LEARN JAVASCRIPT

BEGINNERS EDITION

A Complete Beginner's Guide to Learn JavaScript

Suman Kunwar

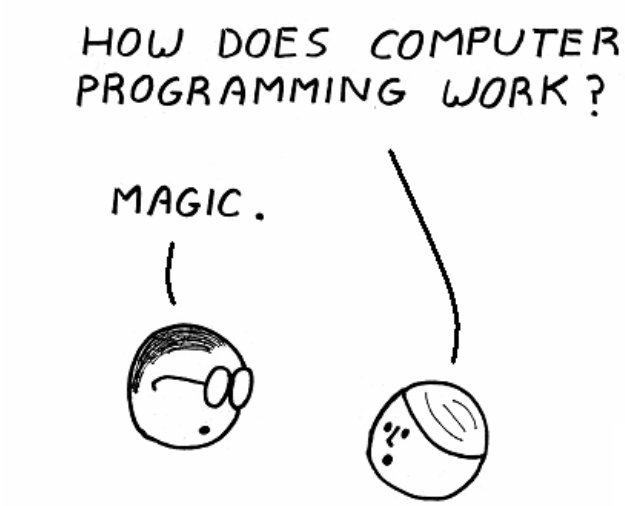
Table of Contents

Bab 1

Pendahuluan

Komputer umum dalam dunia saat ini, karena mereka mampu melakukan berbagai tugas dengan cepat dan akurat. Mereka digunakan dalam banyak industri yang berbeda, seperti bisnis, perawatan kesehatan, pendidikan, dan hiburan, dan telah menjadi bagian penting dari kehidupan sehari-hari bagi banyak orang. Selain itu, mereka juga digunakan untuk melakukan perhitungan ilmiah dan matematika yang kompleks, menyimpan dan memproses jumlah data yang besar, dan berkomunikasi dengan orang-orang di seluruh dunia.

Pemrograman melibatkan menciptakan seperangkat instruksi, yang disebut program, agar diikuti oleh komputer. Menulis program dapat menjadi tugas yang membosankan dan menjengkelkan pada saat tertentu karena komputer sangat presisi dan memerlukan instruksi yang spesifik untuk menyelesaikan tugas.



Bahasa pemrograman adalah bahasa buatan yang digunakan untuk memberikan instruksi kepada komputer. Mereka digunakan dalam sebagian besar tugas pemrograman dan didasarkan pada cara manusia berkomunikasi satu sama lain. Seperti bahasa manusia, bahasa pemrograman memungkinkan kata-kata dan frasa digabungkan untuk menyatakan konsep baru. Menarik untuk dicatat bahwa cara yang paling efektif untuk berkomunikasi dengan komputer melibatkan penggunaan bahasa yang mirip dengan bahasa manusia.

Di masa lalu, cara utama untuk berinteraksi dengan komputer adalah melalui antarmuka berbasis bahasa seperti dasar dan DOS. Ini sebagian besar digantikan oleh antarmuka visual, yang lebih mudah untuk dipelajari tetapi menawarkan lebih sedikit fleksibilitas. Namun, bahasa pemrograman seperti *JavaScript* masih digunakan dan dapat ditemukan di peramban web modern dan hampir semua perangkat.

JavaScript (*JS singkatnya*) adalah bahasa pemrograman yang digunakan untuk membuat interaksi dinamis saat mengembangkan halaman web, permainan, aplikasi, dan bahkan server. JavaScript dimulai di Netscape, sebuah peramban web yang dikembangkan pada tahun 1990-an, dan saat ini menjadi salah satu bahasa pemrograman yang paling terkenal dan digunakan.

Pada awalnya, dibuat untuk membuat halaman web menjadi hidup dan hanya dapat berjalan di peramban. Sekarang, ia berjalan di semua perangkat yang mendukung mesin JavaScript. Objek standar seperti Array, Date, dan Math tersedia di JavaScript, serta operator, struktur kontrol, dan pernyataan. *JavaScript di sisi klien* dan *JavaScript di sisi server* adalah versi yang diperluas dari JavaScript inti.

- *JavaScript di sisi klien* memungkinkan peningkatan dan manipulasi halaman web dan peramban klien. Respons terhadap peristiwa pengguna seperti klik mouse, input formulir, dan navigasi halaman adalah beberapa contohnya.
- *JavaScript di sisi server* memungkinkan akses ke server, database, dan sistem file.

JavaScript adalah bahasa yang diinterpretasikan. Saat menjalankan Javascript, interpreter menginterpretasikan setiap baris dan menjalankannya. Peramban modern menggunakan teknologi Just In Time (JIT) untuk kompilasi, yang mengkompilasi JavaScript menjadi kode byte yang dapat dijalankan.

> "LiveScript" adalah nama awal yang diberikan pada JavaScript.

Kode, dan apa yang harus dilakukan dengannya

Kode adalah instruksi tertulis yang membentuk suatu program. Di sini, banyak bab berisi banyak kode, dan penting untuk membaca dan menulis kode sebagai bagian dari pembelajaran cara memprogram. Anda seharusnya tidak hanya sekali melihat contoh-contoh itu - baca dengan cermat dan cobalah untuk memahaminya. Ini mungkin sulit pada awalnya, tetapi dengan latihan, Anda akan

memperbaiki diri. Hal yang sama berlaku untuk latihan-latihan - pastikan Anda benar-benar mencoba untuk menulis solusi sebelum menganggap Anda memahaminya. Ini juga membantu untuk mencoba menjalankan solusi latihan dalam interpreter JavaScript, karena hal ini akan memungkinkan Anda melihat apakah kode Anda berfungsi dengan benar dan mungkin mendorong Anda untuk bereksperimen dan lebih jauh dari latihan-latihan tersebut.

Konvensi tipografi

Di sini, teks ditulis dengan huruf tetap yang mewakili elemen dari suatu program. Ini bisa menjadi fragmen yang berdiri sendiri atau referensi ke bagian program yang dekat. Program-program, seperti yang ditunjukkan di bawah ini, ditulis dengan cara ini:

```
const numbers = [45, 4, 9, 16, 25];
let txt = "";
for (let x in numbers) {
  txt += numbers[x];
}
```

Terkadang, output yang diharapkan dari suatu program ditulis setelahnya, diawali dengan dua garis miring dan *Hasil*, seperti ini:

```
console.log(txt);

// Hasil: txt = '45491625'
```

Dedikasi

Buku ini didedikasikan, dengan penuh rasa hormat dan kekaguman, kepada semangat komputer dan bahasa pemrograman dalam dunia kita.

> "Seni pemrograman adalah seni mengorganisir kompleksitas, menguasai beragam hal, dan menghindari kekacauan yang tak terkendali seefektif mungkin."
>
> - Edsger Dijkstra

Hak Cipta

Belajar JavaScript: Edisi Pemula
Edisi Pertama

ASIN: B0C53J11V7

Pengantar

"Buku Belajar JavaScript: Edisi Pemula" adalah buku yang menawarkan eksplorasi komprehensif tentang JavaScript, menempatkannya sebagai bahasa yang sangat penting dalam lanskap digital yang terus berubah. Dengan fokus pada fondasi dan praktik, sumber daya ini cocok untuk semua orang yang ingin mempelajari bahasa pemrograman JavaScript.

Buku ini dimulai dengan membahas aspek-aspek mendasar JavaScript, secara bertahap bergerak menuju teknik yang lebih canggih. Ia mengatasi topik-topik kunci seperti variabel, tipe data, struktur kontrol, fungsi, pemrograman berorientasi objek, closure, promises, dan sintaks modern. Setiap bab dibangun di atas bab sebelumnya, memberikan dasar yang kuat bagi para pembelajar dan memfasilitasi pemahaman konsep-konsep yang kompleks.

Salah satu fitur unik dari "Belajar JavaScript" adalah pendekatan praktisnya. Buku ini menawarkan latihan, tantangan pemrograman, dan masalah dunia nyata yang memungkinkan pembaca untuk menerapkan pengetahuan mereka dan mengembangkan keterampilan penting. Dengan berinteraksi dengan contoh-contoh yang nyata, pembaca mendapatkan kepercayaan diri untuk mengatasi masalah umum dalam pengembangan web dan mengungkap potensi JavaScript untuk solusi inovatif.

Ide-ide kompleks seperti closure dan pemrograman asinkron diuraikan melalui penjelasan yang intuitif dan contoh-contoh praktis. Penekanan pada kejelasan dan kesederhanaan memungkinkan pembelajar dari semua tingkatan untuk memahami dan memahami konsep-konsep kunci secara efektif. Buku ini terstruktur dalam tiga bagian, dengan 14 bab pertama membahas konsep-konsep inti. Empat bab berikutnya menguraikan penggunaan JavaScript untuk pemrograman browser web, sedangkan dua bab terakhir mencakup topik-topik lain dan menawarkan latihan. Bagian Lainnya menjelajahi tema dan skenario signifikan yang berkaitan dengan pemrograman JavaScript, diikuti oleh latihan untuk berlatih.

Sebagai kesimpulan, "Belajar JavaScript: Edisi Pemula" adalah sahabat yang sangat penting bagi mereka yang ingin menguasai JavaScript dan berhasil dalam pengembangan web. Dengan cakupan komprehensif, pendekatan praktis, penjelasan yang jelas, fokus aplikasi dunia nyata, dan komitmen untuk pembelajaran berkelanjutan, buku ini berfungsi sebagai sumber daya berharga. Dengan meresapi kontennya, pembaca akan mendapatkan keterampilan dan pengetahuan yang diperlukan untuk membangun aplikasi web yang dinamis dan interaktif, membuka potensi penuh mereka sebagai pengembang JavaScript.

