



Pendahuluan

Modul ajar ini dirancang untuk memberikan pemahaman komprehensif tentang JavaScript, mulai dari dasar hingga konsep lanjutan. Modul ini terbagi menjadi 10 pertemuan, di mana setiap pertemuan akan mencakup penjelasan materi, tujuan pembelajaran, contoh kode, dan langkah-langkah praktikum.

Tujuan Pembelajaran Umum

Setelah menyelesaikan modul ini, peserta diharapkan mampu:

- Memahami dasar-dasar pemrograman JavaScript.
- Menggunakan JavaScript untuk memanipulasi DOM dan membuat interaksi pada halaman web.
- Memahami konsep asynchronous JavaScript.
- Menggunakan fitur-fitur modern JavaScript (ES6+).
- Memiliki dasar pengetahuan tentang Node.js dan NPM.

Pertemuan 1: Pengenalan JavaScript dan Dasar-dasar

Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Memahami apa itu JavaScript dan sejarah singkatnya.
- Mengetahui di mana JavaScript digunakan.
- Menyiapkan lingkungan pengembangan dasar untuk JavaScript.
- Menulis dan menjalankan kode JavaScript pertama mereka.
- Memahami konsep dasar sintaks JavaScript.

Penjelasan Materi

JavaScript adalah bahasa pemrograman tingkat tinggi yang sangat populer, terutama digunakan untuk pengembangan web. Awalnya, JavaScript (dulu dikenal sebagai LiveScript) dibuat oleh Brendan Eich di Netscape pada tahun 1995 untuk menambahkan interaktivitas ke halaman web. Sejak itu, JavaScript telah berkembang pesat dan menjadi salah satu dari tiga teknologi inti World Wide Web, bersama dengan HTML dan CSS. [1]

JavaScript adalah bahasa yang *interpreted*, yang berarti kode dijalankan baris demi baris oleh *interpreter* (seperti browser web) tanpa perlu kompilasi sebelumnya. Ini berbeda dengan bahasa seperti Java atau C++ yang memerlukan proses kompilasi sebelum dijalankan.

Di mana JavaScript Digunakan?

1. **Pengembangan Web Frontend:** Ini adalah penggunaan paling umum. JavaScript memungkinkan interaktivitas pada halaman web, seperti animasi, validasi formulir, galeri gambar, dan pembaruan konten dinamis tanpa memuat ulang halaman. Library dan framework seperti React, Angular, dan Vue.js sangat bergantung pada JavaScript.
2. **Pengembangan Web Backend (Node.js):** Dengan munculnya Node.js, JavaScript dapat dijalankan di sisi server. Ini memungkinkan pengembang untuk menggunakan satu bahasa (JavaScript) untuk seluruh tumpukan pengembangan (frontend dan backend), yang dikenal sebagai *full-stack JavaScript development*.
3. **Pengembangan Aplikasi Mobile:** Framework seperti React Native dan NativeScript memungkinkan pengembangan aplikasi mobile lintas platform menggunakan JavaScript.
4. **Pengembangan Desktop:** Aplikasi desktop dapat dibangun menggunakan JavaScript dengan framework seperti Electron (digunakan oleh aplikasi seperti VS Code, Slack, dan Discord).
5. **Game Development:** Beberapa game berbasis web atau bahkan game desktop sederhana dapat dibuat dengan JavaScript.

Sintaks Dasar JavaScript

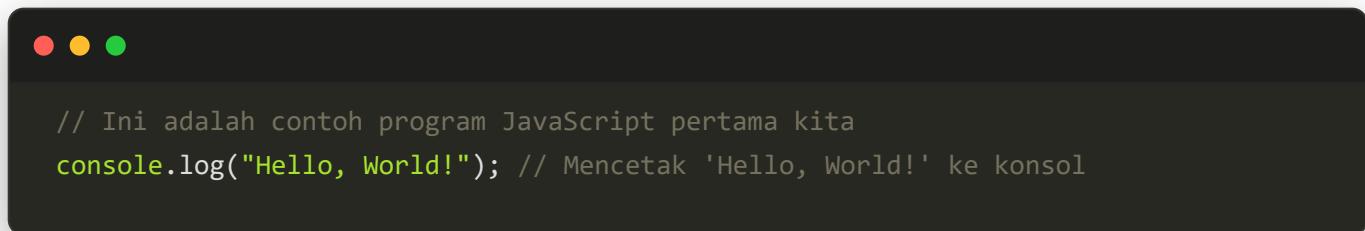
JavaScript memiliki sintaks yang mirip dengan C atau Java. Beberapa aturan dasar yang perlu diingat:

- **Case-sensitive:** `myVariable` berbeda dengan `myvariable`.
- **Pernyataan diakhiri dengan titik koma (semicolon):** Meskipun seringkali opsional di JavaScript modern karena *Automatic Semicolon Insertion (ASI)*, sangat disarankan untuk menggunakannya untuk kejelasan dan menghindari potensi bug.
- **Komentar:** Digunakan untuk menjelaskan kode dan tidak dieksekusi oleh interpreter.
 - Komentar satu baris: `// Ini adalah komentar satu baris`
 - Komentar multi-baris: `/* Ini adalah komentar multi-baris */`

Kode Contoh: Hello World!

Ini adalah program

pertama yang biasa dibuat oleh programmer untuk memastikan lingkungan pengembangan mereka berfungsi dengan baik.



```
// Ini adalah contoh program JavaScript pertama kita
console.log("Hello, World!"); // Mencetak 'Hello, World!' ke konsol
```

Penjelasan Kode:

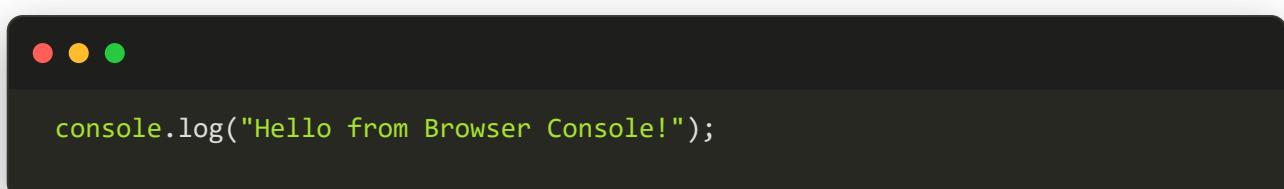
- `console.log()`: Ini adalah fungsi bawaan JavaScript yang digunakan untuk mencetak pesan ke konsol browser atau terminal Node.js. Ini sangat berguna untuk *debugging*.
- `"Hello, World!"`: Ini adalah *string literal*, yaitu teks yang akan dicetak. Teks harus diapit oleh tanda kutip tunggal (`'`) atau ganda (`"`).

Langkah-langkah Praktikum: Menjalankan JavaScript Pertama Anda

Ada beberapa cara untuk menjalankan kode JavaScript:

Metode 1: Menggunakan Konsol Browser (Paling Mudah untuk Pemula)

1. Buka browser web favorit Anda (Chrome, Firefox, Edge, dll.).
2. Tekan `F12` atau klik kanan di mana saja pada halaman web dan pilih "Inspect" atau "Inspect Element". Ini akan membuka Developer Tools.
3. Pilih tab "Console".
4. Ketik kode berikut di prompt konsol dan tekan `Enter`:



```
console.log("Hello from Browser Console!");
```

5. Anda akan melihat output `Hello from Browser Console!` di konsol.

Metode 2: Menggunakan File HTML (Cara Umum untuk Web Development)

1. Buka editor teks Anda (VS Code, Sublime Text, Notepad++, dll.).
2. Buat file baru dan simpan dengan nama `index.html`.
3. Salin dan tempel kode berikut ke dalam `index.html`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My First JavaScript</title>
</head>
<body>
    <h1>Hello JavaScript!</h1>

    <script>
        // Kode JavaScript Anda di sini
        console.log("Hello from HTML file!");
    </script>
</body>
</html>
```

4. Simpan file `index.html`.
5. Buka file `index.html` di browser Anda (Anda bisa menyeret file ke jendela browser atau mengklik dua kali file tersebut).
6. Buka Developer Tools (`F12`) dan lihat tab "Console". Anda akan melihat output `Hello from HTML file!`.

Metode 3: Menggunakan File JavaScript Eksternal (Praktik Terbaik untuk Proyek Besar)

1. Buat dua file di folder yang sama: `index.html` dan `script.js`.
2. Salin dan tempel kode berikut ke dalam `index.html`:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>External JavaScript</title>
</head>
<body>
    <h1>Hello External JavaScript!</h1>

    <!-- Menghubungkan file JavaScript eksternal -->
    <script src="script.js"></script>
</body>
</html>
```

3. Salin dan tempel kode berikut ke dalam `script.js` :

```
// Kode JavaScript Anda di sini
console.log("Hello from external script.js!");
```

4. Simpan kedua file.

5. Buka `index.html` di browser Anda.

6. Buka Developer Tools (`F12`) dan lihat tab "Console". Anda akan melihat output
`Hello from external script.js!`.

Metode 4: Menggunakan Node.js (Untuk JavaScript di Luar Browser)

1. **Instal Node.js:** Jika Anda belum menginstalnya, kunjungi nodejs.org dan unduh installer yang sesuai dengan sistem operasi Anda. Ikuti instruksi instalasi.

2. Setelah instalasi selesai, buka terminal atau Command Prompt Anda.

3. Buat file baru bernama `app.js` di folder mana pun.

4. Salin dan tempel kode berikut ke dalam `app.js` :

```
// Kode JavaScript untuk Node.js
console.log("Hello from Node.js!");
```

5. Simpan file `app.js`.

6. Buka terminal atau Command Prompt, navigasikan ke direktori tempat Anda menyimpan `app.js`.

7. Jalankan perintah berikut:



```
node app.js
```

8. Anda akan melihat output `Hello from Node.js!` di terminal Anda.

Referensi

[1] Mozilla. (n.d.). *JavaScript*. MDN Web Docs. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

Pertemuan 2: Tipe Data, Variabel, dan Operator

Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Memahami berbagai tipe data primitif dan non-primitif dalam JavaScript.
- Mendeklarasikan dan menggunakan variabel dengan benar.
- Memahami konsep *scope* variabel (global dan lokal).
- Menggunakan berbagai jenis operator dalam JavaScript.

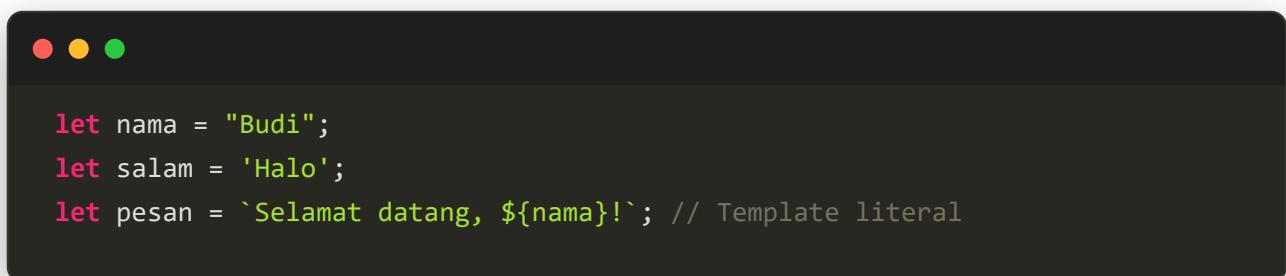
Penjelasan Materi

Tipe Data

JavaScript adalah bahasa dengan *dynamic typing*, yang berarti Anda tidak perlu secara eksplisit mendeklarasikan tipe data variabel saat mendeklarasikannya. Tipe data akan ditentukan secara otomatis saat kode dieksekusi. JavaScript memiliki beberapa tipe data:

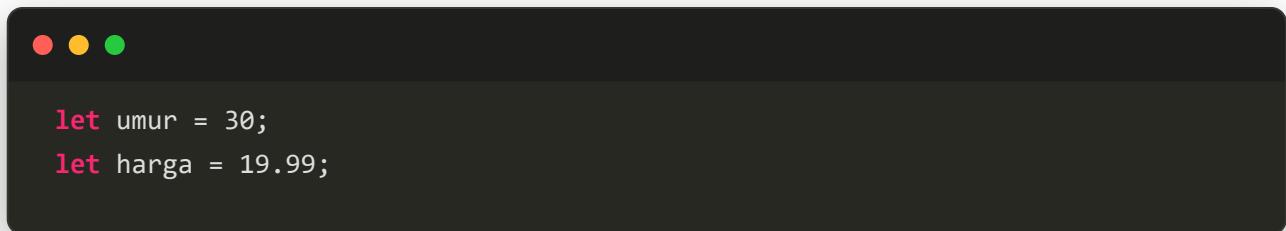
1. Tipe Data Primitif:

- **String** : Merepresentasikan teks. Diapit oleh tanda kutip tunggal ('), ganda ("), atau *backticks* (`) untuk *template literals*.



```
let nama = "Budi";
let salam = 'Halo';
let pesan = `Selamat datang, ${nama}!`; // Template literal
```

- **Number** : Merepresentasikan angka, baik integer maupun floating-point. JavaScript tidak membedakan antara integer dan float.



```
let umur = 30;
let harga = 19.99;
```

- **Boolean** : Merepresentasikan nilai kebenaran: **true** atau **false** .

```
let isStudent = true;  
let hasLicense = false;
```

- **Undefined** : Merepresentasikan variabel yang telah dideklarasikan tetapi belum diberi nilai. Ini adalah nilai default untuk variabel yang tidak diinisialisasi.

