 5', () => {
  expect(add(2, 3)).toBe(5);
});
```

- Untuk menjalankan test, tambahkan "test": "jest" di dalam file package.json, lalu jalankan:

```
bash
```

```
npm run test
```

# Test-Driven Development (TDD)



**Test-Driven Development (TDD)** adalah pendekatan dalam pengembangan perangkat lunak di mana pengembang menulis tes terlebih dahulu sebelum menulis kode yang sebenarnya. Proses TDD meliputi langkah-langkah berikut:

1. **Menulis Test:** Buat tes untuk sebuah fitur sebelum fitur tersebut diimplementasikan.
2. **Menjalankan Tes:** Pastikan tes gagal, karena kode fitur belum ada.
3. **Menulis Kode:** Buat kode untuk memenuhi tes tersebut.
4. **Menjalankan Tes Ulang:** Pastikan tes berhasil.
5. **Refactor:** Refactor kode untuk meningkatkan kualitasnya tanpa mengubah fungsionalitas.