Bab 2

Basics

Dalam bab pertama ini, kita akan mempelajari dasar-dasar pemrograman dan bahasa JavaScript.

Pemrograman berarti menulis kode. Sebuah buku terdiri dari bab, paragraf, kalimat, frasa, kata-kata, dan akhirnya tanda baca dan huruf, begitu juga sebuah program dapat dipecah menjadi komponen-komponen yang lebih kecil. Untuk saat ini, yang paling penting adalah pernyataan. Sebuah pernyataan adalah analog dengan kalimat dalam sebuah buku. Dalam konteksnya sendiri, pernyataan memiliki struktur dan tujuan, tetapi tanpa konteks pernyataan-pernyataan lain di sekitarnya, itu tidak begitu bermakna.

Pernyataan secara lebih santai (dan umum) dikenal sebagai *baris kode*. Itu karena pernyataan cenderung ditulis pada baris-baris individual. Oleh karena itu, program dibaca dari atas ke bawah, dari kiri ke kanan. Anda mungkin bertanya-tanya apa itu kode (juga disebut kode sumber). Itu adalah istilah yang mencakup seluruh program atau bagian terkecilnya. Oleh karena itu, sebuah baris kode hanyalah sebuah baris dari program Anda.

Berikut adalah contoh sederhana:

```
let hello = "Hallo";
let world = "Dunia";

// Pesan sama dengan "Halo Dunia"
let message = hello + " " + world;
```

Kode ini dapat dieksekusi oleh program lain yang disebut *interpreter* yang akan membaca kode tersebut, dan mengeksekusi semua pernyataan dalam urutan yang benar.

Comments

Komentar adalah pernyataan yang tidak akan dieksekusi oleh interpreter. Komentar digunakan untuk menandai catatan bagi para programmer lain atau memberikan deskripsi singkat tentang apa yang dilakukan kode, sehingga memudahkan orang lain memahami apa yang dilakukan kode Anda. Komentar juga digunakan untuk sementara menonaktifkan kode tanpa memengaruhi alur kendali program.

Di JavaScript, komentar dapat ditulis dalam 2 cara berbeda:

- *Komentar satu baris*: Dimulai dengan dua garis miring ke depan (`//`) dan berlanjut hingga akhir baris. Apa pun yang berada setelah garis miring tersebut diabaikan oleh interpreter JavaScript. Contoh:

```
// Ini adalah komentar, akan diabaikan oleh interpreter
let a =
  "Ini adalah sebuah variabel yang didefinisikan dalam sebuah pernyataan.";
```

- *Komentar multi-baris*: Dimulai dengan garis miring dan tanda asterisk (`/*`) dan diakhiri dengan tanda asterisk dan garis miring (`*/`). Apa pun yang berada di antara tanda pembuka dan penutup tersebut diabaikan oleh interpreter JavaScript. Contoh:

```
/*
Ini adalah komentar multi-baris,
akan diabaikan oleh interpreter
*/
let a =
  "Ini adalah sebuah variabel yang didefinisikan dalam sebuah pernyataan.";
```

Termasuk komentar dalam kode sangat penting untuk menjaga kualitas kode, mendorong kolaborasi, dan menyederhanakan proses debugging. Dengan memberikan konteks dan penjelasan untuk berbagai bagian program, komentar membuatnya lebih mudah dipahami dalam jangka waktu yang lebih lama. Oleh karena itu, memasukkan komentar dalam kode dianggap sebagai praktik yang bermanfaat.

Variables

Langkah pertama untuk benar-benar memahami pemrograman adalah melihat kembali aljabar. Jika diingat-ingat di bangku sekolah, aljabar diawali dengan penulisan istilah-istilah seperti berikut ini.

```
3 + 5 = 8
```

Anda mulai melakukan perhitungan ketika Anda memasukkan sesuatu yang tidak diketahui, misalnya x di bawah ini:

```
3 + x = 8
```

Menggeser orang-orang di sekitar Anda dapat menentukan x:

```
x = 8 - 3
-> x = 5
```

Ketika Anda memperkenalkan lebih dari satu, Anda membuat istilah Anda lebih fleksibel - Anda menggunakan variabel:

```
x + y = 8
```

Anda dapat mengubah nilai x dan y dan rumusnya tetap benar:

```
x = 4
y = 4
```

or

```
x = 3
y = 5
```

Hal yang sama berlaku untuk bahasa pemrograman. Dalam pemrograman, variabel adalah wadah untuk nilai-nilai yang berubah. Variabel dapat menampung segala macam nilai dan juga hasil perhitungan. Variabel memiliki `nama` dan `nilai` yang dipisahkan dengan tanda sama dengan (=). Namun, penting untuk diingat bahwa bahasa pemrograman yang berbeda memiliki batasan dan batasannya sendiri mengenai apa yang dapat digunakan sebagai nama variabel. Hal ini karena kata-kata tertentu mungkin dicadangkan untuk fungsi atau operasi tertentu dalam bahasa tersebut.

Mari kita lihat cara kerjanya dalam Javascript. Kode berikut mendefinisikan dua variabel, menghitung hasil penjumlahan keduanya, dan mendefinisikan hasil ini sebagai nilai variabel ketiga.

```
let x = 5;
let y = 6;
let result = x + y;
```

Ada pedoman tertentu yang perlu diikuti saat memberi nama variabel. Mereka

- Nama variabel harus diawali dengan huruf, garis bawah (_), atau tanda dolar ($).
- Setelah karakter pertama, kita dapat menggunakan huruf, angka, garis bawah, atau tanda dolar.

- JavaScript membedakan antara huruf besar dan kecil (peka huruf besar-kecil), jadi myVariable, MyVariable, dan MYVARIABLE semuanya merupakan variabel terpisah.
- Agar kode Anda mudah dibaca dan dipelihara, disarankan untuk menggunakan nama variabel deskriptif yang mencerminkan tujuannya secara akurat.

Exercise

Tentukan variabel `x` sama dengan 20.

```
let x =
```

ES6 Version

ECMAScript 2015 atau ES2015 juga dikenal sebagai E6 adalah pembaruan signifikan pada bahasa pemrograman JavaScript sejak 2009. Di ES6 kami memiliki tiga cara untuk mendeklarasikan variabel.

```
var x = 5;
const y = "Test";
let z = true;
```

Jenis deklarasi bergantung pada cakupannya. Berbeda dengan kata kunci `var`, yang mendefinisikan variabel secara global atau lokal ke seluruh fungsi terlepas dari cakupan bloknya, `let` memungkinkan Anda mendeklarasikan variabel yang cakupannya terbatas pada blok, pernyataan, atau ekspresi di mana variabel tersebut digunakan. Misalnya.

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2; // variabel yang sama
    console.log(x); //2
  }
  console.log(x); //2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2;
    console.log(x); // 2
  }
  console.log(x); // 1
}
```

`const` variabel tidak dapat diubah artinya tidak boleh ditugaskan kembali.

```
const x = "hi!";
x = "selamat tinggal"; // ini akan terjadi kesalahan
```

Types

Komputer sangat canggih dan dapat menggunakan variabel yang lebih kompleks daripada hanya angka. Di sinilah tipe variabel masuk. Variabel ada dalam beberapa jenis, dan berbagai bahasa mendukung jenis yang berbeda.

Jenis yang paling umum adalah:

- **Number**: Angka bisa berupa bilangan bulat (contoh: `1`, `-5`, `100`) atau nilai pecahan (contoh: `3.14`, `-2.5`, `0.01`). JavaScript tidak memiliki tipe terpisah untuk bilangan bulat dan nilai pecahan; JavaScript menganggap keduanya sebagai angka.
- **String**: String adalah urutan karakter, dapat diwakili oleh tanda kutip tunggal (contoh: `'hello'`) atau tanda kutip ganda (contoh: `"world"`).
- **Boolean**: Boolean mewakili nilai benar atau salah. Ini dapat ditulis sebagai `true` atau `false` (tanpa tanda kutip).
- **Null**: Tipe null mewakili nilai null, yang berarti "tidak ada nilai." Ini dapat ditulis sebagai `null` (tanpa tanda kutip).
- **Undefined**: Tipe undefined mewakili nilai yang belum diatur. Jika sebuah variabel telah dideklarasikan tetapi belum diberi nilai, maka nilainya adalah `undefined`.
- **Object**: Objek adalah kumpulan properti, masing-masing memiliki nama dan nilai. Anda dapat membuat objek menggunakan tanda kurung kerawang (`{}`) dan memberikan properti-properti padanya menggunakan pasangan nama-nilai.
- **Array**: Array adalah jenis objek khusus yang dapat menyimpan koleksi item. Anda dapat membuat array menggunakan tanda kurung siku (`[]`) dan memberikan daftar nilai ke dalamnya.
- **Function**: Fungsi adalah blok kode yang dapat didefinisikan dan kemudian dipanggil dengan nama. Fungsi dapat menerima argumen (input) dan mengembalikan nilai (output). Anda dapat membuat fungsi menggunakan kata kunci `function`.

JavaScript adalah bahasa "*loosely typed*," yang berarti Anda tidak perlu secara eksplisit mendeklarasikan jenis data dari variabel. Anda hanya perlu menggunakan kata kunci `var` untuk menunjukkan bahwa Anda mendeklarasikan sebuah variabel, dan interpreter akan menentukan jenis data yang Anda gunakan dari konteks dan penggunaan tanda kutip.

Exercise

Deklarasikan tiga variabel dan inisialisasi dengan nilai berikut: `usia` sebagai angka, `nama` sebagai string, dan `isMarried` sebagai boolean.