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

- **Null** : Merepresentasikan ketiadaan nilai yang disengaja. Ini adalah nilai yang harus secara eksplisit ditetapkan.

```
let mobil = null; // mobil tidak memiliki nilai
```

- **Symbol (ES6)**: Tipe data unik dan immutable yang dapat digunakan sebagai kunci properti objek. Digunakan untuk menghindari konflik nama properti.

```
const id = Symbol('id');  
const user = { [id]: 123 };
```

- **BigInt (ES2020)**: Merepresentasikan bilangan bulat dengan presisi arbitrer. Digunakan untuk angka yang lebih besar dari `Number.MAX_SAFE_INTEGER`.

```
const bigNumber = 123456789012345678901234567890123456789n;
```

2. Tipe Data Non-Primitif (Object):

- **Object** : Koleksi properti yang tidak berurutan, di mana setiap properti memiliki nama (kunci) dan nilai. Array dan fungsi adalah jenis objek khusus.

```
let orang = {  
    nama: "Alice",  
    umur: 25,  
    pekerjaan: "Developer"  
};
```

Variabel

Variabel adalah wadah untuk menyimpan nilai data. Dalam JavaScript, variabel dapat dideklarasikan menggunakan tiga kata kunci:

- **var** : Cara lama untuk mendeklarasikan variabel. Memiliki *function scope* dan dapat di-*hoist*.

```
var x = 10;
```

- **let (ES6)**: Cara modern untuk mendeklarasikan variabel. Memiliki *block scope* dan tidak dapat di-*hoist* ke atas scope-nya.

```
let y = 20;
```

- **const (ES6)**: Digunakan untuk mendeklarasikan konstanta. Memiliki *block scope* dan nilainya tidak dapat diubah setelah diinisialisasi. Namun, untuk objek dan array, properti di dalamnya masih bisa diubah.

```
const PI = 3.14;  
const arr = [1, 2, 3];  
arr.push(4); // Ini boleh  
// PI = 3.14159; // Ini akan error
```

Perbedaan `var`, `let`, dan `const` :

Fitur	<code>var</code>	<code>let</code>	<code>const</code>
Scope	Function scope	Block scope	Block scope
Re-deklarasi	Ya	Tidak	Tidak
Re-assign	Ya	Ya	Tidak
Hoisting	Ya (dengan <code>undefined</code> awal)	Ya (dengan <code>Temporal Dead Zone</code>)	Ya (dengan <code>Temporal Dead Zone</code>)

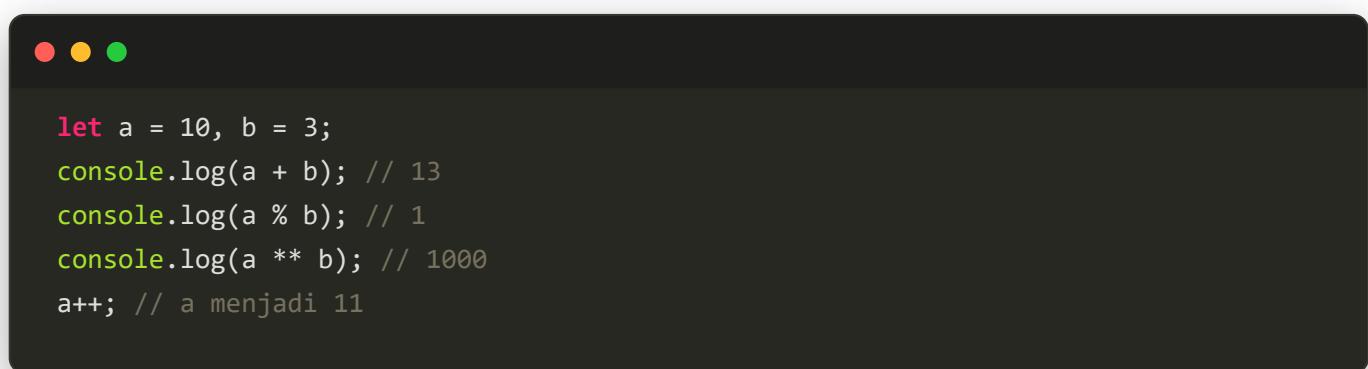
Disarankan untuk selalu menggunakan `let` atau `const` karena mereka menyediakan *scope* yang lebih jelas dan membantu mencegah bug.

Operator

Operator digunakan untuk melakukan operasi pada nilai (operand).

1. Operator Aritmatika:

- `+` (Penjumlahan)
- `-` (Pengurangan)
- `*` (Perkalian)
- `/` (Pembagian)
- `%` (Modulus/Sisa bagi)
- `**` (Eksponensial, ES2016)
- `++` (Increment)
- `--` (Decrement)



```
let a = 10, b = 3;
console.log(a + b); // 13
console.log(a % b); // 1
console.log(a ** b); // 1000
a++; // a menjadi 11
```

2. Operator Penugasan (Assignment Operators):

- `=` (Penugasan sederhana)
- `+=`, `-=`, `*=`, `/=`, `%=`, `**=`



```
let x = 5;  
x += 3; // Sama dengan x = x + 3; x sekarang 8
```

3. Operator Perbandingan (Comparison Operators):

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

- `==` (Sama dengan, membandingkan nilai saja)
- `===` (Sama dengan ketat, membandingkan nilai dan tipe data)
- `!=` (Tidak sama dengan, membandingkan nilai saja)
- `!==` (Tidak sama dengan ketat, membandingkan nilai dan tipe data)
- `>`, `<`, `>=`, `<=`



```
console.log(5 == '5'); // true (nilai sama)  
console.log(5 === '5'); // false (tipe data berbeda)  
console.log(10 > 5); // true
```

4. Operator Logika (Logical Operators):

- `&&` (AND logis): `true` jika kedua operand `true`.
- `||` (OR logis): `true` jika salah satu operand `true`.
- `!` (NOT logis): Membalik nilai boolean.



```
let isRaining = true;  
let isCold = false;  
console.log(isRaining && isCold); // false  
console.log(isRaining || isCold); // true  
console.log(!isRaining); // false
```

5. Operator String:

- `+` (Konkatenasi/Penggabungan string)

```
let firstName = "John";
let lastName = "Doe";
let fullName = firstName + " " + lastName; // "John Doe"
```

6. Operator Ternary (Conditional Operator):

Sintaks: `kondisi ? nilai_jika_true : nilai_jika_false`

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

Kode Contoh

```
// Contoh penggunaan tipe data dan variabel
let namaDepan = "Alice"; // String
let usia = 28;           // Number
let isVerified = true;   // Boolean
let pekerjaan;          // Undefined
let hobi = null;         // Null

console.log("Nama: " + namaDepan + ", Usia: " + usia + ", Verified: " + isVerified);

// Contoh penggunaan operator aritmatika
let num1 = 20;
let num2 = 7;
console.log("Penjumlahan: " + (num1 + num2)); // 27
console.log("Pengurangan: " + (num1 - num2)); // 13
console.log("Perkalian: " + (num1 * num2)); // 140
console.log("Pembagian: " + (num1 / num2)); // 2.857...
console.log("Modulus: " + (num1 % num2)); // 6

// Contoh penggunaan operator perbandingan
let angkaA = 15;
let angkaB = "15";
console.log("angkaA == angkaB: " + (angkaA == angkaB)); // true
console.log("angkaA === angkaB: " + (angkaA === angkaB)); // false
```

```
console.log("angkaA !== angkaB: " + (angkaA !== angkaB)); // true

// Contoh penggunaan operator logika
let punyaKTP = true;
let sudahMenikah = false;
console.log("Bisa memilih: " + (punyaKTP && !sudahMenikah)); // true (jika punya KTP dan belum menikah)
console.log("Bisa masuk klub: " + (punyaKTP || sudahMenikah)); // true (jika punya KTP atau sudah menikah)

// Contoh operator ternary
let nilai = 75;
let grade = (nilai >= 70) ? "Lulus" : "Tidak Lulus";
console.log("Grade: " + grade); // Lulus
```

Langkah-langkah Praktikum

1. Buka editor teks Anda dan buat file baru bernama `pertemuan2.js`.
2. Salin dan tempel kode contoh di atas ke dalam `pertemuan2.js`.
3. Buka terminal atau Command Prompt, navigasikan ke direktori tempat Anda menyimpan `pertemuan2.js`.
4. Jalankan file menggunakan Node.js:

5.

```
node pertemuan2.js
```

6. Amati output di konsol. Coba ubah nilai variabel dan operator untuk melihat bagaimana output berubah.

7. Latihan:

- o Buat variabel `panjang` dan `lebar` dengan nilai numerik. Hitung luas dan keliling persegi panjang menggunakan operator aritmatika dan cetak hasilnya ke konsol.
- o Buat dua variabel string, `kata1` dan `kata2`. Gabungkan keduanya menjadi satu kalimat dan cetak.
- o Buat variabel `suhuCelsius` dengan nilai numerik. Konversikan ke Fahrenheit menggunakan rumus `(Celsius * 9/5) + 32` dan cetak hasilnya.
- o Buat variabel `isMember` (boolean) dan `totalBelanja` (number). Gunakan operator logika dan ternary untuk menentukan apakah pelanggan mendapatkan diskon 10% (jika member ATAU total belanja > 100) dan cetak pesan yang sesuai.

Referensi

-
- [1] MDN Web Docs. (n.d.). *Data structures*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures
 - [2] MDN Web Docs. (n.d.). *Variables*. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var>
 - [3] MDN Web Docs. (n.d.). *Operators*. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators>

Pertemuan 3: Struktur Kontrol (Conditional dan Looping)

Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Menggunakan pernyataan kondisional (`if`, `else if`, `else`, `switch`) untuk mengontrol alur program.
- Menggunakan berbagai jenis perulangan (`for`, `while`, `do...while`, `for...of`, `for...in`) untuk mengulang blok kode.
- Memahami penggunaan `break` dan `continue` dalam perulangan.

Penjelasan Materi

Pernyataan Kondisional

Pernyataan kondisional memungkinkan kita untuk mengeksekusi blok kode tertentu hanya jika suatu kondisi terpenuhi. Ini adalah dasar dari logika pengambilan keputusan dalam pemrograman.

1. `if`, `else if`, `else`

Digunakan untuk mengeksekusi blok kode berdasarkan satu atau lebih kondisi.

```
let nilai = 75;

if (nilai >= 90) {
    console.log("Grade A");
} else if (nilai >= 80) {
    console.log("Grade B");
} else if (nilai >= 70) {
    console.log("Grade C");
} else {
    console.log("Grade D");
}
```

2. `switch`

Digunakan sebagai alternatif untuk banyak `else if` ketika Anda memiliki banyak kondisi yang didasarkan pada nilai tunggal. Pernyataan `break` digunakan untuk keluar dari blok `switch` setelah kasus yang cocok ditemukan. `default` adalah blok yang dieksekusi jika tidak ada kasus yang cocok.

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

```
switch (hari) {
  case "Senin":
    console.log("Hari kerja pertama");
    break;
  case "Sabtu":
  case "Minggu":
    console.log("Akhir pekan");
    break;
  default:
    console.log("Hari kerja");
}
```

Perulangan (Looping)

Perulangan digunakan untuk mengeksekusi blok kode berulang kali selama kondisi tertentu terpenuhi atau untuk setiap item dalam koleksi.

1. `for` loop

Perulangan `for` adalah yang paling umum digunakan ketika Anda tahu berapa kali Anda ingin mengulang. Ini terdiri dari tiga bagian: inisialisasi, kondisi, dan ekspresi iterasi.

```
for (let i = 0; i < 5; i++) {
  console.log("Iterasi ke-" + (i + 1));
}

// Output:
// Iterasi ke-1
// Iterasi ke-2
// Iterasi ke-3
// Iterasi ke-4
// Iterasi ke-5
```

2. `while` loop

Perulangan `while` mengeksekusi blok kode selama kondisi yang ditentukan adalah `true`. Kondisi dievaluasi sebelum setiap iterasi.

```
let count = 0;
while (count < 3) {
```

```
    console.log("Count: " + count);
    count++;
}
// Output:
// Count: 0
// Count: 1
// Count: 2
```

3. do...while loop

Mirip dengan `while` loop, tetapi blok kode dieksekusi setidaknya sekali sebelum kondisi dievaluasi. Ini berarti blok kode akan selalu berjalan minimal satu kali, bahkan jika kondisi awalnya `false`.

```
● ● ●

let i = 0;
do {
    console.log("Do-While Iterasi: " + i);
    i++;
} while (i < 0); // Kondisi false, tapi tetap jalan sekali
// Output:
// Do-While Iterasi: 0
```

4. for...of loop (ES6)

Digunakan untuk mengulang nilai-nilai dari objek yang dapat diulang (seperti Array, String, Map, Set, dll.).

```
● ● ●

const colors = ["red", "green", "blue"];
for (const color of colors) {
    console.log(color);
}
// Output:
// red
// green
// blue
```

5. for...in loop

Digunakan untuk mengulang properti (kunci) dari sebuah objek. Tidak disarankan untuk mengulang array karena urutan properti tidak dijamin dan dapat mengulang properti yang diwarisi.



```
const person = { name: "John", age: 30, city: "New York" };
for (const key in person) {
    console.log(` ${key}: ${person[key]}`);
}
// Output:
// name: John
// age: 30
// city: New York
```

break **dan** **continue**

- **break** : Menghentikan perulangan sepenuhnya dan melanjutkan eksekusi kode setelah perulangan.
- **continue** : Melewatkkan iterasi saat ini dari perulangan dan melanjutkan ke iterasi berikutnya.



```
// Contoh break
for (let i = 0; i < 10; i++) {
    if (i === 5) {
        break; // Perulangan berhenti saat i adalah 5
    }
    console.log("Break: " + i);
}
// Output:
// Break: 0
// Break: 1
// Break: 2
// Break: 3
// Break: 4

// Contoh continue
for (let i = 0; i < 5; i++) {
    if (i === 2) {
        continue; // Melewatkkan iterasi saat i adalah 2
    }
    console.log("Continue: " + i);
}
// Output:
// Continue: 0
// Continue: 1
```

```
// Continue: 3  
// Continue: 4
```

Kode Contoh

```
// Contoh penggunaan if-else if-else  
let suhu = 28;  
  
if (suhu > 30) {  
    console.log("Suhu sangat panas!");  
} else if (suhu > 25) {  
    console.log("Suhu hangat");  
} else {  
    console.log("Suhu dingin");  
}  
  
// Contoh penggunaan switch  
let buah = "apel";  
  
switch (buah) {  
    case "apel":  
        console.log("Ini buah apel.");  
        break;  
    case "pisang":  
        console.log("Ini buah pisang.");  
        break;  
    default:  
        console.log("Buah tidak dikenal.");  
}  
  
// Contoh penggunaan for loop untuk menjumlahkan angka  
let total = 0;  
for (let i = 1; i <= 10; i++) {  
    total += i; // total = total + i  
}  
console.log("Jumlah angka 1 sampai 10 adalah: " + total); // 55  
  
// Contoh penggunaan while loop untuk menghitung mundur  
let hitungMundur = 5;  
while (hitungMundur > 0) {  
    console.log("Hitung mundur: " + hitungMundur);
```

```
hitungMundur--;  
}  
console.log("Mulai!");  
  
// Contoh penggunaan for...of dengan array  
const namaSiswa = ["Andi", "Budi", "Citra"];  
console.log("Daftar Siswa:");  
for (const nama of namaSiswa) {  
    console.log("- " + nama);  
}  
  
// Contoh penggunaan for...in dengan objek  
const buku = { judul: "Filosofi Teras", penulis: "Henry Manampiring", tahun: 2018 };  
console.log("Detail Buku:");  
for (const properti in buku) {  
    console.log(` ${properti}: ${buku[properti]}`);  
}  
  
// Contoh kombinasi loop dan conditional dengan break dan continue  
console.log("Angka genap dari 0 sampai 10 (kecuali 4):");  
for (let i = 0; i <= 10; i++) {  
    if (i % 2 !== 0) {  
        continue; // Lewati jika ganjil  
    }  
    if (i === 4) {  
        continue; // Lewati angka 4  
    }  
    console.log(i);  
}
```

Langkah-langkah Praktikum

1. Buka editor teks Anda dan buat file baru bernama `pertemuan3.js`.
2. Salin dan tempel kode contoh di atas ke dalam `pertemuan3.js`.
3. Buka terminal atau Command Prompt, navigasikan ke direktori tempat Anda menyimpan `pertemuan3.js`.
4. Jalankan file menggunakan Node.js:



```
node pertemuan3.js
```

5. Amati output di konsol. Coba ubah kondisi dan nilai dalam perulangan untuk melihat bagaimana output berubah.

6. Latihan:

- Buat program yang meminta pengguna memasukkan angka (gunakan `prompt()` jika di browser, atau baca input dari terminal jika di Node.js - untuk Node.js, Anda mungkin perlu modul `readline`). Tentukan apakah angka tersebut positif, negatif, atau nol menggunakan `if-else if-else`.
- Buat program yang mencetak semua angka dari 1 hingga 20, tetapi untuk kelipatan 3 cetak "Fizz", untuk kelipatan 5 cetak "Buzz", dan untuk kelipatan 3 dan 5 cetak "FizzBuzz". Gunakan `for` loop dan pernyataan kondisional.
- Buat program yang menghitung faktorial dari sebuah angka (misal: $5! = 5 * 4 * 3 * 2 * 1 = 120$) menggunakan `while` loop.
- Buat array berisi nama-nama buah. Gunakan `for...of` loop untuk mencetak setiap nama buah.