```
let age =
let name =
let isMarried =
```

Operator `typeof` digunakan untuk memeriksa tipe data suatu variabel.

```
`typeof "John";` // Mengembalikan "string"
`typeof 3.14;` // Mengembalikan "number"
`typeof NaN;` // Mengembalikan "number"
`typeof false;` // Mengembalikan "boolean"
`typeof [1, 2, 3, 4];` // Mengembalikan "object"
`typeof { name: "John", age: 34 };` // Mengembalikan "object"
`typeof new Date();` // Mengembalikan "object"
`typeof function () {};` // Mengembalikan "function"
`typeof myCar;` // Mengembalikan "undefined" *
`typeof null;`; // Mengembalikan "object"
```

Tipe data yang digunakan dalam JavaScript dapat dibedakan menjadi dua kategori berdasarkan nilai yang dikandungnya.

Tipe data yang dapat berisi nilai:

- `string`
- `number`
- `boolean`
- `object`
- `function`

> `Objek` , `Tanggal` , `Array` , `String` , `Nomor` , dan `Boolean` adalah jenis objek yang tersedia dalam JavaScript.

Tipe data yang tidak dapat berisi nilai:

- `null`
- `undefined` Nilai data primitif adalah nilai data sederhana tanpa properti dan metode tambahan, dan bukan merupakan objek. Mereka tidak dapat diubah, artinya tidak dapat diubah. Ada 7 tipe data primitif:
- string
- number
- bigint
- boolean
- undefined
- symbol
- null

Exercise

Deklarasikan variabel bernama `person` dan inisialisasi dengan objek yang berisi properti berikut: `age` sebagai angka, `name` sebagai string, dan `isMarried` sebagai boolean.

```
let person =
```

Equality

Ketika menulis program, kita sering perlu menentukan kesetaraan variabel dalam hubungannya dengan variabel lain. Hal ini dilakukan menggunakan operator kesetaraan. Operator kesetaraan paling dasar adalah operator `==`. Operator ini berusaha sebisa mungkin untuk menentukan apakah dua variabel setara, bahkan jika mereka bukan tipe yang sama.

Sebagai contoh, anggap:

```
let foo = 42;
let bar = 42;
let baz = "42";
let qux = "life";
```

`foo == bar` akan dievaluasi sebagai `true` dan `baz == qux` akan dievaluasi sebagai `false`, seperti yang diharapkan. Namun, `foo == baz` juga akan dievaluasi sebagai `true` meskipun `foo` dan `baz` memiliki tipe yang berbeda. Di balik layar, operator kesetaraan `==` berusaha memaksa operan-operannya memiliki tipe yang sama sebelum menentukan kesetaraan mereka. Ini berbeda dengan operator kesetaraan `===`.

Operator kesetaraan `===` menentukan bahwa dua variabel setara jika mereka memiliki tipe yang sama *dan* memiliki nilai yang sama. Dengan asumsi yang sama seperti sebelumnya, ini berarti bahwa `foo === bar` masih akan dievaluasi sebagai `true`, tetapi `foo === baz` sekarang akan dievaluasi sebagai `false`. `baz === qux` masih akan dievaluasi sebagai `false`.

Exercise

Gunakan operator `==` dan `===` untuk membandingkan nilai `str1` dan `str2`.

```
let str1 = "halo";
let str2 = "HALLO";
let bool1 = true;
let bool2 = 1;
// membandingkan menggunakan ==
let stringResult1 =
let boolResult1 =
// membandingkan menggunakan ==
let stringResult1 =
let boolResult2 =
```

Bab 3

Angka

JavaScript memiliki **hanya satu jenis angka** - titik apung 64 bit. Ini sama dengan `double` di Java. Berbeda dengan sebagian besar bahasa pemrograman lain, tidak ada tipe bilangan bulat terpisah, jadi 1 dan 1.0 adalah nilai yang sama. Membuat angka cukup mudah, bisa dilakukan seperti tipe variabel lainnya dengan menggunakan kata kunci `var`.

Angka dapat dibuat dari nilai konstan:

```
// Ini adalah angka desimal:
let a = 1.2;

// Ini adalah angka bulat:
let b = 10;
```

Atau, dari nilai variabel lain:

```
let a = 2;
let b = a;
```

Presisi bilangan bulat akurat hingga 15 digit dan angka maksimum adalah 17.

```
let x = 999999999999999; // x akan menjadi 999999999999999
let y = 9999999999999999; // y akan menjadi 10000000000000000
```

Konstanta numerik diinterpretasikan sebagai heksadesimal jika diawali dengan `0x`.

```
let z = 0xff; // 255
```

Math

Objek `Math` memungkinkan untuk melakukan operasi matematika dalam JavaScript. Objek ini statis dan tidak memiliki konstruktor. Seseorang dapat menggunakan metode dan properti dari objek `Math` tanpa membuat objek `Math` terlebih dahulu. Untuk mengakses propertinya, seseorang dapat menggunakan *Math.property.* Beberapa properti matematika dijelaskan di bawah ini:

```
Math.E; // mengembalikan angka Euler
Math.PI; // mengembalikan PI
Math.SQRT2; // mengembalikan akar kuadrat dari 2
Math.SQRT1_2; // mengembalikan akar kuadrat dari 1/2
Math.LN2; // mengembalikan logaritma natural dari 2
Math.LN10; // mengembalikan logaritma natural dari 10
Math.LOG2E; // mengembalikan logaritma basis 2 dari E
Math.LOG10E; // mengembalikan logaritma basis 10 dari E
```

Contoh beberapa metode matematika adalah:

```
Math.pow(8, 2); // 64
Math.round(4.6); // 5
Math.ceil(4.9); // 5
Math.floor(4.9); // 4
Math.trunc(4.9); // 4
Math.sign(-4); // -1
Math.sqrt(64); // 8
Math.abs(-4.7); // 4.7
Math.sin((90 * Math.PI) / 180); // 1 (sudut sin 90 derajat)
Math.cos((0 * Math.PI) / 180); // 1 (kosinus sudut 0 derajat)
Math.min(0, 150, 30, 20, -8, -200); // -200
Math.max(0, 150, 30, 20, -8, -200); // 150
Math.random(); // 0.44763808380924375
Math.log(2); // 0.6931471805599453
Math.log2(8); // 3
Math.log10(1000); // 3
```

Untuk mengakses metode matematika, seseorang dapat memanggil metodenya langsung dengan argumen di mana diperlukan.

Metode	Deskripsi
`abs(x)`	Mengembalikan nilai absolut dari `x`
`acos(x)`	Mengembalikan arkkosinus dari `x` , dalam radian
`acosh(x)`	Mengembalikan arkkosinus hiperbolik dari `x`
`asin(x)`	Mengembalikan arksinus dari `x` , dalam radian
`asinh(x)`	Mengembalikan arksinus hiperbolik dari `x`
`atan(x)`	Mengembalikan arktangen dari `x` sebagai nilai numerik antara `-PI/2` dan `PI/2` radian
`atan2(y,x)`	Mengembalikan arktangen dari hasil bagi argumennya
`atanh(x)`	Mengembalikan arktangen hiperbolik dari `x`
`cbrt(x)`	Mengembalikan akar kubik dari `x`
`ceil(x)`	Mengembalikan pembulatan ke atas ke bilangan bulat terdekat dari `x`
`cos(x)`	Mengembalikan kosinus dari `x` , dalam radian
`cosh(x)`	Mengembalikan kosinus hiperbolik dari `x`
`exp(x)`	Mengembalikan nilai eksponensial dari `x`
`floor(x)`	Mengembalikan pembulatan ke bawah ke bilangan bulat terdekat dari `x`
`log(x)`	Mengembalikan logaritma alami dari `x`
`max(x,y,z,...n)`	Mengembalikan angka dengan nilai tertinggi
`min(x,y,z,...n)`	Mengembalikan angka dengan nilai terendah
`pow(x,y)`	Mengembalikan nilai `x` pangkat `y`
`random()`	Mengembalikan angka antara 0 dan 1
`round(x)`	Membulatkan angka ke angka terdekat `x`
`sign(x)`	Mengembalikan apakah `x` negatif, `null` , atau positif (-1, 0, 1)
`sin(x)`	Mengembalikan sin dari `x` , dalam radian
`sinh(x)`	Mengembalikan sin hiperbolik dari `x`
`sqrt(x)`	Mengembalikan akar kuadrat dari `x`
`tan(x)`	Mengembalikan tangen dari suatu sudut
`tanh(x)`	Mengembalikan tangen hiperbolik dari `x`
`trunc(x)`	Mengembalikan bagian bilangan bulat dari suatu angka (`x`)

Operator Dasar

Dalam JavaScript, operator adalah simbol atau kata kunci yang digunakan untuk melakukan operasi pada operan (nilai dan variabel). Sebagai contoh,

```
2 + 3; //5
```

Di sini `+` adalah operator yang melakukan penambahan, dan `2` dan `3` adalah operan.

Jenis Operator

Ada berbagai operator yang didukung oleh JavaScript. Mereka adalah sebagai berikut:

- Operator Aritmatika
- Operator Penugasan
- Operator Perbandingan
- Operator Logika
- Operator Ternary
- Operator Bitwise
- Operator `typeof`

Operator Aritmatika

Operator aritmatika digunakan untuk melakukan operasi matematika pada nilai. Mereka meliputi

- Operator Penambahan (`+`)
- Operator Pengurangan (`-`)
- Operator Perkalian (`*`)
- Operator Pembagian (`/`)
- Operator Sisa (`%`)

Operator Penambahan (`+`)

Operator penambahan menambahkan dua angka. Sebagai contoh:

```
console.log(1 + 2); // 3
console.log(1 + -2); // -1
```

Operator Pengurangan (`-`)

Operator pengurangan mengurangkan satu angka dari yang lain. Sebagai contoh:

```
console.log(3 - 2); // 1
console.log(3 - -2); // 5
```

Operator Perkalian (`*`)

Operator perkalian mengalikan dua angka. Sebagai contoh:

```
console.log(2 * 3); // 6
console.log(2 * -3); // -6
```

Operator Pembagian (`/`)

Operator pembagian membagi satu angka dengan yang lain. Sebagai contoh:

```
console.log(6 / 2); // 3
console.log(6 / -2); // -3
```

Operator Sisa (`%`)

Operator sisa mengembalikan sisa dari operasi pembagian. Sebagai contoh:

```
console.log(10 % 3); // 1
console.log(11 % 3); // 2
console.log(12 % 3); // 0
```

Penerjemah JavaScript bekerja dari kiri ke kanan. Seseorang dapat menggunakan tanda kurung seperti dalam matematika untuk memisahkan dan mengelompokkan ekspresi: `c = (a / b) + d`

> JavaScript menggunakan operator `+` baik untuk penambahan maupun penggabungan (konkatenasi). Angka ditambahkan sementara string digabungkan.

Istilah `NaN` adalah kata yang dipesan yang menunjukkan bahwa sebuah angka bukan angka yang sah, ini muncul ketika kita melakukan operasi aritmatika dengan string yang bukan angka akan menghasilkan `NaN` (Not a Number).

```
let x = 100 / "10";
```

Metode `parseInt` mem-parsing nilai sebagai string dan mengembalikan bilangan bulat pertama.

```
parseInt("10"); // 10
parseInt("10.00"); // 10
parseInt("10.33"); // 10
parseInt("34 45 66"); // 34
parseInt(" 60 "); // 60
parseInt("40 years"); //40
parseInt("He was 40"); //NaN
```

Dalam JavaScript, jika kita menghitung angka di luar angka terbesar yang mungkin, ia akan mengembalikan `Infinity`.

```
let x = 2 / 0; // Infinity
let y = -2 / 0; // -Infinity
```

Exercise

Gunakan operator matematika +, -, *, /, dan % untuk melakukan operasi berikut pada `num1` dan `num2`.

```
let num1 = 10;
let num2 = 5;

// Tambahkan num1 dan num2.
let addResult =
// Kurangkan num2 dari num1.
let subtractResult =
// Kalikan num1 dan num2.
let multiplyResult =
// Bagi num1 oleh num2.
let divideResult =
// Temukan sisa num1 dibagi num2.
let reminderResult =
```

Operator Penugasan

Operator penugasan digunakan untuk memberi nilai ke variabel atau mengevaluasi nilai yang diberikan. *Penggabungan operator penugasan mungkin untuk memberikan satu nilai ke beberapa nilai.* Mereka mencakup operator penugasan (`=`) dan operator penugasan gabungan seperti `+=` , `-=` , `*=` , dan `/=` .

`=` (Operator Penugasan)

Operator ini digunakan untuk memberikan nilai di sebelah kanan ke variabel di sebelah kiri. Contoh:

```
let x = 10; //Memberikan nilai 10 ke variabel x.
```

Operator Penugasan Gabungan

Operator-operator ini menggabungkan operasi aritmatika dengan operasi penugasan. Mereka adalah pintasan untuk melakukan operasi dan kemudian memberikan hasilnya kembali ke variabel. Sebagai contoh:

`+=` (Penugasan Penambahan)

Ini menambahkan nilai di sebelah kanan ke variabel dan memberikan hasilnya kembali ke variabel.

`-=` (Penugasan Pengurangan)

Ini mengurangkan nilai di sebelah kanan dari variabel dan memberikan hasilnya kembali ke variabel.

`*=-` (Penugasan Perkalian)

Ini mengalikan variabel dengan nilai di sebelah kanan dan memberikan hasilnya kembali ke variabel.

`/=` (Penugasan Pembagian)

Ini membagi variabel dengan nilai di sebelah kanan dan memberikan hasilnya kembali ke variabel.

`%=` (Penugasan Sisa)

Ini menghitung sisa ketika variabel dibagi dengan nilai di sebelah kanan dan memberikan hasilnya kembali ke variabel.

```
let a = 10;
a += 5; // Setara dengan a = a + 5; (a menjadi 15)
a -= 3; // Setara dengan a = a - 3; (a menjadi 12)
a *= 2; // Setara dengan a = a * 2; (a menjadi 24)
a /= 4; // Setara dengan a = a / 4; (a menjadi 6)
a %= 5; // Setara dengan a = a % 5; (a menjadi 1)
```

Operator Perbandingan

Operator perbandingan digunakan untuk membandingkan dua nilai atau ekspresi dan mengembalikan hasil `boolean`, yang bisa berupa `true` atau `false`. Operator-operator ini umumnya digunakan dalam pernyataan kondisional untuk membuat keputusan atau mengevaluasi kondisi.

Sama dengan (`==`)

Operator ini memeriksa apakah nilai di sebelah kiri dan kanan adalah sama. Jika mereka sama, ia mengembalikan `true`, jika tidak, ia mengembalikan `false`. Ini tidak mempertimbangkan tipe data.

```
5 == 5; // true
"5" == 5; // true (konversi tipe data implisit)
```

Tidak sama dengan (`!=`)

Operator ini memeriksa apakah nilai di sebelah kiri dan kanan tidak sama. Jika mereka tidak sama, ia mengembalikan `true`, jika tidak, ia mengembalikan `false`.

```
5 != 3; // true
"5" != 5; // false (konversi tipe data implisit)
```

Sama dengan secara ketat (`===`)

Operator ini memeriksa apakah nilai di sebelah kiri dan kanan sama dan memiliki tipe data yang sama. Jika nilai dan tipe data keduanya cocok, ia mengembalikan `true`, jika tidak, ia mengembalikan `false`.

```
5 === 5; // true
"5" === 5; // false (tipe data yang berbeda)
```

Tidak sama dengan secara ketat (`!==`)

Operator ini memeriksa apakah nilai di sebelah kiri dan kanan tidak sama atau memiliki tipe data yang berbeda. Jika mereka tidak sama atau memiliki tipe data yang berbeda, ia mengembalikan `true`, jika tidak, ia mengembalikan `false`.

```
5 !== "5"; // true (tipe data yang berbeda)
5 !== 5; // false
```

Lebih besar dari (`>`)

Operator ini memeriksa apakah nilai di sebelah kiri lebih besar dari nilai di sebelah kanan. Jika nilai di sebelah kiri lebih besar, ia mengembalikan `true`, jika tidak, ia mengembalikan `false`.

```
8 > 5; // true
3 > 10; // false
```

Lebih kecil dari (`<`)

Operator ini memeriksa apakah nilai di sebelah kiri lebih kecil dari nilai di sebelah kanan. Jika nilai di sebelah kiri lebih kecil, ia mengembalikan `true`, jika tidak, ia mengembalikan `false`.

```
3 < 5; // true
8 < 2; // false
```

Lebih besar dari atau sama dengan (`>=`)

Operator ini memeriksa apakah nilai di sebelah kiri lebih besar atau sama dengan nilai di sebelah kanan. Jika nilai di sebelah kiri lebih besar atau sama, ia mengembalikan `true`, jika tidak, ia mengembalikan `false`.

```
8 >= 5; // true
3 >= 8; // false
```

Lebih kecil dari atau

sama dengan (`<=`)

Operator ini memeriksa apakah nilai di sebelah kiri lebih kecil atau sama dengan nilai di sebelah kanan. Jika nilai di sebelah kiri lebih kecil atau sama, ia mengembalikan `true`, jika tidak, ia mengembalikan `false`.

```
3 <= 5; // true
8 <= 2; // false
```

Operator Logika

Operator logika digunakan untuk melakukan operasi logika pada nilai Boolean atau ekspresi Boolean. Operator-operator ini memungkinkan Anda untuk menggabungkan atau memanipulasi nilai Boolean untuk membuat keputusan atau mengevaluasi kondisi yang kompleks.

DAN Logika (`&&`)

Operator logika AND mengembalikan `true` jika kedua operan adalah `true`. Jika setidaknya salah satu dari operan adalah `false`, ia mengembalikan `false`.

```
true && true; // true
true && false; // false
false && true; // false
false && false; // false
```

ATAU Logika (`||`)

Operator logika OR mengembalikan `true` jika setidaknya satu dari operan adalah `true`. Ia mengembalikan `false` hanya jika kedua operan adalah `false`.

```
true || true; // true
true || false; // true
false || true; // true
false || false; // false
```

TIDAK Logika (`!`)

Operator logika NOT membalikkan nilai dari operan. Ia mengembalikan `true` jika operan adalah `false`, dan ia mengembalikan `false` jika operan adalah `true`.

```
!true; // false
!false; // true
```

Operator Ternary

Operator ternary memiliki tiga operan. Ini adalah bentuk singkat dari kondisi `if/else`.

Ini adalah bentuk singkat dari kondisi `if-else`.

Sintaks

```
Y =  ? A : B
Jika kondisinya benar maka Y = A, jika tidak Y = B
```

```
let isEven = 8 % 2 === 0 ? "Genap" : "Ganjil";
console.log(isEven); // "Genap"
```

Operator Bitwise

Operator bitwise digunakan untuk memanipulasi bit individual dari angka biner. Mereka melakukan operasi pada level bit, yang sangat berguna dalam situasi di mana Anda perlu mengendalikan atau menganalisis data tingkat rendah.

Bitwise AND (`&`)

Operator ini membandingkan setiap bit dari dua angka dan mengembalikan 1 untuk setiap bit yang bernilai 1 pada kedua angka. Semua bit lainnya diatur ke 0.