Referensi

[1] MDN Web Docs. (n.d.). *Control flow and error handling*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/control_flow

[2] MDN Web Docs. (n.d.). *Loops and iteration*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

Pertemuan 4: Fungsi dan Scope

Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Mendefinisikan dan memanggil fungsi dalam JavaScript.
- Memahami berbagai jenis fungsi (fungsi deklarasi, ekspresi fungsi, arrow function).
- Menggunakan parameter dan nilai kembalian (return value) pada fungsi.
- Memahami konsep *scope* (global, lokal/function, block) dalam JavaScript.
- Memahami konsep *hoisting* pada fungsi dan variabel.

Penjelasan Materi

Fungsi

Fungsi adalah blok kode yang dirancang untuk melakukan tugas tertentu. Fungsi memungkinkan kita untuk mengorganisir kode, membuatnya lebih mudah dibaca, digunakan kembali, dan di-debug. Fungsi juga membantu menghindari pengulangan kode (DRY - Don't Repeat Yourself).

1. Mendefinisikan Fungsi

Ada beberapa cara untuk mendefinisikan fungsi di JavaScript:

- **Deklarasi Fungsi (Function Declaration):**

Ini adalah cara paling umum untuk mendefinisikan fungsi. Fungsi ini di-*hoist*, artinya dapat dipanggil sebelum didefinisikan dalam kode.

```
function sapa(nama) {  
    return "Halo, " + nama + "!";  
}
```

- **Ekspresi Fungsi (Function Expression):**

Fungsi didefinisikan sebagai bagian dari ekspresi, biasanya dengan menetapkannya ke sebuah variabel. Fungsi ini tidak di-*hoist* seperti deklarasi fungsi.

```
const hitungLuasPersegi = function(sisi) {  
    return sisi * sisi;  
};
```

- **Arrow Function (ES6):**

Sintaks yang lebih ringkas untuk menulis ekspresi fungsi. Sangat berguna untuk fungsi anonim dan memiliki perilaku `this` yang berbeda (akan dibahas di pertemuan lanjutan).

```
const tambah = (a, b) => a + b;

// Dengan satu parameter, tanda kurung bisa dihilangkan
const kuadrat = x => x * x;

// Dengan lebih dari satu baris kode, gunakan kurung kurawal dan return eksplisit
const cetakPesan = (pesan) => {
    console.log("Pesanan diterima:");
    return pesan;
};
```

2. Memanggil Fungsi

Untuk menjalankan kode di dalam fungsi, Anda harus memanggilnya dengan nama fungsi diikuti oleh tanda kurung `()`.

```
console.log(sapa("Budi")); // Output: Halo, Budi!
let luas = hitungLuasPersegi(5);
console.log("Luas persegi: " + luas); // Output: Luas persegi: 25
console.log("Hasil tambah: " + tambah(10, 5)); // Output: Hasil tambah: 15
```

3. Parameter dan Argumen

- **Parameter:** Variabel yang terdaftar di dalam tanda kurung fungsi saat fungsi didefinisikan. Mereka bertindak sebagai *placeholder* untuk nilai yang akan dilewatkan ke fungsi.
- **Argumen:** Nilai aktual yang dilewatkan ke fungsi saat fungsi dipanggil.

```
function greet(firstName, lastName) { // firstName, lastName adalah parameter
    console.log(`Hello, ${firstName} ${lastName}!`);
}
greet("Jane", "Doe"); // "Jane", "Doe" adalah argumen
```

4. Nilai Kembalian (Return Value)

Pernyataan `return` digunakan untuk mengembalikan nilai dari fungsi. Ketika `return` dieksekusi, fungsi berhenti dan nilai yang dikembalikan dikirim kembali ke pemanggil fungsi. Jika tidak ada pernyataan `return` atau `return` tanpa nilai, fungsi akan mengembalikan `undefined` secara default.

```
function kali(x, y) {  
    return x * y;  
}  
  
let hasil = kali(4, 6); // hasil akan menjadi 24  
console.log(hasil);
```

Scope

Scope adalah area di mana variabel dan fungsi dapat diakses. JavaScript memiliki beberapa jenis *scope*:

1. Global Scope

Variabel yang dideklarasikan di luar fungsi atau blok kode apa pun memiliki *global scope*. Mereka dapat diakses dari mana saja dalam program JavaScript.

```
let globalVar = "Saya global";  
  
function showGlobal() {  
    console.log(globalVar); // Bisa diakses  
}  
showGlobal();  
console.log(globalVar); // Bisa diakses
```

2. Function Scope (Local Scope)

Variabel yang dideklarasikan di dalam fungsi (menggunakan `var`) memiliki *function scope*. Mereka hanya dapat diakses di dalam fungsi tersebut.

```
function showLocal() {  
    var localVar = "Saya lokal (function scope)";  
    console.log(localVar);  
}
```

```
showLocal();  
// console.log(localVar); // Error: localVar is not defined
```

3. Block Scope (ES6: `let` dan `const`)

Variabel yang dideklarasikan dengan `let` atau `const` di dalam blok kode (misalnya, di dalam `if`, `for`, atau `while` loop, atau hanya sepasang kurung kurawal `{}`) memiliki *block scope*. Mereka hanya dapat diakses di dalam blok tersebut.

```
if (true) {  
    let blockVar = "Saya lokal (block scope)";  
    const anotherBlockVar = "Saya juga lokal (block scope)";  
    console.log(blockVar);  
    console.log(anotherBlockVar);  
}  
// console.log(blockVar); // Error: blockVar is not defined
```

Penting: Selalu gunakan `let` atau `const` untuk deklarasi variabel untuk memanfaatkan *block scope* dan menghindari masalah yang terkait dengan *hoisting* `var` dan *global scope* yang tidak disengaja.

Hoisting

Hoisting adalah perilaku JavaScript di mana deklarasi variabel dan fungsi dipindahkan secara konseptual ke bagian atas *scope* mereka sebelum kode dieksekusi. Namun, hanya deklarasi yang di-*hoist*, bukan inisialisasinya.

- **Variabel `var`** : Deklarasi di-*hoist* dan diinisialisasi dengan `undefined`.

```
console.log(myVar); // Output: undefined  
var myVar = 10;  
console.log(myVar); // Output: 10
```

- **Variabel `let` dan `const`** : Deklarasi di-*hoist*, tetapi tidak diinisialisasi. Mereka berada dalam "Temporal Dead Zone" (TDZ) sampai baris deklarasi dieksekusi. Mengaksesnya sebelum deklarasi akan menyebabkan `ReferenceError`.

```
// console.log(myLet); // ReferenceError: Cannot access 'myLet' before initiali
let myLet = 20;
```

- **Deklarasi Fungsi:** Seluruh fungsi di-*hoist*, sehingga dapat dipanggil sebelum didefinisikan.

```
sayHello(); // Output: Hello!
function sayHello() {
    console.log("Hello!");
}
```

- **Ekspresi Fungsi:** Hanya deklarasi variabelnya yang di-*hoist*, bukan fungsi itu sendiri.

```
// sayHi(); // TypeError: sayHi is not a function (karena sayHi adalah undefined
const sayHi = function() {
    console.log("Hi!");
};
sayHi(); // Output: Hi!
```

Kode Contoh

```
// Deklarasi Fungsi
function hitungRataRata(a, b, c) {
    return (a + b + c) / 3;
}

let rata = hitungRataRata(10, 20, 30);
console.log("Rata-rata: " + rata); // Output: Rata-rata: 20

// Ekspresi Fungsi
const cekGenapGanjil = function(angka) {
    if (angka % 2 === 0) {
        return "Genap";
```

```
    } else {
        return "Ganjil";
    }
};

console.log("Angka 7 adalah " + cekGenapGanjil(7)); // Output: Angka 7 adalah Ganjil

// Arrow Function
const kaliDua = num => num * 2;
console.log("5 dikali dua adalah " + kaliDua(5)); // Output: 5 dikali dua adalah 10

const sapaPengguna = (nama, waktu) => {
    return `Selamat ${waktu}, ${nama}!`;
};
console.log(sapaPengguna("Dian", "pagi")); // Output: Selamat pagi, Dian!

// Contoh Scope
let namaGlobal = "Global User";

function tampilkanNama() {
    let namaLokal = "Lokal User";
    console.log("Dari dalam fungsi (lokal): " + namaLokal); // Bisa diakses
    console.log("Dari dalam fungsi (global): " + namaGlobal); // Bisa diakses

    if (true) {
        let namaBlok = "Blok User";
        console.log("Dari dalam blok (blok): " + namaBlok); // Bisa diakses
    }
    // console.log(namaBlok); // Error: namaBlok is not defined
}

tampilkanNama();
console.log("Dari luar fungsi (global): " + namaGlobal); // Bisa diakses
// console.log(namaLokal); // Error: namaLokal is not defined

// Contoh Hoisting
console.log(hoistedVar); // undefined
var hoistedVar = "Saya dihoist";
console.log(hoistedVar);

// console.log(hoistedLet); // ReferenceError
// let hoistedLet = "Saya tidak dihoist";

hoistedFunction(); // Fungsi ini bisa dipanggil karena dihoist
```

```
function hoistedFunction() {  
    console.log("Saya adalah fungsi yang dihoist.");  
}  
  
// notHoistedFunction(); // TypeError: notHoistedFunction is not a function  
const notHoistedFunction = function() {  
    console.log("Saya adalah ekspresi fungsi, tidak dihoist.");  
};  
notHoistedFunction();
```

Langkah-langkah Praktikum

1. Buka editor teks Anda dan buat file baru bernama `pertemuan4.js`.
2. Salin dan tempel kode contoh di atas ke dalam `pertemuan4.js`.
3. Buka terminal atau Command Prompt, navigasikan ke direktori tempat Anda menyimpan `pertemuan4.js`.
4. Jalankan file menggunakan Node.js:



```
node pertemuan4.js
```

5. Amati output di konsol. Coba buat fungsi Anda sendiri dengan parameter dan nilai kembalian yang berbeda.

6. Latihan:

- Buat fungsi bernama `cekSuhu` yang menerima satu parameter `suhu`. Fungsi ini harus mengembalikan string "Panas" jika suhu di atas 30, "Normal" jika antara 20 dan 30 (inklusif), dan "Dingin" jika di bawah 20.
- Buat fungsi `hitungDiskon` yang menerima `hargaAwal` dan `persenDiskon`. Fungsi ini harus mengembalikan harga setelah diskon. Pastikan `persenDiskon` adalah angka antara 0 dan 100.
- Buat sebuah *arrow function* bernama `isPalindrome` yang menerima sebuah string. Fungsi ini harus mengembalikan `true` jika string tersebut adalah palindrom (dibaca sama dari depan dan belakang, abaikan besar kecil huruf dan spasi), dan `false` jika tidak. Contoh: "Kasur Rusak" adalah palindrom.
- Jelaskan dalam komentar di kode Anda, mengapa variabel yang dideklarasikan dengan `var` di dalam `if` statement masih bisa diakses di luar `if` statement, sedangkan `let` atau `const` tidak bisa.

Referensi

-
- [1] MDN Web Docs. (n.d.). *Functions*. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions>
 - [2] MDN Web Docs. (n.d.). *Scope*. Retrieved from <https://developer.mozilla.org/en-US/docs/Glossary/Scope>
 - [3] MDN Web Docs. (n.d.). *Hoisting*. Retrieved from <https://developer.mozilla.org/en-US/docs/Glossary/Hoisting>

Pertemuan 5: Array dan Object

Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Memahami konsep Array dan Object sebagai struktur data fundamental dalam JavaScript.
- Membuat, mengakses, dan memodifikasi elemen dalam Array.
- Menggunakan metode-metode Array yang umum (push, pop, shift, unshift, splice, slice, forEach, map, filter, reduce).
- Membuat, mengakses, dan memodifikasi properti dalam Object.
- Memahami perbedaan antara Array dan Object serta kapan menggunakan.

Penjelasan Materi

Array

Array adalah struktur data yang digunakan untuk menyimpan koleksi item yang berurutan. Setiap item dalam array memiliki indeks numerik, dimulai dari 0.

1. Membuat Array

- **Array Literal (Cara Paling Umum):**

```
let buah = ["Apel", "Pisang", "Ceri"];
let angka = [1, 2, 3, 4, 5];
let campuran = ["String", 123, true, null];
```

- **Konstruktor `Array()` :**

```
let warna = new Array("Merah", "Hijau", "Biru");
let ukuran = new Array(5); // Membuat array kosong dengan 5 slot
```

2. Mengakses Elemen Array

Elemen diakses menggunakan indeksnya (dimulai dari 0).

```
let buah = ["Apel", "Pisang", "Ceri"];
```

```
console.log(buah[0]); // Output: Apel
console.log(buah[1]); // Output: Pisang
console.log(buah[2]); // Output: Ceri
console.log(buah.length); // Output: 3 (jumlah elemen)
```

3. Memodifikasi Elemen Array

Anda dapat mengubah nilai elemen dengan menugaskan nilai baru ke indeks tertentu.

```
let buah = ["Apel", "Pisang", "Ceri"];
buah[1] = "Mangga";
console.log(buah); // Output: ["Apel", "Mangga", "Ceri"]
```

4. Metode Array Umum

- **push()** : Menambahkan satu atau lebih elemen ke akhir array dan mengembalikan panjang array baru.

```
let angka = [1, 2];
angka.push(3, 4);
console.log(angka); // Output: [1, 2, 3, 4]
```

- **pop()** : Menghapus elemen terakhir dari array dan mengembalikan elemen yang dihapus.

```
let angka = [1, 2, 3];
let last = angka.pop();
console.log(angka); // Output: [1, 2]
console.log(last); // Output: 3
```

- **shift()** : Menghapus elemen pertama dari array dan mengembalikan elemen yang dihapus.

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

```
let first = angka.shift();
console.log(angka); // Output: [2, 3]
console.log(first); // Output: 1
```

- **unshift()** : Menambahkan satu atau lebih elemen ke awal array dan mengembalikan panjang array baru.

```
let angka = [3, 4];
angka.unshift(1, 2);
console.log(angka); // Output: [1, 2, 3, 4]
```

- **splice(start, deleteCount, item1, ...)** : Mengubah konten array dengan menghapus elemen yang ada dan/atau menambahkan elemen baru.

```
let buah = ["Apel", "Pisang", "Ceri", "Durian"];
buah.splice(1, 2, "Mangga", "Anggur"); // Hapus 2 elemen mulai dari indeks 1, lalu tambahkan "Mangga" dan "Anggur"
console.log(buah); // Output: ["Apel", "Mangga", "Anggur", "Durian"]

buah.splice(2, 1); // Hapus 1 elemen di indeks 2
console.log(buah); // Output: ["Apel", "Mangga", "Durian"]
```

- **slice(start, end)** : Mengembalikan salinan dangkal (shallow copy) dari sebagian array ke dalam array baru. Tidak memodifikasi array asli.

```
let angka = [1, 2, 3, 4, 5];
let subArray = angka.slice(1, 4); // Dari indeks 1 (inklusif) sampai 4 (eksklusif)
console.log(subArray); // Output: [2, 3, 4]
console.log(angka); // Output: [1, 2, 3, 4, 5] (array asli tidak berubah)
```

- **forEach(callback)** : Mengeksekusi fungsi *callback* sekali untuk setiap elemen array.

```
let nama = ["Alice", "Bob", "Charlie"];
nama.forEach(function(item, index) {
    console.log(`Indeks ${index}: ${item}`);
});
// Output:
// Indeks 0: Alice
// Indeks 1: Bob
// Indeks 2: Charlie
```

- `map(callback)` : Membuat array baru dengan hasil pemanggilan fungsi *callback* pada setiap elemen array.

```
let angka = [1, 2, 3];
let kuadrat = angka.map(function(num) {
    return num * num;
});
console.log(kuadrat); // Output: [1, 4, 9]
```

- `filter(callback)` : Membuat array baru dengan semua elemen yang lulus uji yang diimplementasikan oleh fungsi *callback* yang disediakan.

```
let angka = [1, 2, 3, 4, 5, 6];
let genap = angka.filter(function(num) {
    return num % 2 === 0;
});
console.log(genap); // Output: [2, 4, 6]
```

- `reduce(callback, initialValue)` : Mengeksekusi fungsi *reducer* (yang Anda sediakan) pada setiap elemen array, menghasilkan satu nilai output tunggal.

```
let angka = [1, 2, 3, 4];
let sum = angka.reduce(function(accumulator, currentValue) {
```

```
        return accumulator + currentValue;
}, 0); // 0 adalah initialValue untuk accumulator
console.log(sum); // Output: 10
```

Object

Object adalah koleksi properti yang tidak berurutan, di mana setiap properti terdiri dari pasangan kunci (key) dan nilai (value). Kunci properti biasanya adalah string (atau Symbol), dan nilainya bisa berupa tipe data apa pun (termasuk array, fungsi, atau objek lain).