```
1010 & 1100; // 1000
```

Bitwise OR (`|`)

Operator ini membandingkan setiap bit dari dua angka dan mengembalikan 1 untuk setiap bit yang bernilai 1 pada setidaknya salah satu dari angka tersebut.

```
1010 | 1100; // 1110
```

Bitwise XOR (`^`)

Operator ini membandingkan setiap bit dari dua angka dan mengembalikan 1 untuk setiap bit yang bernilai 1 pada salah satu angka tetapi tidak pada keduanya.

```
1010 ^ 1100; // 0110
```

Bitwise NOT (`~`)

Operator ini membalikkan (membalik) semua bit dari sebuah angka. Ini mengubah setiap 0 menjadi 1 dan setiap 1 menjadi 0.

```
~1010; // 0101
```

Pergeseran Kiri (`<<`)

Operator ini menggeser bit dari sebuah angka ke kiri sebanyak posisi tertentu, mengisi posisi yang digeser dengan 0.

```
1010 << 2; // 101000 (dipindahkan ke kiri sebanyak 2 posisi)
```

Pergeseran Kanan (`>>`)

Operator ini menggeser bit dari sebuah angka ke kanan sebanyak posisi tertentu. Posisi yang digeser diisi berdasarkan bit terkiri (bit tanda).

```
1010 >> 2; // 0010 (dipindahkan ke kanan sebanyak 2 posisi)
```

Operator `typeof`

Ini mengembalikan tipe operan, Tipe yang mungkin ada dalam javascript adalah undefined, Object, boolean, number, string, symbol, dan function.

```
let nilai1 = 42;
let nilai2 = "Halo, Dunia!";
let nilai3 = true;
let nilai4 = null;

console.log(typeof nilai1); // "number"
console.log(typeof nilai2); // "string"
console.log(typeof nilai3); // "boolean"
console.log(typeof nilai4); // "object" (Catatan: `typeof null` mengembalikan "object" karena alasan sejarah)
```

Operator Canggih

Ketika operator-operator digabungkan tanpa tanda kurung, urutan di mana mereka diterapkan ditentukan oleh *precedence* (kedahsyatan) operator. Perkalian `(*)` dan pembagian `(/)` memiliki prioritas yang lebih tinggi daripada penambahan `(+)` dan pengurangan `(-)`.

```
// perkalian dilakukan terlebih dahulu, diikuti oleh penambahan
let x = 100 + 50 * 3; // 250
// dengan tanda kurung, operasi di dalam tanda kurung dihitung terlebih dahulu
let y = (100 + 50) * 3; // 450
// operasi dengan prioritas yang sama dihitung dari kiri ke kanan
let z = 100 / 50 * 3;
```

Ada beberapa operator matematika canggih yang dapat digunakan saat menulis program. Berikut adalah daftar beberapa operator matematika canggih utama:

- **Operator Modulo (`%`)**: Operator modulo mengembalikan sisa dari operasi pembagian. Sebagai contoh:

```
console.log(10 % 3); // 1
console.log(11 % 3); // 2
console.log(12 % 3); // 0
```

- **Operator Pemangkatan (`)**: Operator pemangkatan mengangkat suatu angka ke pangkat angka lain. Ini adalah operator yang lebih baru dan tidak didukung di semua peramban, sehingga Anda mungkin perlu menggunakan fungsi Math.pow` sebagai gantinya. Sebagai contoh:

```
console.log(2 ** 3); // 8
console.log(3 ** 2); // 9
console.log(4 ** 3); // 64
```

- **Operator Peningkatan (`++`)**: Operator peningkatan menambahkan satu pada suatu angka. Ini dapat digunakan sebagai awalan (sebelum operand) atau akhiran (setelah operand). Sebagai contoh:

```
let x = 1;
x++; // x sekarang 2
++x; // x sekarang 3
```

- **Operator Penurunan (`--`)**: Operator penurunan mengurangkan satu pada suatu angka. Ini dapat digunakan sebagai awalan (sebelum operand) atau akhiran (setelah operand). Sebagai contoh:

```
let y = 3;
y--; // y sekarang 2
--y; // y sekarang 1
```

- **Objek Math**: Objek `Math` adalah objek bawaan dalam JavaScript yang menyediakan fungsi dan konstanta matematika. Anda dapat menggunakan metode objek `Math` untuk melakukan operasi matematika canggih, seperti mencari akar kuadrat suatu angka, menghitung sinus suatu angka, atau menghasilkan angka acak. Sebagai contoh:

```
console.log(Math.sqrt(9)); // 3
console.log(Math.sin(0)); // 0
console.log(Math.random()); // angka acak antara 0 dan 1
```

Ini hanya beberapa contoh dari operator-operator matematika canggih dan fungsi-fungsi yang tersedia dalam JavaScript. Ada banyak lagi yang bisa Anda gunakan untuk melakukan operasi matematika canggih saat menulis program.

Exercise

Gunakan operator-operator canggih berikut untuk melakukan operasi pada `num1` dan `num2`.

```
let num1 = 10;
let num2 = 5;

// Operator ++ untuk menambahkan nilai num1.
const result1 =
// Operator -- untuk mengurangkan nilai num2.
const result2 =
// Operator += untuk menambahkan num2 ke num1.
const result3 =
// Operator -= untuk mengurangkan num2 dari num1.
const result4 =
```

Operator Penggabungan Nullish '??'

Operator penggabungan `nullish` mengembalikan argumen pertama jika tidak `null/undefined`, jika tidak, maka argumen kedua. Ditulis dengan dua tanda tanya `??`. Hasil dari `x ?? y` adalah:

- Jika `x` terdefinisi, maka `x`,
- Jika `y` tidak terdefinisi, maka `y`.

Ini adalah penambahan terbaru dalam bahasa ini dan mungkin memerlukan polyfill untuk mendukung peramban lama.

Bab 4

String

String dalam JavaScript memiliki banyak kesamaan dengan implementasi string dalam bahasa pemrograman tingkat tinggi lainnya. Mereka mewakili pesan dan data berbasis teks. Dalam kursus ini, kita akan membahas dasar-dasar penggunaan string, bagaimana cara membuat string baru, dan melakukan operasi umum pada string.

Berikut adalah contoh string:

```
"Halo Dunia"
```

Indeks string dimulai dari nol, yang berarti bahwa posisi awal karakter pertama adalah `0`, diikuti oleh karakter lainnya dengan peningkatan urutan. Berbagai metode didukung oleh string dan mengembalikan nilai baru. Metode-metode ini dijelaskan di bawah ini.

Nama	Deskripsi
`charAt()`	Mengembalikan karakter pada indeks tertentu
`charCodeAt()`	Mengembalikan karakter Unicode pada indeks tertentu
`concat()`	Mengembalikan dua atau lebih string yang digabungkan
`constructor`	Mengembalikan fungsi konstruktor string
`endsWith()`	Memeriksa apakah sebuah string diakhiri dengan nilai tertentu
`fromCharCode()`	Mengembalikan nilai Unicode sebagai karakter
`includes()`	Memeriksa apakah sebuah string mengandung nilai tertentu
`indexOf()`	Mengembalikan indeks kemunculan pertama
`lastIndexOf()`	Mengembalikan indeks kemunculan terakhir
`length`	Mengembalikan panjang string
`localeCompare()`	Membandingkan dua string dengan urutan lokal
`match()`	Menyesuaikan string dengan sebuah nilai atau ekspresi reguler
`prototype`	Digunakan untuk menambahkan properti dan metode dari sebuah objek
`repeat()`	Mengembalikan string baru dengan jumlah salinan yang ditentukan
`replace()`	Mengembalikan string dengan nilai yang diganti oleh ekspresi reguler atau string lainnya
`search()`	Mengembalikan indeks berdasarkan kesesuaian string dengan sebuah nilai atau ekspresi reguler
`slice()`	Mengembalikan string yang berisi sebagian dari string
`split()`	Memisahkan string menjadi array substring
`startsWith()`	Memeriksa apakah string dimulai dengan karakter tertentu
`substr()`	Mengekstrak bagian dari string, dimulai dari indeks awal
`substring()`	Mengekstrak bagian dari string, antara dua indeks
`toLocaleLowerCase()`	Mengembalikan string dengan karakter huruf kecil menggunakan bahasa tuan rumah
`toLocaleUpperCase()`	Mengembalikan string dengan karakter huruf besar menggunakan bahasa tuan rumah
`toLowerCase()`	Mengembalikan string dengan karakter huruf kecil
`toString()`	Mengembalikan string atau objek string sebagai string
`toUpperCase()`	Mengembalikan string dengan karakter huruf besar
`trim()`	Mengembalikan string dengan spasi yang dihapus
`trimEnd()`	Mengembalikan string dengan spasi yang dihapus dari akhir
`trimStart()`	Mengembalikan string dengan spasi yang dihapus dari awal
`valueOf()`	Mengembalikan nilai primitif dari string atau objek string

Creation

String dapat didefinisikan dengan mengapit teks dengan tanda kutip tunggal atau tanda kutip ganda:

```
// Anda dapat menggunakan tanda kutip tunggal
let str = "String kita yang indah";

// Dan juga tanda kutip ganda
let otherStr = "String lain yang bagus";
```

Dalam Javascript, String dapat berisi karakter UTF-8:

```
"中文 español English हिन्दी العربية português বাংলা русский 日本語 ਪੰਜਾਬੀ 한국어";
```

Anda juga dapat menggunakan konstruktor `String` untuk membuat objek string:

```
const stringObject = new String("Ini adalah string");
```

Namun, umumnya tidak disarankan untuk menggunakan konstruktor `String` untuk membuat string, karena hal ini dapat menyebabkan kebingungan antara string primitif dan objek string. Lebih baik menggunakan literal string untuk membuat string.

Anda juga dapat menggunakan literal template untuk membuat string. Literal template adalah string yang diapit oleh backtick (`` `) dan dapat berisi placeholder untuk nilai. Placeholder ditandai dengan sintaks `` ${}` ``.

```
const name = "John";
const message = `Halo, ${name}!`;
```

Literal template juga dapat berisi beberapa baris dan dapat mencakup ekspresi apa pun di dalam placeholder.

> String tidak dapat dikurangkan, dikalikan, atau dibagi.

Ganti

Metode `replace` memungkinkan kita untuk mengganti karakter, kata, atau kalimat dengan sebuah string. Sebagai contoh.

```
let str = "Halo Dunia!";
let new_str = str.replace("Halo", "Hi");

console.log(new_str);

// Hasil: Hi Dunia!
```

> Untuk mengganti nilai pada semua instansi dari ekspresi reguler, tambahkan modifikasi `g` .

Ini mencari sebuah string untuk nilai atau ekspresi reguler, dan mengembalikan string baru dengan nilai yang diganti. Ini tidak mengubah string asli. Mari kita lihat contoh penggantian global dan tanda besar/ketidak-pekaan huruf.

```
let teks = "Pak Biru memiliki rumah biru dan mobil biru";
let hasil = teks.replace(/biru/gi, "merah");

console.log(hasil);
//Hasil: Pak Merah memiliki rumah merah dan mobil merah
```

Panjang

Menghitung berapa banyak karakter dalam sebuah string sangat mudah dalam Javascript menggunakan properti `.length`. Properti `length` mengembalikan jumlah karakter dalam string, termasuk spasi dan karakter khusus.

```
let ukuran = "String indah kita".length;
console.log(ukuran);
// ukuran: 17

let ukuranStringKosong = "".length;
console.log(ukuranStringKosong);
// ukuranStringKosong: 0
```

Properti panjang dari string kosong adalah `0`.

> Properti `length` adalah properti hanya-baca, sehingga Anda tidak dapat memberikan nilai baru kepadanya.

Concatenation

Penggabungan melibatkan penggabungan dua atau lebih string bersama, menciptakan string yang lebih besar yang berisi data yang digabungkan dari string asli. Penggabungan string adalah menambahkan satu atau lebih string ke string lain. Ini dilakukan dalam JavaScript dengan cara berikut.

- menggunakan operator `+`
- menggunakan metode `concat()`
- menggunakan metode `join()` array
- menggunakan literal template (diperkenalkan dalam ES6)

Metode string `concat()` menerima daftar string sebagai parameter dan mengembalikan string baru setelah penggabungan, yaitu gabungan dari semua string tersebut. Sementara metode array `join()` digunakan untuk menggabungkan semua elemen yang ada dalam sebuah array dengan mengubahnya menjadi satu string.

Literal template menggunakan tanda backtick (`` ` ``) dan menyediakan cara mudah untuk membuat string multi-baris dan melakukan interpolasi string. Ekspresi dapat digunakan di dalam backtick menggunakan tanda $ dan kurung kurawal ${ekspresi}`.

```
const icon = "👋";
// menggunakan Template Strings
`hi ${icon}`;

// menggunakan Metode join()
["hi", icon].join(" ");

// menggunakan Metode concat()
"".concat("hi ", icon);

// menggunakan operator +
"hi " + icon;
// hi 👋
```

Pecah

Metode `split()` membagi sebuah string menjadi daftar substring dan mengembalikannya sebagai array.

- menggunakan metode `split()`
- menggunakan template literal (diperkenalkan dalam ES6)

Metode split() membutuhkan:

separator (opsional) - Pola (string atau ekspresi reguler) yang menjelaskan di mana setiap pemecahan harus terjadi. limit (opsional) - Bilangan bulat non-negatif yang membatasi jumlah potongan untuk memecah string yang diberikan.

```
console.log("ABCDEF".split("")); // [ 'A', 'B', 'C', 'D', 'E', 'F' ]

const text = "Java keren. Java seru.";

let pola = ".";
let teksBaru = text.split(pola);
console.log(teksBaru); // [ 'Java keren', ' Java seru', '' ]

let pola1 = ".";
// hanya memecah string hingga maksimum dua bagian
let teksBaru1 = text.split(pola1, 2);
console.log(teksBaru1); // [ 'Java keren', ' Java seru' ]

const text2 = "JavaScript ;  Python ;C;C++";
let pola2 = ";";
let teksBaru2 = text2.split(pola2);
console.log(teksBaru2); // [ 'JavaScript ', '  Python ', 'C', 'C++' ]

// menggunakan RegEx
let pola3 = /\s*(?:;|$)\s*/;
let teksBaru3 = text2.split(pola3);
console.log(teksBaru3); // [ 'JavaScript', 'Python', 'C', 'C++' ]

//Output
["A", "B", "C", "D", "E", "F"][("Java keren", " Java seru", "")][
  ("Java keren", " Java seru")
][("JavaScript ", "  Python ", "C", "C++")][
  ("JavaScript", "Python", "C", "C++")
];
```

Bab 5

Conditional Logic

Sebuah kondisi adalah pengujian terhadap sesuatu. Kondisi sangat penting dalam pemrograman, dalam beberapa cara:

Pertama, kondisi dapat digunakan untuk memastikan bahwa program Anda berfungsi, terlepas dari data apa pun yang Anda gunakan untuk pemrosesan. Jika Anda hanya mengandalkan data tanpa melakukan pengujian, Anda akan mengalami masalah dan program Anda akan gagal. Jika Anda menguji apakah tindakan yang ingin Anda lakukan mungkin dan apakah semua informasi yang diperlukan ada dalam format yang benar, masalah tersebut tidak akan terjadi, dan program Anda akan lebih stabil. Mengambil tindakan pencegahan seperti ini juga dikenal sebagai pemrograman defensif.

Kondisi juga dapat digunakan untuk menciptakan cabang dalam logika program. Anda mungkin pernah bertemu dengan diagram cabang sebelumnya, misalnya ketika mengisi formulir. Pada dasarnya, ini mengacu pada mengeksekusi "cabang" (bagian) kode yang berbeda, tergantung pada apakah kondisi terpenuhi atau tidak.

Di bab ini, kita akan mempelajari dasar-dasar logika kondisional dalam JavaScript.

If

Kondisi termudah adalah pernyataan if dan sintaksnya adalah `if(kondisi){ lakukan ini … }` . Kondisi tersebut harus benar agar kode di dalam kurung kurawal dapat dieksekusi. Misalnya, Anda dapat menguji sebuah string dan menetapkan nilai string lain bergantung pada nilainya seperti dijelaskan di bawah.

```
let country = "France";
let weather;
let food;
let currency;

if (country === "England") {
  weather = "mengerikan";
  food = "memuaskan";
  currency = "pound sterling";
}

if (country === "France") {
  weather = "bagus";
  food = "Menakjubkan, tetapi jarang sekali vegetarian";
  currency = "lucu, kecil, dan berwarna-warni";
}

if (country === "Germany") {
  weather = "rata-rata";
  food = "hal terburuk yang pernah ada";
  currency = "lucu, kecil, dan berwarna-warni";
}

let message =
  "Ini adalah " +
  country +
  ", Cuacanya adalah " +
  weather +
  ", Makanannya adalah " +
  food +
  " dan " +
  "mata uangnya adalah " +
  currency;

console.log(message);
// 'Ini adalah France, Cuacanya adalah bagus, makanannya adalah Menakjubkan, tetapi jarang sekali vegetarian da
```

Nested If-Else

Di JavaScript, Anda dapat menggunakan pernyataan `if-else` bersarang untuk membuat logika kondisional yang lebih kompleks.

Basic Syntax

```
if (condition1) {
  // Kode yang akan dijalankan ketika condition1 adalah benar
} else {
  if (condition2) {
    // Kode yang akan dijalankan ketika condition1 adalah salah dan condition2 adalah benar
  } else {
    // Kode yang akan dijalankan ketika baik condition1 maupun condition2 adalah salah
  }
}
```

Program berikut menentukan status seorang siswa berdasarkan usia mereka dan mencetak pesan yang sesuai.

```
let age = 20;
let isStudent = true;

if (age >= 18) {
  if (isStudent) {
 console.log("Anda adalah siswa dewasa.");

  } else {
console.log("Anda adalah seorang dewasa, tetapi bukan seorang siswa.");

  }
} else {
console.log("Anda bukan seorang dewasa.");

}

// Output: Anda adalah seorang siswa dewasa.
```

Program ini memeriksa hujan, suhu, dan salju untuk memberikan saran cuaca.

```
let temperature = 25;
let isRaining = true;
let isSnowing = false;

if (isRaining) {
  console.log("Sedang hujan. Jangan lupa bawa payung.");

  if (temperature < 10) {
    console.log("Dan cuacanya dingin. Anda mungkin perlu membawa jaket juga.");
  }
} else if (isSnowing) {
  console.log("Sedang bersalju. Bersiaplah untuk jalan yang licin.");
} else {
  console.log("Tidak ada hujan atau salju. Nikmati cuacanya!");
}

// Output: Sedang hujan. Jangan lupa bawa payung.
```

Program ini memeriksa usia seseorang, pengalaman berkendara sebelumnya, dan status ujian tulis untuk menentukan kelayakan mendapatkan izin mengemudi.