1. Membuat Object

- **Object Literal (Cara Paling Umum):**

```
let orang = {
    nama: "Budi",
    umur: 30,
    pekerjaan: "Engineer",
    hobi: ["Membaca", "Coding"]
};
```

- **Konstruktor `Object()` :**

```
let mobil = new Object();
mobil.merk = "Toyota";
mobil.model = "Camry";
```

2. Mengakses Properti Object

- **Dot Notation (Paling Umum):**

```
console.log(orang.nama);      // Output: Budi
console.log(orang.pekerjaan); // Output: Engineer
```

- **Bracket Notation (Berguna untuk kunci dinamis atau kunci dengan spasi/karakter khusus):**

```
● ● ●  
console.log(orang["umur"]); // Output: 30  
let key = "pekerjaan";  
console.log(orang[key]); // Output: Engineer  
  
let data = { "nama lengkap": "Siti Aminah" };  
console.log(data["nama lengkap"]); // Output: Siti Aminah
```

3. Memodifikasi Properti Object

Anda dapat mengubah nilai properti atau menambahkan properti baru.

```
● ● ●  
let orang = {  
    nama: "Budi",  
    umur: 30  
};  
orang.umur = 31; // Mengubah nilai properti  
orang.kota = "Jakarta"; // Menambahkan properti baru  
console.log(orang); // Output: { nama: "Budi", umur: 31, kota: "Jakarta" }
```

4. Menghapus Properti Object

Gunakan operator `delete`.

```
● ● ●  
let orang = {  
    nama: "Budi",  
    umur: 30  
};  
delete orang.umur;  
console.log(orang); // Output: { nama: "Budi" }
```

5. Iterasi Properti Object

Gunakan `for...in` loop (seperti yang dibahas di Pertemuan 3) atau metode `Object.keys()`, `Object.values()`, `Object.entries()`.

```
const produk = { nama: "Laptop", harga: 1200, stok: 50 };

// Menggunakan Object.keys()
Object.keys(produk).forEach(key => {
    console.log(`Kunci: ${key}`);
});

// Menggunakan Object.values()
Object.values(produk).forEach(value => {
    console.log(`Nilai: ${value}`);
});

// Menggunakan Object.entries()
Object.entries(produk).forEach(([key, value]) => {
    console.log(`${key}: ${value}`);
});
```

Kode Contoh

```
// Contoh Array
let daftarBelanja = ["Susu", "Roti", "Telur", "Buah"];
console.log("Daftar Belanja Awal:", daftarBelanja);

// Menambahkan item
daftarBelanja.push("Kopi");
console.log("Setelah push Kopi:", daftarBelanja);

// Menghapus item terakhir
let itemTerakhir = daftarBelanja.pop();
console.log("Item terakhir dihapus:", itemTerakhir);
console.log("Daftar Belanja Setelah pop:", daftarBelanja);

// Menambahkan item di awal
daftarBelanja.unshift("Gula");
console.log("Setelah unshift Gula:", daftarBelanja);

// Menghapus item pertama
let itemPertama = daftarBelanja.shift();
console.log("Item pertama dihapus:", itemPertama);
```

```
console.log("Daftar Belanja Setelah shift:", daftarBelanja);

// Menggunakan map untuk membuat array baru
let harga = [10000, 15000, 20000];
let hargaDiskon = harga.map(h => h * 0.9); // Diskon 10%
console.log("Harga Diskon:", hargaDiskon);

// Menggunakan filter untuk memilih item
let angka = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let angkaGanjil = angka.filter(num => num % 2 !== 0);
console.log("Angka Ganjil:", angkaGanjil);

// Menggunakan reduce untuk menjumlahkan
let totalHarga = harga.reduce((sum, current) => sum + current, 0);
console.log("Total Harga:", totalHarga);

console.log("\n=====\\n");

// Contoh Object
let buku = {
    judul: "Atomic Habits",
    penulis: "James Clear",
    tahunTerbit: 2018,
    genre: ["Self-help", "Productivity"],
    penerbit: {
        nama: "Gramedia Pustaka Utama",
        kota: "Jakarta"
    },
    isBestSeller: true
};

console.log("Informasi Buku:", buku);

// Mengakses properti
console.log("Judul Buku:", buku.judul);
console.log("Penulis:", buku["penulis"]);
console.log("Genre Pertama:", buku.genre[0]);
console.log("Kota Penerbit:", buku.penerbit.kota);

// Memodifikasi properti
buku.tahunTerbit = 2019;
console.log("Tahun Terbit Baru:", buku.tahunTerbit);

// Menambahkan properti baru
```

```
buku.jumlahHalaman = 320;  
console.log("Buku dengan jumlah halaman:", buku);  
  
// Menghapus properti  
delete buku.isBestSeller;  
console.log("Buku setelah isBestSeller dihapus:", buku);  
  
// Iterasi properti objek  
console.log("\nProperti Buku:");  
for (let key in buku) {  
    if (buku.hasOwnProperty(key)) { // Pastikan properti milik objek itu sendiri  
        console.log(`\$${key}: ${buku[key]}`);  
    }  
}
```

Langkah-langkah Praktikum

1. Buka editor teks Anda dan buat file baru bernama `pertemuan5.js`.
2. Salin dan tempel kode contoh di atas ke dalam `pertemuan5.js`.
3. Buka terminal atau Command Prompt, navigasikan ke direktori tempat Anda menyimpan `pertemuan5.js`.
4. Jalankan file menggunakan Node.js:



```
node pertemuan5.js
```

5. Amati output di konsol. Coba buat array dan objek Anda sendiri dengan data yang berbeda, lalu coba akses dan modifikasi propertiya.

6. Latihan:

- Buat sebuah array bernama `mahasiswa` yang berisi beberapa objek. Setiap objek mahasiswa harus memiliki properti `nama`, `nim`, dan `nilai` (berupa array angka).
- Gunakan metode `forEach` untuk mencetak nama dan NIM setiap mahasiswa.
- Gunakan metode `map` untuk membuat array baru yang berisi hanya nama-nama mahasiswa.
- Gunakan metode `filter` untuk mendapatkan array mahasiswa yang memiliki nilai rata-rata di atas 75.
- Gunakan metode `reduce` untuk menghitung total nilai rata-rata dari semua mahasiswa.
- Buat sebuah objek `perpustakaan` yang memiliki properti `nama` dan `daftarBuku` (berupa array objek buku). Setiap objek buku memiliki `judul`, `penulis`, dan `tahunTerbit`.
- Tambahkan buku baru ke `daftarBuku`.

- Cetak semua judul buku yang terbit sebelum tahun 2000.

Referensi

[1] MDN Web Docs. (n.d.). *Array*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

[2] MDN Web Docs. (n.d.). *Object*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object

Pertemuan 6: DOM Manipulation

Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Memahami apa itu DOM (Document Object Model) dan perannya dalam pengembangan web.
- Mengakses elemen HTML menggunakan JavaScript.
- Memodifikasi konten, atribut, dan gaya (style) elemen HTML.
- Menambah dan menghapus elemen HTML secara dinamis.
- Memahami konsep *event* dan *event handling* dasar.

Penjelasan Materi

Apa itu DOM?

DOM (Document Object Model) adalah antarmuka pemrograman untuk dokumen HTML dan XML. Ini merepresentasikan halaman web sebagai struktur pohon (tree structure) di mana setiap node adalah bagian dari dokumen (elemen, atribut, teks, dll.). JavaScript dapat menggunakan DOM untuk mengakses dan memanipulasi halaman web, mengubah struktur, gaya, dan kontennya secara dinamis setelah halaman dimuat di browser. [1]

Bayangkan DOM sebagai peta interaktif dari halaman web Anda. Setiap bagian dari halaman (paragraf, gambar, tombol) adalah sebuah "lokasi" di peta tersebut, dan JavaScript adalah "navigator" yang dapat menemukan lokasi tersebut, mengubahnya, atau bahkan menambahkan lokasi baru.

Mengakses Elemen HTML

Untuk memanipulasi elemen, pertama kita harus "menemukan" atau "mengakses" elemen tersebut. Ada beberapa metode yang umum digunakan:

1. `document.getElementById()`

Mengakses elemen berdasarkan atribut `id` uniknya. Mengembalikan satu elemen atau `null` jika tidak ditemukan.



```
const myElement = document.getElementById("myId");
```

2. `document.getElementsByClassName()`

Mengakses elemen berdasarkan atribut `class` mereka. Mengembalikan `HTMLCollection` (mirip array) dari elemen-elemen yang cocok.



```
const myElements = document.getElementsByClassName("myClass");
```

3. `document.getElementsByTagName()`

Mengakses elemen berdasarkan nama tag HTML mereka (misalnya, `p`, `div`, `a`). Mengembalikan `HTMLCollection`.



```
const paragraphs = document.getElementsByTagName("p");
```

4. `document.querySelector()` (ES6)

Mengakses elemen pertama yang cocok dengan selektor CSS yang ditentukan. Sangat fleksibel dan direkomendasikan untuk sebagian besar kasus.



```
const firstParagraph = document.querySelector("p");
const myButton = document.querySelector("#myButton"); // Menggunakan ID
const firstItem = document.querySelector(".list-item"); // Menggunakan Class
```

5. `document.querySelectorAll()` (ES6)

Mengakses semua elemen yang cocok dengan selektor CSS yang ditentukan. Mengembalikan `NodeList` (mirip array) dari elemen-elemen yang cocok.



```
const allParagraphs = document.querySelectorAll("p");
const allListItems = document.querySelectorAll(".list-item");
```

Memodifikasi Elemen HTML

Setelah elemen diakses, kita bisa memodifikasi berbagai aspeknya:

1. Mengubah Konten Teks (`textContent` atau `innerText`)

- `textContent` : Mengatur atau mendapatkan konten teks dari elemen dan semua turunannya. Lebih disukai karena lebih konsisten.

- `innerText` : Mirip dengan `textContent`, tetapi memperhitungkan gaya CSS (misalnya, tidak akan mengembalikan teks dari elemen yang tersembunyi).



```
const heading = document.getElementById("myHeading");
heading.textContent = "Judul Baru!";
```

2. Mengubah Konten HTML (`innerHTML`)

Mengatur atau mendapatkan konten HTML dari elemen. Hati-hati saat menggunakan ini dengan input pengguna karena dapat menyebabkan serangan XSS (Cross-Site Scripting).



```
const container = document.getElementById("myContainer");
container.innerHTML = "<p>Ini adalah <strong>paragraf baru</strong>.</p>";
```

3. Mengubah Atribut (`setAttribute`, `getAttribute`, `removeAttribute`)

- `setAttribute(name, value)` : Menetapkan nilai atribut.
- `getAttribute(name)` : Mendapatkan nilai atribut.
- `removeAttribute(name)` : Menghapus atribut.



```
const myImage = document.getElementById("myImage");
myImage.setAttribute("src", "new_image.jpg");
myImage.setAttribute("alt", "Gambar Baru");
console.log(myImage.getAttribute("src"));
myImage.removeAttribute("alt");
```

4. Mengubah Gaya (Style) CSS

Mengakses properti gaya CSS melalui properti `style` dari elemen. Properti CSS yang memiliki tanda hubung (misalnya, `background-color`) diubah menjadi *camelCase* (misalnya, `backgroundColor`).



```
const myDiv = document.getElementById("myDiv");
myDiv.style.backgroundColor = "lightblue";
```

```
myDiv.style.color = "darkblue";
myDiv.style.fontSize = "20px";
```

5. Mengelola Kelas CSS (`classList`)

Properti `classList` menyediakan cara yang mudah untuk menambah, menghapus, atau mengganti kelas CSS pada elemen.

- `add()` : Menambahkan satu atau lebih kelas.
- `remove()` : Menghapus satu atau lebih kelas.
- `toggle()` : Menambahkan kelas jika tidak ada, menghapus jika ada.
- `contains()` : Memeriksa apakah elemen memiliki kelas tertentu.

```
const myButton = document.getElementById("myButton");
myButton.classList.add("active", "highlight");
myButton.classList.remove("active");
myButton.classList.toggle("hidden"); // Menambah jika tidak ada, menghapus jika ada
if (myButton.classList.contains("highlight")) {
    console.log("Tombol memiliki kelas highlight.");
}
```

Menambah dan Menghapus Elemen

JavaScript memungkinkan kita untuk membuat elemen HTML baru dan menambahkannya ke DOM, atau menghapus elemen yang sudah ada.

1. Membuat Elemen Baru (`document.createElement()`)

```
const newParagraph = document.createElement("p");
newParagraph.textContent = "Ini adalah paragraf yang dibuat secara dinamis.;"
```

2. Menambahkan Elemen ke DOM

- `appendChild()` : Menambahkan elemen sebagai anak terakhir dari elemen induk.
- `prepend()` : Menambahkan elemen sebagai anak pertama dari elemen induk.
- `insertBefore(newNode, referenceNode)` : Menambahkan elemen baru sebelum elemen referensi.

```
const container = document.getElementById("container");
container.appendChild(newParagraph);

const firstChild = container.firstChild;
const newDiv = document.createElement("div");
newDiv.textContent = "Ini div baru di awal.";
container.insertBefore(newDiv, firstChild);
```

3. Menghapus Elemen dari DOM (`removeChild()` , `remove()`)

- `removeChild(childNode)` : Dijalankan pada elemen induk untuk menghapus anak tertentu.
- `remove()` : Dijalankan pada elemen itu sendiri untuk menghapus dirinya dari DOM (lebih modern).

```
const elementToRemove = document.getElementById("oldElement");
elementToRemove.parentNode.removeChild(elementToRemove); // Cara lama
// ATAU
elementToRemove.remove(); // Cara baru
```

Event Handling Dasar

Event adalah tindakan atau kejadian yang terjadi di halaman web (misalnya, klik mouse, penekanan tombol, halaman dimuat). *Event handling* adalah cara JavaScript merespons *event* ini.

1. Menggunakan Properti `on` (Kurang Direkomendasikan)

```
<button onclick="alert('Tombol diklik!')>Klik Saya</button>
```

Atau di JavaScript:

```
const myButton = document.getElementById("myButton");
myButton.onclick = function() {
  alert("Tombol diklik!");
};
```

2. Menggunakan `addEventListener()` (Paling Direkomendasikan)

Metode ini memungkinkan Anda melampirkan beberapa *event handler* ke satu elemen dan memberikan kontrol yang lebih baik. Sintaks: `element.addEventListener(event, function, useCapture)`.

```
const myButton = document.getElementById("myButton");
myButton.addEventListener("click", function() {
    console.log("Tombol diklik!");
});

// Anda bisa menambahkan event listener lain untuk event yang sama
myButton.addEventListener("click", () => {
    myButton.style.backgroundColor = "red";
});
```

Kode Contoh

Buat file `index.html` dan `script.js` di folder yang sama.

`index.html` :

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DOM Manipulation Example</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        #main-heading {
            color: navy;
        }
        .highlight {
            background-color: yellow;
            padding: 5px;
        }
    </style>
</head>
<body>
    <h1 id="main-heading">DOM Manipulation Example</h1>
    <p>Ini adalah contoh manipulasi DOM. Klik tombol di bawah ini untuk melihat perubahan pada halaman.</p>
    <button id="highlight-me">Klik Saya!</button>
</body>
</html>
```

```
.box {  
    border: 1px solid black;  
    padding: 10px;  
    margin-top: 10px;  
}  
</style>  
</head>  
<body>  
    <h1 id="main-heading">Selamat Datang di DOM Manipulation!</h1>  
    <p class="intro-text">Ini adalah paragraf pengantar.</p>  
    <p class="intro-text">Ini paragraf kedua.</p>  
  
    <button id="changeTextBtn">Ubah Judul</button>  
    <button id="addRemoveClassBtn">Toggle Highlight</button>  
    <button id="createAddElementBtn">Tambah Elemen Baru</button>  
    <button id="removeElementBtn">Hapus Elemen Terakhir</button>  
  
    <div id="container" class="box">  
        <h2>Kontainer Elemen</h2>  
        <p id="first-paragraph">Paragraf pertama di dalam kontainer.</p>  
        <p>Paragraf kedua di dalam kontainer.</p>  
    </div>  
  
    <script src="script.js"></script>  
</body>  
</html>
```

script.js :

```
// 1. Mengakses dan Memodifikasi Konten Teks  
const mainHeading = document.getElementById("main-heading");  
const changeTextBtn = document.getElementById("changeTextBtn");  
  
changeTextBtn.addEventListener("click", function() {  
    mainHeading.textContent = "Judul Berhasil Diubah oleh JavaScript!";  
    mainHeading.style.color = "red";  
});  
  
// 2. Mengakses dan Mengelola Kelas CSS  
const introParagraphs = document.querySelectorAll(".intro-text");  
const addRemoveClassBtn = document.getElementById("addRemoveClassBtn");
```

```
addRemoveClassBtn.addEventListener("click", function() {
    introParagraphs.forEach(p => {
        p.classList.toggle("highlight");
    });
});

// 3. Menambah Elemen Baru
const container = document.getElementById("container");
const createAddElementBtn = document.getElementById("createAddElementBtn");
let counter = 1;

createAddElementBtn.addEventListener("click", function() {
    const newDiv = document.createElement("div");
    newDiv.textContent = `Elemen baru ke-${counter++} ditambahkan!`;
    newDiv.style.backgroundColor = "#e0ffe0";
    newDiv.style.padding = "5px";
    newDiv.style.marginTop = "5px";
    container.appendChild(newDiv);
});

// 4. Menghapus Elemen
const removeElementBtn = document.getElementById("removeElementBtn");

removeElementBtn.addEventListener("click", function() {
    const allDivsInContainer = container.querySelectorAll("div");
    if (allDivsInContainer.length > 0) {
        // Hapus div terakhir yang ditambahkan
        allDivsInContainer[allDivsInContainer.length - 1].remove();
    } else {
        // Jika tidak ada div, coba hapus paragraf terakhir
        const allParagraphsInContainer = container.querySelectorAll("p");
        if (allParagraphsInContainer.length > 0) {
            allParagraphsInContainer[allParagraphsInContainer.length - 1].remove();
        } else {
            alert("Tidak ada elemen lagi untuk dihapus di kontainer!");
        }
    }
});

// Contoh lain: Mengubah atribut gambar (jika ada gambar di HTML)
// const myImage = document.querySelector("img");
// if (myImage) {
//     myImage.setAttribute("src", "https://via.placeholder.com/150");
// }
```

```
//     myImage.setAttribute("alt", "Placeholder Image");
// }
```

Langkah-langkah Praktikum

1. Buat folder baru untuk pertemuan ini (misalnya, `pertemuan6`).
2. Di dalam folder tersebut, buat dua file: `index.html` dan `script.js`.
3. Salin dan tempel kode `index.html` dan `script.js` di atas ke file masing-masing.
4. Buka file `index.html` di browser web Anda.
5. Buka Developer Tools (`F12`) dan perhatikan tab "Elements" dan "Console".
6. Klik tombol-tombol yang ada di halaman dan amati perubahan yang terjadi pada halaman dan di konsol.

7. Latihan:

- Tambahkan sebuah input teks dan sebuah tombol "Submit" ke `index.html`.
- Ketika tombol "Submit" diklik, ambil nilai dari input teks dan tampilkan di sebuah elemen `div` baru di bawahnya.
- Tambahkan tombol "Reset" yang akan mengosongkan input teks dan menghapus semua elemen yang ditambahkan secara dinamis.
- Buat sebuah daftar (``) dengan beberapa item (``). Tambahkan tombol yang ketika diklik akan mengubah warna latar belakang setiap item daftar secara bergantian (misalnya, merah, biru, merah, biru).
- Coba tambahkan atau hapus atribut pada elemen yang sudah ada (misalnya, tambahkan atribut `disabled` pada tombol setelah diklik sekali).

Referensi

- [1] MDN Web Docs. (n.d.). *Introduction to the DOM*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
- [2] MDN Web Docs. (n.d.). *Manipulating documents*. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents

Pertemuan 7: Event Handling dan Form Validation

Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Memahami konsep *event* dan *event flow* dalam JavaScript.
- Mendaftarkan *event listener* untuk berbagai jenis *event*.
- Menggunakan objek *Event* untuk mendapatkan informasi tentang *event*.
- Mencegah perilaku default browser.
- Melakukan validasi formulir dasar menggunakan JavaScript.

Penjelasan Materi

Event Handling

Event adalah tindakan atau kejadian yang terjadi di halaman web yang dapat dideteksi oleh JavaScript. Contoh *event* meliputi klik mouse, penekanan tombol keyboard, pengiriman formulir, pemuatan halaman, dan banyak lagi. *Event handling* adalah proses di mana JavaScript merespons *event* ini dengan menjalankan fungsi atau blok kode tertentu.

1. Event Flow (Event Bubbling dan Event Capturing)

Ketika sebuah *event* terjadi pada elemen DOM, *event* tersebut tidak hanya terjadi pada elemen itu sendiri, tetapi juga "mengalir" melalui hierarki DOM. Ada dua fase utama dalam *event flow*:

- **Capturing Phase:** *Event* dimulai dari elemen teratas (misalnya, `window` atau `document`) dan bergerak ke bawah menuju elemen target.
 - **Bubbling Phase:** Setelah mencapai elemen target, *event* "menggelembung" kembali ke atas melalui elemen induk hingga mencapai elemen teratas lagi.
- Secara default, `addEventListener()` menggunakan fase *bubbling*. Anda dapat mengubahnya ke fase *capturing* dengan menambahkan argumen ketiga `true`.

2. Mendaftarkan Event Listener

Cara paling direkomendasikan untuk mendaftarkan *event listener* adalah menggunakan

`addEventListener()`.



```
element.addEventListener(event, handler, [options]);
```

- `event` : Nama *event* yang akan didengarkan (misalnya, `'click'`, `'mouseover'`, `'submit'`).
- `handler` : Fungsi yang akan dieksekusi ketika *event* terjadi.

- `options` (opsional): Objek dengan properti seperti `capture` (boolean, default `false`), `once` (boolean, default `false`, *listener* hanya dipanggil sekali), `passive` (boolean, default `false`), untuk *performance* pada *scrolling*).

Contoh:

```
const myButton = document.getElementById("myButton");

// Fungsi anonim sebagai handler
myButton.addEventListener("click", function() {
    alert("Tombol diklik!");
});

// Fungsi bernama sebagai handler
function handleButtonClick() {
    console.log("Tombol telah diklik.");
}
myButton.addEventListener("click", handleButtonClick);

// Arrow function sebagai handler
myButton.addEventListener("mouseover", () => {
    myButton.style.backgroundColor = "lightblue";
});
myButton.addEventListener("mouseout", () => {
    myButton.style.backgroundColor = ""; // Kembali ke warna semula
});
```

3. Objek Event (`event object`)

Ketika sebuah *event* terjadi, fungsi *handler* secara otomatis menerima objek `Event` sebagai argumen pertamanya. Objek ini berisi informasi berguna tentang *event* yang terjadi.

Beberapa properti penting dari objek `Event` :

- `event.target` : Elemen DOM tempat *event* sebenarnya terjadi.
- `event.currentTarget` : Elemen DOM tempat *event listener* terpasang.
- `event.type` : Jenis *event* yang terjadi (misalnya, `'click'`, `'keydown'`).
- `event.preventDefault()` : Mencegah tindakan default browser untuk *event* tersebut (misalnya, mencegah pengiriman formulir atau navigasi link).
- `event.stopPropagation()` : Menghentikan *event* dari *bubbling* atau *capturing* lebih lanjut melalui DOM.

Contoh:

```
document.getElementById("myLink").addEventListener("click", function(event) {  
    event.preventDefault(); // Mencegah link membuka halaman baru  
    console.log("Link diklik, tapi tidak navigasi.");  
    console.log("Target event:", event.target);  
});
```

4. Menghapus Event Listener (`removeEventListener()`)

Penting untuk menghapus *event listener* jika tidak lagi diperlukan, terutama pada aplikasi *single-page* untuk mencegah *memory leak*.

```
myButton.removeEventListener("click", handleButtonClick);
```

Catatan: Anda tidak dapat menghapus *event listener* yang didaftarkan dengan fungsi anonim kecuali Anda menyimpan referensi ke fungsi tersebut.

Form Validation

Validasi formulir adalah proses memastikan bahwa data yang dimasukkan pengguna ke dalam formulir memenuhi kriteria tertentu sebelum dikirim ke server. Validasi dapat dilakukan di sisi klien (menggunakan JavaScript) atau di sisi server. Validasi sisi klien memberikan umpan balik instan kepada pengguna, meningkatkan pengalaman pengguna.

1. Mengakses Elemen Formulir

Sama seperti elemen DOM lainnya, elemen formulir (input, textarea, select) dapat diakses menggunakan metode seperti `getElementById()`, `querySelector()`, dll.

```
const myForm = document.getElementById("myForm");  
const emailInput = document.getElementById("email");  
const passwordInput = document.getElementById("password");
```

2. Event `submit` pada Formulir

Validasi formulir biasanya dilakukan ketika formulir akan dikirim. Kita dapat mendengarkan event `submit` pada elemen formulir.

```
myForm.addEventListener("submit", function(event) {  
    // Lakukan validasi di sini  
    // Jika validasi gagal, panggil event.preventDefault();  
});
```

3. Contoh Validasi Dasar

- **Validasi Kosong:** Memastikan bidang tidak kosong.

```
if (emailInput.value.trim() === "") {  
    alert("Email tidak boleh kosong!");  
    event.preventDefault(); // Mencegah pengiriman formulir  
}
```

- **Validasi Panjang Minimum/Maksimum:** Memastikan panjang input sesuai.

```
if (passwordInput.value.length < 6) {  
    alert("Password minimal 6 karakter!");  
    event.preventDefault();  
}
```

- **Validasi Format Email (Regex Sederhana):** Memastikan format email benar.

```
const emailRegex = /^[^\w-]+(\.[\w-]+)*@[^\w-]+(\.[\w-]+)+$/;  
if (!emailRegex.test(emailInput.value)) {  
    alert("Format email tidak valid!");  
    event.preventDefault();  
}
```

- **Validasi Angka:** Memastikan input adalah angka.

```
const ageInput = document.getElementById("age");
if (isNaN(ageInput.value) || parseInt(ageInput.value) <= 0) {
    alert("Umur harus angka positif!");
    event.preventDefault();
}
```

4. Memberikan Umpan Balik kepada Pengguna

Daripada menggunakan `alert()`, praktik terbaik adalah menampilkan pesan kesalahan langsung di samping bidang input yang bermasalah, atau mengubah gaya bidang input untuk menunjukkan kesalahan.

Kode Contoh

Buat file `index.html` dan `script.js` di folder yang sama.

`index.html` :

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Event Handling & Form Validation</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        .container {
            border: 1px solid #ccc;
            padding: 20px;
            margin-bottom: 20px;
        }
        input[type="text"], input[type="password"], input[type="email"] {
            width: 100%;
            padding: 8px;
            margin: 5px 0 10px 0;
            display: inline-block;
            border: 1px solid #ccc;
            border-radius: 4px;
            box-sizing: border-box;
        }
    </style>
</head>
<body>
    <div class="container">
        <form>
            <label>Name:</label>
            <input type="text" id="name" value="John Doe" />
            <span style="color: red; font-size: small; margin-left: 10px;">*
            <label>Age:</label>
            <input type="text" id="age" value="25" />
            <span style="color: red; font-size: small; margin-left: 10px;">*
            <label>Email:</label>
            <input type="email" id="email" value="john.doe@example.com" />
            <span style="color: red; font-size: small; margin-left: 10px;">*
            <br>
            <input type="checkbox" id="checkbox1" checked="" /> I agree to the terms and conditions
            <input type="checkbox" id="checkbox2" /> I want to receive marketing emails
            <br>
            <button type="submit" style="background-color: #007bff; color: white; padding: 5px 10px; border: none; border-radius: 3px; font-weight: bold;">Submit
```

```
        }
    
```

```
    button {
        background-color: #4CAF50;
        color: white;
        padding: 10px 15px;
        border: none;
        border-radius: 4px;
        cursor: pointer;
    }
    button:hover {
        opacity: 0.8;
    }
    .error-message {
        color: red;
        font-size: 0.9em;
        margin-top: -8px;
        margin-bottom: 5px;
        display: block;
    }
    .input-error {
        border-color: red !important;
    }
</style>
</head>
<body>

<h1>Event Handling Example</h1>
<div class="container">
    <button id="myButton">Klik Saya!</button>
    <p id="message"></p>
    <a href="https://www.google.com" id="googleLink">Kunjungi Google (akan diceg</div>

<h1>Form Validation Example</h1>
<div class="container">
    <form id="registrationForm">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username">
        <span class="error-message" id="usernameError"></span>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email">
        <span class="error-message" id="emailError"></span>
    
```

```
<label for="password">Password:</label>
<input type="password" id="password" name="password">
<span class="error-message" id="passwordError"></span>

<label for="confirmPassword">Konfirmasi Password:</label>
<input type="password" id="confirmPassword" name="confirmPassword">
<span class="error-message" id="confirmPasswordError"></span>

<button type="submit">Daftar</button>
</form>
</div>

<script src="script.js"></script>
</body>
</html>
```