```
let age = 19;
let hasPriorExperience = true;
let hasPassedWrittenTest = true;

if (age >= 18) {
  if (hasPriorExperience) {
    console.log("Selamat! Anda memenuhi syarat untuk mendapatkan izin mengemudi.");
  } else {
    console.log("Maaf, Anda memerlukan pengalaman berkendara sebelumnya untuk mendapatkan izin mengemudi.");
  }
} else {
  console.log("Maaf, Anda harus berusia 18 tahun atau lebih untuk mengajukan izin mengemudi.");

  if (hasPassedWrittenTest) {
    console.log("Anda telah lulus ujian tulis, tetapi Anda harus menunggu hingga berusia 18 tahun untuk mengaju
  } else {
    console.log("Anda perlu lulus ujian tulis terlebih dahulu dan menunggu hingga berusia 18 tahun untuk mengaj
  }
}

// Output: Selamat! Anda memenuhi syarat untuk mendapatkan izin mengemudi.
```

Else

Terdapat juga klausa `else` yang akan diterapkan ketika kondisi pertama tidak benar. Ini sangat berguna jika Anda ingin merespons berbagai nilai, tetapi ingin memperlakukan satu nilai tertentu secara khusus.

```
let umbrellaMandatory;

if (country === "England") {
  umbrellaMandatory = true;
} else {
  umbrellaMandatory = false;
}
```

Klausa `else` dapat digabungkan dengan `if` lainnya. Mari kita buat ulang contoh dari artikel sebelumnya.

```
if (country === "England") {
  ...
} else if (country === "France") {
  ...
} else if (country === "Germany") {
  ...
}
```

Exercise

Dari nilai berikut tulis pernyataan kondisional yang memeriksa apakah `angka1` lebih besar dari `angka2`. Jika ya, tetapkan "angka1 lebih besar dari angka2" ke variabel `hasil`. Jika tidak, tetapkan "angka1 kurang dari atau sama dengan angka2".

```
let num1 = 10;
let num2 = 5;
let result;
// periksa apakah angka1 lebih besar dari angka2
if( condition ) {

}else {

}
```

Switch

`switch` adalah pernyataan kondisional yang melakukan tindakan berdasarkan kondisi berbeda. Ia menggunakan perbandingan yang ketat (`===`) untuk mencocokkan kondisi dan mengeksekusi blok kode dari kondisi yang cocok. Sintaks ekspresi `switch` ditunjukkan di bawah ini.

```
switch (expression) {
  case x:
    // blok kode
    break;
  case y:
    // blok kode
    break;
  default:
  // blok kode
}
```

Ekspresi dievaluasi sekali dan dibandingkan dengan setiap kasus. Jika ada kecocokan, maka blok kode yang terkait akan dieksekusi; jika tidak, blok kode `default` akan dieksekusi. Kata kunci `break` digunakan untuk menghentikan eksekusi dan dapat ditempatkan di mana saja. Jika tanpa `break` , kondisi berikutnya akan dievaluasi bahkan jika kondisinya tidak cocok.

Berikut adalah contoh mendapatkan nama hari berdasarkan kondisi `switch` .

```
switch (new Date().getDay()) {
  case 0:
    day = "Minggu";
    break;
  case 1:
    day = "Senin";
    break;
  case 2:
    day = "Selasa";
    break;
  case 3:
    day = "Rabu";
    break;
  case 4:
    day = "Kamis";
    break;
  case 5:
    day = "Jumat";
    break;
  case 6:
    day = "Sabtu";
}
```

Dalam beberapa kasus pencocokan, nilai pencocokan **first** yang dipilih, jika bukan nilai default yang dipilih. Jika tidak ada default dan tidak ada kasus yang cocok, program melanjutkan ke pernyataan berikutnya setelah kondisi peralihan.

Exercise

Dari nilai `dayOfWeek` berikut, tulis pernyataan `switch` yang memeriksa nilai dari `dayOfWeek`. Jika `dayOfWeek` adalah "Senin", "Selasa", "Rabu", "Kamis", atau "Jumat", berikan nilai "Ini hari kerja" ke variabel `result`. Jika `dayOfWeek` adalah "Sabtu" atau "Minggu", berikan nilai "Ini akhir pekan" ke variabel `result`.

```
let dayOfWeek = "Senin";
let result;
// Periksa apakah ini hari kerja atau akhir pekan
switch(expression) {
case x:
// Blok kode
break;
case y:
// Blok kode
break;
default:
// Blok kode
}
```

Comparators

Mari fokus pada bagian kondisional sekarang:

```
if (country === "France") {
    ...
}
```

Bagian kondisional ini adalah variabel country diikuti oleh tiga tanda sama dengan (===). Tiga tanda sama dengan menguji apakah variabel country memiliki nilai yang benar (France) dan juga tipe yang benar (String). Anda juga dapat menguji kondisi dengan dua tanda sama dengan, tetapi dalam kondisi seperti if (x == 5), itu akan mengembalikan nilai true baik untuk var x = 5; maupun var x = "5";. Tergantung pada apa yang dilakukan program Anda, ini bisa membuat perbedaan yang cukup besar. Sangat disarankan sebagai praktik terbaik bahwa Anda selalu membandingkan kesetaraan dengan tiga tanda sama dengan (=== dan !==) daripada dua (== dan !=).

Penguji kondisi lainnya:

- `x > a` : apakah x lebih besar dari a?
- `x < a` : apakah x lebih kecil dari a?
- `x <= a` : apakah x lebih kecil atau sama dengan a?
- `x >=a` : apakah x lebih besar atau sama dengan a?
- `x != a` : apakah x bukan a?
- `x` : apakah x ada?

Logical Comparison

Untuk menghindari kerumitan if-else, perbandingan logika sederhana dapat digunakan.

```
let topper = marks > 85 ? "YES" : "NO";
```

Pada contoh di atas, `?` adalah operator logis. Kode tersebut menyatakan bahwa jika nilai marks lebih dari 85, yaitu `marks > 85` , maka `topper = YES` ; jika tidak, maka `topper = NO` . Pada dasarnya, jika kondisi perbandingan terbukti benar, argumen pertama diakses, dan jika kondisi perbandingan salah, argumen kedua diakses.

Concatenate

Selain itu, Anda dapat menggabungkan kondisi yang berbeda dengan pernyataan "atau" atau "dan" untuk menguji apakah salah satu pernyataan benar, atau keduanya benar, secara berturut-turut.

Dalam JavaScript, "atau" ditulis sebagai `||` dan "dan" ditulis sebagai `&&`.

Misalkan Anda ingin menguji apakah nilai x berada di antara 10 dan 20. Anda dapat melakukannya dengan pernyataan berikut:

```
if (x > 10 && x < 20) {
    ...
}
```

Jika Anda ingin memastikan bahwa nilai `country` adalah "England" atau "Germany," Anda dapat menggunakan:

```
if (country === "England" || country === "Germany") {
    ...
}
```

> **Catatan**: Seperti operasi pada angka, kondisi dapat dikelompokkan dengan menggunakan tanda kurung, misalnya: `if ((name === "John" || name === "Jennifer") && country === "France")`.

Bab 6

Arrays

Array adalah bagian fundamental dalam pemrograman. Sebuah array adalah daftar data. Kita dapat menyimpan banyak data dalam satu variabel, yang membuat kode kita lebih mudah dibaca dan dipahami. Ini juga memudahkan kita dalam melakukan operasi pada data terkait.

Data dalam array disebut **elemen**.

Berikut adalah contoh sederhana dari sebuah array:

```
// 1, 1, 2, 3, 5, and 8 are the elements in this array
let numbers = [1, 1, 2, 3, 5, 8];
```

Array dapat dengan mudah dibuat menggunakan literal array atau dengan menggunakan kata kunci `new`.

```
const cars = ["Saab", "Volvo", "BMW"]; // using array literals
const cars = new Array("Saab", "Volvo", "BMW"); // using the new keyword
```

Nomor indeks digunakan untuk mengakses nilai dalam sebuah array. Indeks elemen pertama dalam array selalu `0` karena indeks array dimulai dari `0`. Nomor indeks juga dapat digunakan untuk mengubah elemen-elemen dalam array.