script.js :

```
// Event Handling Section
const myButton = document.getElementById("myButton");
const messageParagraph = document.getElementById("message");
const googleLink = document.getElementById("googleLink");

// Click event on button
myButton.addEventListener("click", function() {
    messageParagraph.textContent = "Tombol berhasil diklik!";
    messageParagraph.style.color = "green";
});

// Mouseover and mouseout events
myButton.addEventListener("mouseover", function() {
    myButton.style.backgroundColor = "#5cb85c";
});

myButton.addEventListener("mouseout", function() {
    myButton.style.backgroundColor = "#4CAF50";
});

// Prevent default behavior of a link
googleLink.addEventListener("click", function(event) {
    event.preventDefault(); // Mencegah navigasi ke Google
```

```
        alert("Navigasi ke Google dicegah!");
        console.log("Link diklik, tapi event default dicegah.");
    });

// Form Validation Section
const registrationForm = document.getElementById("registrationForm");
const usernameInput = document.getElementById("username");
const emailInput = document.getElementById("email");
const passwordInput = document.getElementById("password");
const confirmPasswordInput = document.getElementById("confirmPassword");

const usernameError = document.getElementById("usernameError");
const emailError = document.getElementById("emailError");
const passwordError = document.getElementById("passwordError");
const confirmPasswordError = document.getElementById("confirmPasswordError");

// Function to display error message
function displayError(element, message) {
    element.textContent = message;
    element.previousElementSibling.classList.add("input-error");
}

// Function to clear error message
function clearError(element) {
    element.textContent = "";
    element.previousElementSibling.classList.remove("input-error");
}

registrationForm.addEventListener("submit", function(event) {
    let isValid = true;

    // Clear previous errors
    clearError(usernameError);
    clearError(emailError);
    clearError(passwordError);
    clearError(confirmPasswordError);

    // Validate Username
    if (usernameInput.value.trim() === "") {
        displayError(usernameError, "Username tidak boleh kosong.");
        isValid = false;
    } else if (usernameInput.value.length < 3) {
        displayError(usernameError, "Username minimal 3 karakter.");
        isValid = false;
    }
})
```

```
}

// Validate Email
const emailRegex = /^[^\w-]+(\.[\w-]+)*@[\w-]+(\.[\w-]+)+$/;
if (emailInput.value.trim() === "") {
    displayError(emailError, "Email tidak boleh kosong.");
    isValid = false;
} else if (!emailRegex.test(emailInput.value)) {
    displayError(emailError, "Format email tidak valid.");
    isValid = false;
}

// Validate Password
if (passwordInput.value.trim() === "") {
    displayError(passwordError, "Password tidak boleh kosong.");
    isValid = false;
} else if (passwordInput.value.length < 6) {
    displayError(passwordError, "Password minimal 6 karakter.");
    isValid = false;
}

// Validate Confirm Password
if (confirmPasswordInput.value.trim() === "") {
    displayError(confirmPasswordError, "Konfirmasi password tidak boleh kosong.");
    isValid = false;
} else if (confirmPasswordInput.value !== passwordInput.value) {
    displayError(confirmPasswordError, "Password tidak cocok.");
    isValid = false;
}

if (!isValid) {
    event.preventDefault(); // Stop form submission if validation fails
} else {
    alert("Formulir berhasil divalidasi dan siap dikirim!");
    // Di sini Anda bisa mengirim data formulir ke server (misalnya, menggunakan
}
});

// Optional: Real-time validation on input change (for better UX)
usernameInput.addEventListener("input", () => {
    if (usernameInput.value.trim() === "") {
        displayError(usernameError, "Username tidak boleh kosong.");
    } else if (usernameInput.value.length < 3) {
        displayError(usernameError, "Username minimal 3 karakter.");
    }
});
```

```
        } else {
            clearError(usernameError);
        }
    });

emailInput.addEventListener("input", () => {
    const emailRegex = /^[^\w-]+(\.[\w-]+)*@[^\w-]+\(\.[\w-]+\)+$/;
    if (emailInput.value.trim() === "") {
        displayError(emailError, "Email tidak boleh kosong.");
    } else if (!emailRegex.test(emailInput.value)) {
        displayError(emailError, "Format email tidak valid.");
    } else {
        clearError(emailError);
    }
});

passwordInput.addEventListener("input", () => {
    if (passwordInput.value.trim() === "") {
        displayError(passwordError, "Password tidak boleh kosong.");
    } else if (passwordInput.value.length < 6) {
        displayError(passwordError, "Password minimal 6 karakter.");
    } else {
        clearError(passwordError);
    }
});

confirmPasswordInput.addEventListener("input", () => {
    if (confirmPasswordInput.value.trim() === "") {
        displayError(confirmPasswordError, "Konfirmasi password tidak boleh kosong.");
    } else if (confirmPasswordInput.value !== passwordInput.value) {
        displayError(confirmPasswordError, "Password tidak cocok.");
    } else {
        clearError(confirmPasswordError);
    }
});
```

Langkah-langkah Praktikum

1. Buat folder baru untuk pertemuan ini (misalnya, `pertemuan7`).
2. Di dalam folder tersebut, buat dua file: `index.html` dan `script.js`.
3. Salin dan tempel kode `index.html` dan `script.js` di atas ke file masing-masing.
4. Buka file `index.html` di browser web Anda.

5. Coba klik tombol dan link, amati perilaku *event handling*.
6. Coba isi formulir dengan data yang tidak valid (kosong, email salah format, password kurang dari 6 karakter, konfirmasi password tidak cocok) dan amati pesan kesalahannya.
7. Isi formulir dengan data yang valid dan amati pesan suksesnya.

8. Latihan:

- Tambahkan validasi untuk bidang input lain (misalnya, nomor telepon hanya boleh angka, usia harus di atas 18).
- Implementasikan validasi *real-time* untuk semua bidang formulir (seperti contoh opsional di `script.js`).
- Buat sebuah tombol "Reset Form" yang akan mengosongkan semua bidang input dan menghapus pesan kesalahan.
- Buat sebuah elemen `div` yang akan berubah warna latar belakangnya setiap kali kursor mouse masuk (`mouseover`) dan keluar (`mouseout`) dari area `div` tersebut.

Referensi

- [1] MDN Web Docs. (n.d.). *Introduction to events*. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events
- [2] MDN Web Docs. (n.d.). *Form data validation*. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Pertemuan 8: Asynchronous JavaScript (Callbacks, Promises, Async/Await)

Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Memahami konsep pemrograman asinkron dalam JavaScript.
- Menggunakan *callback functions* untuk menangani operasi asinkron.
- Memahami dan menggunakan *Promises* untuk mengelola operasi asinkron yang lebih kompleks.
- Menggunakan sintaks `async/await` untuk menulis kode asinkron yang lebih bersih dan mudah dibaca.
- Memahami perbedaan antara eksekusi sinkron dan asinkron.

Penjelasan Materi

Sinkron vs. Asinkron

JavaScript secara default adalah bahasa *single-threaded* dan *synchronous*. Ini berarti kode dieksekusi baris demi baris, dan setiap operasi harus selesai sebelum operasi berikutnya dimulai. Jika ada operasi yang memakan waktu lama (misalnya, mengambil data dari server, membaca file), ini akan "memblokir" eksekusi kode selanjutnya, membuat aplikasi terasa tidak responsif.

Sinkron (Synchronous):

- Kode dieksekusi secara berurutan, satu per satu.
- Setiap operasi harus selesai sebelum operasi berikutnya dimulai.
- Memblokir *thread* utama, dapat menyebabkan UI "freeze" jika ada operasi yang lama.

Asinkron (Asynchronous):

- Operasi yang memakan waktu lama dijalankan di latar belakang.
- Kode tidak menunggu operasi tersebut selesai; eksekusi berlanjut ke baris berikutnya.
- Ketika operasi asinkron selesai, ia akan memberi tahu *thread* utama untuk menjalankan *callback* atau *handler* yang sesuai.
- Tidak memblokir *thread* utama, menjaga aplikasi tetap responsif.

Contoh operasi asinkron:

- Mengambil data dari API (menggunakan `fetch` atau `XMLHttpRequest`).
- Membaca atau menulis file.
- Timer (`setTimeout` , `setInterval`).
- Interaksi pengguna (klik, input).

Callback Functions

Callback function adalah fungsi yang dilewatkan sebagai argumen ke fungsi lain, dan akan dieksekusi setelah fungsi utama selesai melakukan tugasnya. Ini adalah cara paling dasar untuk menangani asinkronitas di JavaScript.

```
function fetchData(callback) {
    setTimeout(() => {
        const data = "Data dari server";
        callback(data); // Panggil callback setelah data siap
    }, 2000); // Simulasi penundaan 2 detik
}

function displayData(data) {
    console.log("Data diterima: " + data);
}

console.log("Memulai pengambilan data...");
fetchData(displayData); // displayData adalah callback
console.log("Operasi lain berjalan...");
```

Kekurangan Callback:

- **Callback Hell (Pyramid of Doom):** Ketika ada banyak operasi asinkron yang bergantung satu sama lain, kode menjadi sangat bersarang dan sulit dibaca/dipelihara.

```
getData(function(a) {
    getMoreData(a, function(b) {
        getEvenMoreData(b, function(c) {
            // ... semakin dalam
        });
    });
});
```

- **Error Handling:** Penanganan kesalahan menjadi rumit.

Promises

Promises diperkenalkan di ES6 untuk mengatasi masalah *callback hell* dan membuat penanganan asinkron lebih terstruktur. Sebuah *Promise* merepresentasikan nilai yang mungkin tersedia sekarang, di masa depan,

atau tidak sama sekali. *Promise* memiliki tiga status:

- **Pending:** Status awal, belum terpenuhi atau ditolak.
- **Fulfilled (Resolved):** Operasi berhasil diselesaikan, *Promise* mengembalikan nilai.
- **Rejected:** Operasi gagal, *Promise* mengembalikan alasan kegagalan (error).

Membuat Promise:

Sebuah *Promise* dibuat menggunakan konstruktor `new Promise()`, yang menerima fungsi *executor* dengan dua argumen: `resolve` dan `reject`.

```
const myPromise = new Promise((resolve, reject) => {
    const success = true;
    if (success) {
        resolve("Operasi berhasil!"); // Panggil resolve jika berhasil
    } else {
        reject("Operasi gagal!"); // Panggil reject jika gagal
    }
});
```

Menggunakan Promise (`.then()`, `.catch()`, `.finally()`):

- `.then(onFulfilled, onRejected)` : Digunakan untuk menangani hasil *Promise* yang berhasil (`onFulfilled`) atau gagal (`onRejected`).
- `.catch(onRejected)` : Cara yang lebih baik untuk menangani kesalahan, setara dengan `.then(null, onRejected)`.
- `.finally(onFinally)` : Akan dieksekusi terlepas dari apakah *Promise* berhasil atau gagal.

```
myPromise
    .then(result => {
        console.log(result); // Output: Operasi berhasil!
    })
    .catch(error => {
        console.error(error); // Output: Operasi gagal!
    })
    .finally(() => {
        console.log("Promise selesai.");
    });
// Chaining Promises
```

```
function doSomething() {
    return new Promise(resolve => {
        setTimeout(() => resolve("Sesuatu telah dilakukan"), 1000);
    });
}

function doSomethingElse(result) {
    return new Promise(resolve => {
        setTimeout(() => resolve(` ${result} dan sesuatu yang lain`), 1000);
    });
}

doSomething()
    .then(result => doSomethingElse(result))
    .then(finalResult => console.log(finalResult))
    .catch(error => console.error(error));
```

Promise.all() dan Promise.race() :

- **Promise.all(iterable)** : Mengambil array *Promises* dan mengembalikan *Promise* baru yang akan *resolve* ketika semua *Promises* dalam array telah *resolve*. Jika salah satu *Promise* *reject*, maka **Promise.all** akan langsung *reject*.
- **Promise.race(iterable)** : Mengambil array *Promises* dan mengembalikan *Promise* baru yang akan *resolve* atau *reject* segera setelah salah satu *Promise* dalam array *resolve* atau *reject*.

Async/Await (ES2017)

async/await adalah sintaks yang dibangun di atas *Promises* untuk membuat kode asinkron terlihat dan terasa seperti kode sinkron. Ini membuat kode asinkron jauh lebih mudah dibaca dan ditulis.

- **async keyword:** Digunakan di depan deklarasi fungsi untuk menandakan bahwa fungsi tersebut akan selalu mengembalikan *Promise*. Jika fungsi mengembalikan nilai non-*Promise*, JavaScript akan secara otomatis membungkusnya dalam *Promise* yang *resolved*.
- **await keyword:** Hanya dapat digunakan di dalam fungsi **async**. Ini akan "menunggu" *Promise* untuk *resolve* dan mengembalikan nilai *resolved* tersebut. Jika *Promise* *reject*, **await** akan melempar *error* yang dapat ditangkap dengan **try...catch**.



```
function simulateFetch(url) {
    return new Promise(resolve => {
        setTimeout(() => {
```

```
        resolve(`Data dari ${url}`);
    }, 1500);
});

}

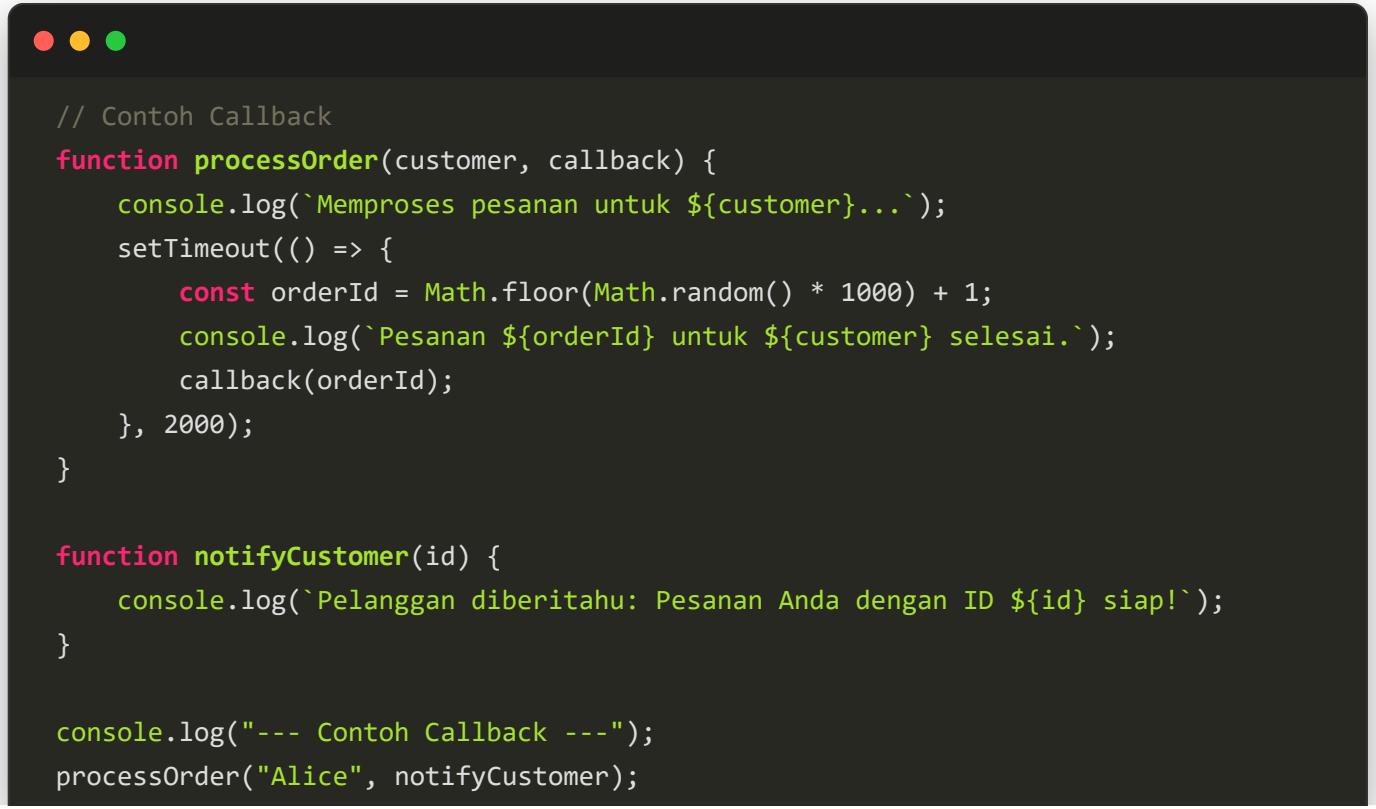
async function getData() {
    try {
        console.log("Mulai mengambil data...");
        const data1 = await simulateFetch("api/users");
        console.log(data1);

        const data2 = await simulateFetch("api/products");
        console.log(data2);

        console.log("Semua data berhasil diambil.");
    } catch (error) {
        console.error("Terjadi kesalahan: ", error);
    }
}

getData();
console.log("Ini akan dieksekusi sebelum data diambil sepenuhnya.");
```

Kode Contoh



```
// Contoh Callback

function processOrder(customer, callback) {
    console.log(`Memproses pesanan untuk ${customer}...`);
    setTimeout(() => {
        const orderId = Math.floor(Math.random() * 1000) + 1;
        console.log(`Pesanan ${orderId} untuk ${customer} selesai.`);
        callback(orderId);
    }, 2000);
}

function notifyCustomer(id) {
    console.log(`Pelanggan diberitahu: Pesanan Anda dengan ID ${id} siap!`);
}

console.log("--- Contoh Callback ---");
processOrder("Alice", notifyCustomer);
```

```
processOrder("Bob", (id) => {
    console.log(`Bob, pesanan Anda #${id} telah dikirim.`);
});

// Contoh Promises
function downloadFile(fileName) {
    return new Promise((resolve, reject) => {
        console.log(`Mulai mengunduh ${fileName}...`);
        const success = Math.random() > 0.5; // Simulasi sukses/gagal acak
        setTimeout(() => {
            if (success) {
                resolve(`${fileName} berhasil diunduh.`);
            } else {
                reject(`${fileName} gagal diunduh!`);
            }
        }, 3000);
    });
}

console.log("\n--- Contoh Promises ---");
downloadFile("dokumen.pdf")
    .then(message => {
        console.log(message);
        return downloadFile("gambar.jpg"); // Chaining
    })
    .then(message => {
        console.log(message);
        console.log("Semua file berhasil diunduh!");
    })
    .catch(error => {
        console.error("Kesalahan umum saat mengunduh: ", error);
    })
    .finally(() => {
        console.log("Proses unduh selesai (baik sukses maupun gagal).");
    });

// Contoh Async/Await
async function fetchUserData() {
    console.log("\n--- Contoh Async/Await ---");
    try {
        console.log("Mengambil data pengguna...");
        const userResponse = await new Promise(resolve => setTimeout(() => resolve({
            console.log("Data pengguna:", userResponse);
        }), 2000));
    } catch (error) {
        console.error("Terjadi kesalahan saat mengambil data pengguna: ", error);
    }
}
```

```
        console.log("Mengambil postingan pengguna...");  
        const postsResponse = await new Promise(resolve => setTimeout(() => resolve(  
            console.log("Postingan pengguna:", postsResponse));  
  
            console.log("Data pengguna dan postingan berhasil diambil.");  
        } catch (error) {  
            console.error("Gagal mengambil data: ", error);  
        }  
    }  
  
fetchUserData();  
console.log("Ini dieksekusi segera setelah memanggil fetchUserData.");  
  
// Contoh Promise.all  
async function fetchMultipleData() {  
    console.log("\n--- Contoh Promise.all ---");  
    try {  
        const [dataA, dataB] = await Promise.all([  
            new Promise(resolve => setTimeout(() => resolve("Result A"), 1000)),  
            new Promise(resolve => setTimeout(() => resolve("Result B"), 2000))  
        ]);  
        console.log("Semua Promise.all berhasil:", dataA, dataB);  
    } catch (error) {  
        console.error("Salah satu Promise.all gagal:", error);  
    }  
}  
fetchMultipleData();
```

Langkah-langkah Praktikum

1. Buka editor teks Anda dan buat file baru bernama `pertemuan8.js`.
2. Salin dan tempel kode contoh di atas ke dalam `pertemuan8.js`.
3. Buka terminal atau Command Prompt, navigasikan ke direktori tempat Anda menyimpan `pertemuan8.js`.
4. Jalankan file menggunakan Node.js:



```
node pertemuan8.js
```

5. Amati output di konsol. Perhatikan urutan eksekusi kode sinkron dan asinkron. Anda akan melihat pesan "Operasi lain berjalan..." atau "Ini dieksekusi segera..." muncul sebelum hasil dari operasi asinkron.
6. Coba ubah nilai `success` di fungsi `downloadFile` menjadi `false` untuk melihat bagaimana `.catch()` menangani kesalahan.

7. Latihan:

- o Buat fungsi `simulateLogin(username, password)` yang mengembalikan *Promise*. *Promise* ini akan *resolve* setelah 2 detik jika `username` adalah "user" dan `password` adalah "pass", dan *reject* jika tidak.
- o Gunakan `async/await` untuk memanggil `simulateLogin`. Tangani kasus sukses dan gagal menggunakan `try...catch`.
- o Buat tiga fungsi yang masing-masing mengembalikan *Promise* yang *resolve* setelah waktu yang berbeda (misalnya, 1, 2, dan 3 detik) dengan pesan yang berbeda. Gunakan `Promise.all()` untuk menunggu ketiga *Promise* ini selesai dan cetak semua hasilnya.
- o Buat dua fungsi yang masing-masing mengembalikan *Promise* yang *resolve* setelah waktu yang berbeda. Gunakan `Promise.race()` untuk melihat *Promise* mana yang *resolve* lebih dulu dan cetak hasilnya.

Referensi

- [1] MDN Web Docs. (n.d.). *Asynchronous JavaScript*. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Asynchronous>
- [2] MDN Web Docs. (n.d.). *Using Promises*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises
- [3] MDN Web Docs. (n.d.). *async function*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

Pertemuan 9: ES6+ Features (Arrow Functions, Destructuring, Classes)

Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Memahami dan menggunakan *Arrow Functions* sebagai alternatif penulisan fungsi.
- Menggunakan *Destructuring Assignment* untuk mengekstrak nilai dari array dan objek.
- Memahami konsep *Classes* dan *Object-Oriented Programming (OOP)* dasar dalam JavaScript.
- Menggunakan *Spread* dan *Rest Operators* untuk manipulasi array dan argumen fungsi.
- Mengenali dan menggunakan fitur-fitur ES6+ lainnya yang umum.

Penjelasan Materi

ES6 (ECMAScript 2015) membawa banyak fitur baru yang signifikan ke JavaScript, membuat bahasa ini lebih modern, ekspresif, dan kuat. Sejak itu, setiap tahun ada rilis ECMAScript baru (ES2016, ES2017, dst.) yang menambahkan fitur-fitur tambahan. Pertemuan ini akan fokus pada beberapa fitur ES6+ yang paling sering digunakan.

1. Arrow Functions (`=>`)

Sudah sedikit disinggung di Pertemuan 4, *Arrow Functions* menyediakan sintaks yang lebih ringkas untuk menulis ekspresi fungsi. Perbedaan utamanya adalah bagaimana mereka menangani `this` keyword (mereka tidak memiliki `this` mereka sendiri, melainkan mewarisi `this` dari scope induknya) dan mereka tidak dapat digunakan sebagai konstruktor.

Sintaks Ringkas:

```
// Fungsi tradisional
const addTraditional = function(a, b) {
    return a + b;
};

// Arrow function
const addArrow = (a, b) => a + b;

// Satu parameter, tanpa tanda kurung
const square = x => x * x;

// Tanpa parameter
const greet = () => console.log("Hello!");
```

```
// Dengan blok kode (membutuhkan return eksplisit)
const calculate = (a, b) => {
  const sum = a + b;
  return sum * 2;
};
```

Perilaku `this` :

Ini adalah perbedaan paling penting. Dalam fungsi tradisional, nilai `this` ditentukan oleh bagaimana fungsi dipanggil. Dalam *arrow function*, `this` secara leksikal terikat pada *scope* di mana *arrow function* didefinisikan.



```
const person = {
  name: "Alice",
  greetTraditional: function() {
    setTimeout(function() {
      // `this` di sini mengacu pada window/global object (dalam non-strict mode)
      // atau undefined (dalam strict mode), BUKAN pada `person`
      console.log(`Halo, nama saya ${this.name}`);
    }, 1000);
  },
  greetArrow: function() {
    setTimeout(() => {
      // `this` di sini secara leksikal terikat pada `person` object
      console.log(`Halo, nama saya ${this.name}`);
    }, 1000);
  }
};

person.greetTraditional(); // Output: Halo, nama saya (kosong/undefined)
person.greetArrow();       // Output: Halo, nama saya Alice
```

2. Destructuring Assignment

Memungkinkan Anda untuk mengekstrak nilai dari array atau properti dari objek ke dalam variabel terpisah dengan sintaks yang lebih ringkas.

Array Destructuring:

```
const numbers = [10, 20, 30, 40, 50];

// Tanpa destructuring
// const a = numbers[0];
// const b = numbers[1];

// Dengan destructuring
const [a, b, c] = numbers;
console.log(a, b, c); // Output: 10 20 30

// Melewatkkan elemen
const [first, , third] = numbers;
console.log(first, third); // Output: 10 30

// Dengan rest operator
const [head, ...tail] = numbers;
console.log(head); // Output: 10
console.log(tail); // Output: [20, 30, 40, 50]
```

Object Destructuring:

```
const user = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  city: "New York"
};

// Tanpa destructuring
// const firstName = user.firstName;
// const age = user.age;

// Dengan destructuring
const { firstName, age } = user;
console.log(firstName, age); // Output: John 30

// Mengganti nama variabel
const { firstName: fName, lastName: lName } = user;
console.log(fName, lName); // Output: John Doe
```

```
// Dengan nilai default
const { country = "USA", city } = user;
console.log(country, city); // Output: USA New York

// Dengan rest operator
const { firstName: name, ...rest } = user;
console.log(name); // Output: John
console.log(rest); // Output: { lastName: "Doe", age: 30, city: "New York" }
```

3. Classes

ES6 memperkenalkan sintaks `class` yang menyediakan cara yang lebih bersih dan familiar untuk membuat *constructor function* dan bekerja dengan *Object-Oriented Programming (OOP)*. Meskipun terlihat seperti OOP tradisional, di balik layar, JavaScript masih menggunakan *prototypal inheritance*.

Sintaks Dasar:

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    greet() {
        console.log(`Halo, nama saya ${this.name} dan saya ${this.age} tahun.`);
    }
}

const person1 = new Person("Budi", 25);
person1.greet(); // Output: Halo, nama saya Budi dan saya 25 tahun.

const person2 = new Person("Siti", 30);
person2.greet(); // Output: Halo, nama saya Siti dan saya 30 tahun.
```

Inheritance (`extends` dan `super`)

```
class Student extends Person {
```

```
constructor(name, age, studentId) {
    super(name, age); // Memanggil constructor dari kelas induk (Person)
    this.studentId = studentId;
}

study() {
    console.log(`${this.name} sedang belajar dengan ID ${this.studentId}.`);
}

// Override method dari kelas induk
greet() {
    console.log(`Hai, saya ${this.name}, seorang mahasiswa.`);
}
}

const student1 = new Student("Dian", 20, "12345");
student1.greet(); // Output: Hai, saya Dian, seorang mahasiswa.
student1.study(); // Output: Dian sedang belajar dengan ID 12345.
```

4. Spread Operator (...) dan Rest Parameters (...)

Operator yang sama (...) digunakan untuk dua tujuan berbeda tergantung pada konteksnya.

Spread Operator:

Digunakan untuk "menyebarluaskan" elemen-elemen dari array atau properti dari objek ke dalam array atau objek lain, atau sebagai argumen fungsi.

- **Menggabungkan Array:**

```
const arr1 = [1, 2];
const arr2 = [3, 4];
const combinedArr = [...arr1, ...arr2];
console.log(combinedArr); // Output: [1, 2, 3, 4]
```

- **Menggabungkan Objek:**

```
const obj1 = { a: 1, b: 2 };
const obj2 = { c: 3, d: 4 };
```

```
const combinedObj = { ...obj1, ...obj2 };
console.log(combinedObj); // Output: { a: 1, b: 2, c: 3, d: 4 }
```

- **Meneruskan Argumen ke Fungsi:**

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

Rest Parameters:

Digunakan dalam definisi fungsi untuk mengumpulkan semua argumen yang tersisa ke dalam sebuah array.

```
function logArguments(firstArg, ...restArgs) {
    console.log("Argumen pertama:", firstArg);
    console.log("Argumen lainnya:", restArgs);
}

logArguments(1, 2, 3, 4, 5);
// Output:
// Argumen pertama: 1
// Argumen lainnya: [2, 3, 4, 5]

function sumAll(...numbers) {
    return numbers.reduce((total, num) => total + num, 0);
}
console.log(sumAll(1, 2, 3));      // Output: 6
console.log(sumAll(10, 20, 30, 40)); // Output: 100
```

5. Fitur ES6+ Lainnya yang Penting

- **Template Literals (`)**: String yang memungkinkan *embedded expressions* dan *multi-line strings*.

```
const name = "World";
console.log(`Hello, ${name}!`)
```

This is a multi-line string.');

```

- **Default Parameters:** Memberikan nilai default untuk parameter fungsi jika tidak disediakan.

```
function multiply(a, b = 1) {
 return a * b;
}
console.log(multiply(5)); // Output: 5
console.log(multiply(5, 2)); // Output: 10
```

- **Modules ( import / export ):** Cara standar untuk mengorganisir kode JavaScript ke dalam file terpisah yang dapat diimpor dan diekspor. (Akan dibahas lebih lanjut di Pertemuan 10 dengan Node.js).

## Kode Contoh

```
// Contoh Arrow Functions
const greetUser = (name) => `Halo, ${name}!`;
console.log(greetUser("Andi"));

const calculateArea = (length, width) => {
 const area = length * width;
 return `Luas: ${area} satuan.`;
};
console.log(calculateArea(10, 5));

// Contoh Destructuring Array
const rgbColors = [255, 0, 128];
const [red, green, blue] = rgbColors;
console.log(`Red: ${red}, Green: ${green}, Blue: ${blue}`);

const [firstItem, ...remainingItems] = ["Laptop", "Mouse", "Keyboard", "Monitor"];
console.log("First Item:", firstItem);
```

```
console.log("Remaining Items:", remainingItems);

// Contoh Destructuring Object
const product = {
 productName: "Smartphone",
 price: 799,
 inStock: true,
 details: { weight: "200g", color: "Black" }
};

const { productName, price, inStock } = product;
console.log(`Produk: ${productName}, Harga: ${price}, Tersedia: ${inStock}`);

const { details: { color } } = product;
console.log("Warna Produk:", color);

const { manufacturer = "Unknown", price: productPrice } = product;
console.log(`Produsen: ${manufacturer}, Harga Produk: ${productPrice}`);

// Contoh Classes
class Animal {
 constructor(name, species) {
 this.name = name;
 this.species = species;
 }

 makeSound() {
 console.log("Suara hewan...");
 }
}

class Dog extends Animal {
 constructor(name, breed) {
 super(name, "Dog");
 this.breed = breed;
 }

 makeSound() {
 console.log("Guk guk!");
 }

 fetch() {
 console.log(`${this.name} (${this.breed}) sedang mengambil bola.`);
 }
}
```

```
}
```

```
const myDog = new Dog("Buddy", "Golden Retriever");
myDog.makeSound(); // Output: Guk guk!
myDog.fetch();

// Contoh Spread dan Rest Operators
const arrA = [1, 2, 3];
const arrB = [4, 5, 6];
const combinedArray = [...arrA, ...arrB];
console.log("Array Gabungan:", combinedArray);

const userProfile = { id: 1, username: "coder123" };
const userDetails = { email: "coder@example.com", age: 28 };
const fullUser = { ...userProfile, ...userDetails, isActive: true };
console.log("Profil Pengguna Lengkap:", fullUser);

function displayInfo(title, ...items) {
 console.log(`\n${title}:`);
 items.forEach(item => console.log(` - ${item}`));
}

displayInfo("Daftar Buah", "Apel", "Jeruk", "Mangga");
displayInfo("Angka Favorit", 7, 13, 42);
```

## Langkah-langkah Praktikum

1. Buka editor teks Anda dan buat file baru bernama `pertemuan9.js`.
2. Salin dan tempel kode contoh di atas ke dalam `pertemuan9.js`.
3. Buka terminal atau Command Prompt, navigasikan ke direktori tempat Anda menyimpan `pertemuan9.js`.
4. Jalankan file menggunakan Node.js:



```
node pertemuan9.js
```

5. Amati output di konsol. Coba modifikasi contoh-contoh yang ada untuk lebih memahami bagaimana fitur-fitur ES6+ bekerja.
6. **Latihan:**

- Buat sebuah *arrow function* yang menerima array angka dan mengembalikan array baru yang berisi kuadrat dari setiap angka genap dalam array asli.
- Buat sebuah objek `book` dengan properti `title`, `author`, `year`, dan `genres` (array). Gunakan *object destructuring* untuk mengekstrak `title` dan `author` ke variabel terpisah, dan `genres` ke variabel `categories`.
- Buat sebuah kelas `Shape` dengan *constructor* yang menerima `color`. Tambahkan metode `getArea()` yang mengembalikan 0. Kemudian, buat kelas `Circle` yang `extends` `Shape` dan tambahkan properti `radius`. *Override* metode `getArea()` untuk menghitung luas lingkaran (`Math.PI * radius * radius`).
- Gunakan *spread operator* untuk membuat salinan array atau objek, lalu modifikasi salinan tersebut tanpa memengaruhi yang asli.

## Referensi

- [1] MDN Web Docs. (n.d.). *Arrow function expressions*. Retrieved from [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow\\_functions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions)
- [2] MDN Web Docs. (n.d.). *Destructuring assignment*. Retrieved from [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring\\_assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)
- [3] MDN Web Docs. (n.d.). *Classes*. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>
- [4] MDN Web Docs. (n.d.). *Spread syntax (...)*. Retrieved from [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread\\_syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax)
- [5] MDN Web Docs. (n.d.). *Rest parameters*. Retrieved from [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest\\_parameters](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters)

# Pertemuan 10: Pengenalan Node.js dan NPM (Dasar)

## Tujuan Pembelajaran

Setelah menyelesaikan pertemuan ini, peserta diharapkan mampu:

- Memahami apa itu Node.js dan mengapa itu penting.
- Menyiapkan lingkungan Node.js.
- Menggunakan NPM (Node Package Manager) untuk mengelola dependensi proyek.
- Membuat dan menjalankan aplikasi Node.js sederhana.
- Memahami konsep modul di Node.js.

## Penjelasan Materi

### Apa itu Node.js?

Node.js adalah *runtime environment* JavaScript *open-source* dan *cross-platform* yang memungkinkan JavaScript dijalankan di luar browser. Node.js dibangun di atas *engine* JavaScript V8 Google Chrome. Ini berarti Anda dapat menggunakan JavaScript untuk membangun aplikasi *server-side*, *command-line tools*, dan aplikasi desktop, bukan hanya untuk *frontend* web.

### Mengapa Node.js Penting?

- **JavaScript di mana-mana:** Memungkinkan pengembang menggunakan satu bahasa (JavaScript) untuk *frontend* dan *backend*, menyederhanakan proses pengembangan dan mengurangi *context switching*.
- **Non-blocking I/O (Asynchronous):** Node.js menggunakan model I/O *non-blocking* dan *event-driven*, yang membuatnya sangat efisien dan *scalable* untuk aplikasi *real-time* dan aplikasi yang banyak menangani I/O (misalnya, API, *chat applications*).
- **Ekosistem Besar:** Memiliki ekosistem *package* yang sangat besar melalui NPM, yang mempercepat pengembangan dengan menyediakan banyak *library* dan *framework* siap pakai.

### NPM (Node Package Manager)

NPM adalah manajer *package* default untuk Node.js. Ini adalah *tool* baris perintah yang memungkinkan Anda untuk:

- Menginstal *package* (*library/modul*) JavaScript dari registry NPM.
- Mengelola dependensi proyek Anda.
- Menjalankan *script* yang ditentukan dalam file `package.json`.

#### File `package.json` :

Ini adalah file manifest untuk proyek Node.js Anda. Ini berisi metadata tentang proyek (nama, versi, deskripsi, penulis) dan yang paling penting, daftar dependensi (*package* yang dibutuhkan proyek Anda) dan *script* yang dapat dijalankan.

#### Perintah NPM Dasar:

- `npm init` : Menginisialisasi proyek Node.js baru dan membuat file `package.json`.

```
npm init -y # -y untuk melewati semua pertanyaan dan menggunakan default
```

- `npm install <package-name>` : Menginstal *package* ke dalam proyek Anda. Ini akan menambahkannya ke `node_modules` folder dan mencatatnya di `package.json` sebagai dependensi.

```
npm install express # Menginstal framework web Express
npm install lodash # Menginstal utility library Lodash
```

- `npm install` : Menginstal semua dependensi yang tercantum dalam `package.json`.
- `npm install <package-name> --save-dev` atau `npm install <package-name> -D` : Menginstal *package* sebagai *dev dependency* (hanya dibutuhkan selama pengembangan, tidak di produksi).

```
npm install nodemon -D # nodemon untuk restart server otomatis
```

- `npm uninstall <package-name>` : Menghapus *package* dari proyek Anda.
- `npm start` : Menjalankan `script start` yang didefinisikan di `package.json`.
- `npm run <script-name>` : Menjalankan `script kustom` yang didefinisikan di `package.json`.

## Modul di Node.js

Node.js menggunakan sistem modul CommonJS secara default (meskipun ES Modules juga didukung). Ini memungkinkan Anda untuk membagi kode Anda menjadi file-file terpisah (modul) dan kemudian mengimpornya di file lain.

- `require()` : Digunakan untuk mengimpor modul.
- `module.exports` atau `exports` : Digunakan untuk mengeksport nilai dari modul.

### Contoh Modul:

`math.js` :

```
// math.js
function add(a, b) {
 return a + b;
}

function subtract(a, b) {
 return a - b;
}

// Mengekspor fungsi-fungsi ini agar bisa digunakan di file lain
module.exports = {
 add: add,
 subtract: subtract
};

// Atau bisa juga seperti ini (shorthand property names ES6)
// module.exports = {
// add,
// subtract
// };

// Atau mengekspor satu per satu
// exports.add = add;
// exports.subtract = subtract;
```

app.js :

```
// app.js
const math = require("./math"); // Mengimpor modul math.js

console.log("Hasil Penjumlahan: ", math.add(5, 3)); // Output: 8
console.log("Hasil Pengurangan: ", math.subtract(10, 4)); // Output: 6
```

## Membuat Aplikasi Node.js Sederhana (Web Server)

Contoh paling umum dari aplikasi Node.js adalah *web server*.

### Langkah 1: Inisialisasi Proyek

```
mkdir my-node-app
cd my-node-app
npm init -y
```

### Langkah 2: Buat File Server ( `app.js` atau `server.js` )

```
// server.js
const http = require("http"); // Mengimpor modul HTTP bawaan Node.js

const hostname = "127.0.0.1"; // localhost
const port = 3000;

// Membuat server
const server = http.createServer((req, res) => {
 // Mengatur header respons HTTP
 res.statusCode = 200;
 res.setHeader("Content-Type", "text/plain");

 // Mengirim respons ke klien
 res.end("Hello World dari Node.js Server!\n");
});

// Server mendengarkan permintaan pada port dan hostname tertentu
server.listen(port, hostname, () => {
 console.log(`Server berjalan di http://:${hostname}:${port}/`);
});
```

### Langkah 3: Jalankan Server

```
node server.js
```

Buka browser Anda dan kunjungi <http://127.0.0.1:3000/>. Anda akan melihat pesan "Hello World dari Node.js Server!".

## Kode Contoh

```
// File: calculator.js (Modul)
function add(a, b) {
 return a + b;
}

function subtract(a, b) {
 return a - b;
}

function multiply(a, b) {
 return a * b;
}

function divide(a, b) {
 if (b === 0) {
 return "Tidak bisa dibagi nol!";
 }
 return a / b;
}

module.exports = {
 add,
 subtract,
 multiply,
 divide
};

// File: app.js (Aplikasi Utama)
const calculator = require("./calculator"); // Mengimpor modul calculator

console.log("--- Aplikasi Kalkulator Sederhana ---");
console.log("5 + 3 =", calculator.add(5, 3));
console.log("10 - 4 =", calculator.subtract(10, 4));
console.log("6 * 7 =", calculator.multiply(6, 7));
console.log("10 / 2 =", calculator.divide(10, 2));
console.log("10 / 0 =", calculator.divide(10, 0));

// Contoh penggunaan package eksternal (misal: lodash)
// Pastikan Anda sudah menginstal lodash: npm install lodash
const _ = require("lodash");
```

```
const numbers = [1, 2, 3, 4, 5];
console.log("\n--- Contoh Lodash ---");
console.log("Array asli:", numbers);
console.log("Array terbalik:", _.reverse([...numbers])); // Menggunakan spread untuk
console.log("Sum of numbers:", _.sum(numbers));

// Contoh sederhana web server dengan Express.js
// Pastikan Anda sudah menginstal express: npm install express
const express = require("express");
const app = express();
const port = 4000;

app.get("/", (req, res) => {
 res.send("Halo dari Express.js!");
});

app.get("/api/users", (req, res) => {
 const users = [
 { id: 1, name: "Alice" },
 { id: 2, name: "Bob" }
];
 res.json(users);
});

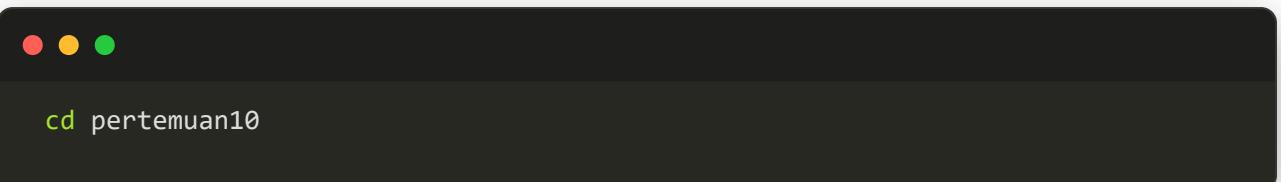
app.listen(port, () => {
 console.log(`Express server berjalan di http://localhost:${port}`);
});

console.log("\nServer Express sedang berjalan di port 4000. Buka browser Anda dan ku
```

## Langkah-langkah Praktikum

### Bagian 1: Modul Node.js Sederhana

1. Buat folder baru untuk pertemuan ini (misalnya, `pertemuan10`).
2. Masuk ke folder tersebut di terminal Anda:



The screenshot shows a dark-themed terminal window. At the top, there are three colored window control buttons (red, yellow, green). Below them, the command `cd pertemuan10` is being typed in. The terminal is set against a dark background with white text.

3. Inisialisasi proyek Node.js:

```
npm init -y
```

4. Buat file `calculator.js` dan salin kode `calculator.js` dari bagian "Kode Contoh" di atas ke dalamnya.

5. Buat file `app.js` dan salin kode `app.js` (bagian aplikasi utama) dari bagian "Kode Contoh" di atas ke dalamnya.

6. Jalankan `app.js`:

```
node app.js
```

7. Amati output di konsol.

## Bagian 2: Menggunakan NPM Package (Lodash)

1. Di dalam folder `pertemuan10` yang sama, instal package `lodash`:

```
npm install lodash
```

2. Buka kembali `app.js` dan hapus komentar pada bagian "Contoh Lodash".

3. Jalankan kembali `app.js`:

```
node app.js
```

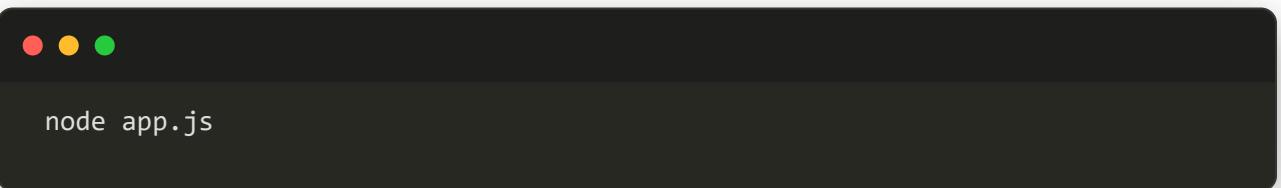
4. Amati output baru yang menunjukkan penggunaan Lodash.

## Bagian 3: Membuat Web Server dengan Express.js

1. Di dalam folder `pertemuan10` yang sama, instal package `express`:

```
npm install express
```

2. Buka kembali `app.js` dan hapus komentar pada bagian "Contoh sederhana web server dengan Express.js".
3. Jalankan kembali `app.js` :



```
node app.js
```

4. Buka browser Anda dan kunjungi `http://localhost:4000/` dan `http://localhost:4000/api/users`. Amati respons dari server.
5. Untuk menghentikan server, tekan `Ctrl+C` di terminal Anda.

#### **Latihan:**

- Buat modul baru bernama `stringUtils.js` yang berisi fungsi untuk membalikkan string (`reverseString`) dan menghitung jumlah karakter dalam string (`countCharacters`). Ekspor fungsi-fungsi ini.
- Di `app.js`, impor `stringUtils.js` dan gunakan fungsi-fungsi tersebut untuk memproses beberapa string.
- Modifikasi server Express.js di `app.js` untuk menambahkan *endpoint* baru, misalnya `/api/products`, yang akan mengembalikan array objek produk.
- Coba instal *package* NPM lain (misalnya, `axios`) untuk membuat permintaan HTTP dari Node.js) dan gunakan dalam `app.js` untuk mengambil data dari API publik (misalnya, JSONPlaceholder).

#### **Referensi**

- [1] Node.js. (n.d.). *About Node.js*. Retrieved from <https://nodejs.org/en/about>
- [2] NPM. (n.d.). *About npm*. Retrieved from <https://docs.npmjs.com/about-npm>
- [3] MDN Web Docs. (n.d.). *Modules*. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules>
- [4] Express.js. (n.d.). *Getting started*. Retrieved from <https://expressjs.com/en/starter/installing.html>