```
const cars = ["Saab", "Volvo", "BMW"];
console.log(cars[0]);
// Result: Saab

cars[0] = "Opel"; // changing the first element of an array
console.log(cars);
// Result: ['Opel', 'Volvo', 'BMW']
```

> Array adalah tipe objek khusus. Anda dapat memiliki objek dalam sebuah array.

Properti `length` dari sebuah array mengembalikan jumlah elemen. Metode yang didukung oleh Array ditunjukkan di bawah ini:

Nama	Deskripsi
`concat()`	Mengembalikan penggabungan dua atau lebih array
`join()`	Menggabungkan semua elemen dalam array menjadi sebuah string
`push()`	Menambahkan satu atau lebih elemen ke ujung array dan mengembalikan panjang array
`pop()`	Menghapus elemen terakhir dari array dan mengembalikan elemen tersebut
`shift()`	Menghapus elemen pertama dari array dan mengembalikan elemen tersebut
`unshift()`	Menambahkan satu atau lebih elemen di depan array dan mengembalikan panjang array
`slice()`	Mengambil sebagian dari array dan mengembalikan array baru
`at()`	Mengembalikan elemen pada indeks yang ditentukan atau `undefined`
`splice()`	Menghapus elemen-elemen dari array dan (opsional) menggantikannya, dan mengembalikan array
`reverse()`	Membalik urutan elemen dalam array dan mengembalikan referensi ke array
`flat()`	Mengembalikan array baru dengan semua elemen sub-array yang digabungkan menjadi satu secara rekursif hingga kedalaman yang ditentukan
`sort()`	Mengurutkan elemen-elemen dalam array di tempatnya, dan mengembalikan referensi ke array
`indexOf()`	Mengembalikan indeks pertama dari elemen pencarian
`lastIndexOf()`	Mengembalikan indeks terakhir dari elemen pencarian
`forEach()`	Mengeksekusi sebuah callback pada setiap elemen dalam array dan mengembalikan `undefined`
`map()`	Mengembalikan array baru dengan nilai pengembalian dari menjalankan `callback` pada setiap item array
`flatMap()`	Menjalankan `map()` diikuti oleh `flat()` dengan kedalaman 1
`filter()`	Mengembalikan array baru yang berisi elemen-elemen yang `callback` mengembalikan `true`
`find()`	Mengembalikan elemen pertama untuk mana `callback` mengembalikan `true`
`findLast()`	Mengembalikan elemen terakhir untuk mana `callback` mengembalikan `true`
`findIndex()`	Mengembalikan indeks pertama untuk mana `callback` mengembalikan `true`
`findLastIndex()`	Mengembalikan indeks terakhir untuk mana `callback` mengembalikan `true`
`every()`	Mengembalikan `true` jika `callback` mengembalikan `true` untuk setiap item dalam array
`some()`	Mengembalikan `true` jika `callback` mengembalikan `true` untuk setidaknya satu item dalam array
`reduce()`	Menggunakan `callback(accumulator, currentValue, currentIndex, array)` untuk tujuan pengurangan dan mengembalikan nilai akhir yang dihasilkan oleh fungsi `callback`
`reduceRight()`	Bekerja mirip dengan `reduce()` , tetapi dimulai dari elemen terakhir

Unshift

Metode `unshift` menambahkan elemen-elemen baru secara berurutan di bagian depan array. Ini mengubah array asli dan mengembalikan panjang array yang baru. Sebagai contoh:

```
let array = [0, 5, 10];
array.unshift(-5); // 4

// HASIL: array = [-5 , 0, 5, 10];
```

> Metode `unshift()` menimpa array asli.

Metode `unshift` menerima satu atau lebih argumen, yang mewakili elemen-elemen yang akan ditambahkan di awal array. Itu menambahkan elemen-elemen sesuai urutan yang diberikan, sehingga elemen pertama akan menjadi elemen pertama dalam array.

Berikut contoh penggunaan `unshift` untuk menambahkan beberapa elemen ke dalam array:

```
const numbers = [1, 2, 3];
const newLength = numbers.unshift(-1, 0);
console.log(numbers); // [-1, 0, 1, 2, 3]
console.log(newLength); // 5
```

Map

Metode `Array.prototype.map()` mengulangi sebuah array dan memodifikasi elemennya menggunakan sebuah fungsi callback. Fungsi callback ini kemudian diterapkan pada setiap elemen dari array.

Berikut adalah sintaks penggunaan `map` .

```
let newArray = oldArray.map(function (element, index, array) {
  // element: elemen saat ini yang sedang diproses dalam array
  // index: indeks dari elemen saat ini yang sedang diproses dalam array
  // array: array yang menggunakan metode map
  // Kembalikan elemen yang akan ditambahkan ke newArray
});
```

Sebagai contoh, katakanlah Anda memiliki sebuah array angka dan ingin membuat array baru yang menggandakan nilai-nilai angka dalam array asli. Anda dapat melakukannya menggunakan `map` seperti ini.

```
const numbers = [2, 4, 6, 8];

const doubledNumbers = numbers.map((number) => number * 2);

console.log(doubledNumbers);

// Result: [4, 8, 12, 16]
```

Anda juga dapat menggunakan sintaks fungsi panah (arrow function) untuk mendefinisikan fungsi yang dilewatkan ke `map` .

```
let doubledNumbers = numbers.map((number) => {
  return number * 2;
});
```

atau

```
let doubledNumbers = numbers.map((number) => number * 2);
```

Metode `map()` tidak menjalankan fungsi untuk elemen yang kosong dan tidak mengubah array asli.

Spread

Sebuah array atau objek dapat dengan cepat disalin ke array atau objek lain dengan menggunakan Operator Spread `(...)`. Ini memungkinkan iterable seperti array untuk diperluas di tempat-tempat di mana diharapkan nol atau lebih argumen (untuk pemanggilan fungsi) atau elemen (untuk literal array), atau ekspresi objek untuk diperluas di tempat-tempat di mana diharapkan nol atau lebih pasangan kunci-nilai (untuk literal objek).

Berikut beberapa contoh penggunaannya:

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

console.log(...arr);
// Hasil: 1 2 3 4 5

let arr1;
arr1 = [...arr]; // menyalin arr ke arr1

console.log(arr1); //Hasil: [1, 2, 3, 4, 5]

arr1 = [6, 7];
arr = [...arr, ...arr1];

console.log(arr); //Hasil: [1, 2, 3, 4, 5, 6, 7]
```

> Operator spread hanya berfungsi di peramban modern yang mendukung fitur tersebut. Jika Anda perlu mendukung peramban yang lebih tua, Anda perlu menggunakan transpiler seperti Babel untuk mengonversi sintaks operator spread menjadi kode ES5 yang setara.

Shift

Metode `shift` menghapus indeks pertama dari array tersebut dan menggeser semua indeks ke kiri. Ini mengubah array asli. Berikut adalah sintaks penggunaan `shift` :

```
array.shift();
```

For example:

```
let array = [1, 2, 3];
array.shift();

// Result: array = [2,3]
```

Anda juga dapat menggunakan metode `shift` bersama dengan sebuah perulangan untuk menghapus semua elemen dari sebuah array. Berikut contoh cara melakukannya:

```
while (array.length > 0) {
  array.shift();
}

console.log(array); // Result: []
```

> Metode `shift` hanya berfungsi pada array, dan tidak pada objek lain yang mirip dengan array seperti objek arguments atau objek NodeList. Jika Anda perlu melakukan shift elemen dari salah satu jenis objek ini, Anda perlu mengonversinya menjadi array terlebih dahulu menggunakan metode `Array.prototype.slice()` .

Pop

Metode `pop` menghapus elemen terakhir dari sebuah array dan mengembalikan elemen tersebut. Metode ini mengubah panjang array.

Berikut adalah sintaks penggunaan `pop` :

```
array.pop();
```

Sebagai contoh:

```
let arr = ["one", "two", "three", "four", "five"];
arr.pop();

console.log(arr);

// Result: ['one', 'two', 'three', 'four']
```

Anda juga dapat menggunakan metode `pop` bersama dengan sebuah perulangan untuk menghapus semua elemen dari sebuah array. Berikut adalah contoh bagaimana Anda dapat melakukannya:

```
while (array.length > 0) {
  array.pop();
}

console.log(array); // Result: []
```

> Metode `pop` hanya berfungsi pada array, dan tidak pada objek lain yang mirip dengan array, seperti objek arguments atau objek NodeList. Jika Anda perlu menghapus elemen dari salah satu jenis objek ini, Anda perlu mengonversinya menjadi array terlebih dahulu menggunakan metode `Array.prototype.slice()` .

Join

Metode `join` mengubah array menjadi sebuah string dan menggabungkannya. Ia tidak mengubah array asli. Berikut adalah sintaks untuk menggunakan `join`:

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

Argumen `separator` adalah opsional dan menentukan karakter yang akan digunakan untuk memisahkan elemen-elemen dalam string hasilnya. Jika diabaikan, elemen-elemen array dipisahkan dengan koma (`,`).

Sebagai contoh:

```
let array = ["one", "two", "three", "four"];

console.log(array.join(" "));

// Hasil: satu dua tiga empat
```

> Anda dapat menentukan pemisah apa pun, tetapi yang default adalah koma (`,`).

Pada contoh di atas, spasi digunakan sebagai pemisah. Anda juga dapat menggunakan `join` untuk mengkonversi objek mirip array (seperti objek arguments atau objek NodeList) menjadi sebuah string dengan pertama-tama mengonversinya menjadi array menggunakan metode `Array.prototype.slice()`:

```
function printArguments() {
  console.log(Array.prototype.slice.call(arguments).join(", "));
}

printArguments("a", "b", "c"); // Hasil: "a, b, c"
```

Length

Array memiliki sebuah properti yang disebut `length`, dan sebenarnya itu sesuai dengan namanya, yaitu panjang dari array tersebut.

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

let l = array.length;

// Hasil: l = 3
```

Properti panjang (length) juga menentukan jumlah elemen dalam sebuah array. Sebagai contoh:

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length = 2;

console.log(fruits);
// Hasil: ['Banana', 'Orange']
```

Anda juga dapat menggunakan properti `length` untuk mendapatkan elemen terakhir dari sebuah array dengan menggunakannya sebagai indeks. Sebagai contoh:

```
console.log(fruits[fruits.length - 1]); // Hasil: Orange
```

Anda juga dapat menggunakan properti `length` untuk menambahkan elemen ke ujung array. Sebagai contoh:

```
fruits[fruits.length] = "Pineapple";
console.log(fruits); // Hasil: ['Banana', 'Orange', 'Pineapple']
```

> Properti `length` secara otomatis diperbarui ketika elemen-elemen ditambahkan atau dihapus dari array.

Juga perlu diingat bahwa properti `length` bukanlah sebuah metode, sehingga Anda tidak perlu menggunakan tanda kurung saat mengaksesnya. Ini hanyalah sebuah properti dari objek array yang dapat Anda akses seperti properti objek lainnya.

Push

Seseorang dapat menambahkan elemen tertentu ke dalam sebuah array, membuat indeks terakhir menjadi item yang baru ditambahkan. Ini mengubah panjang array dan mengembalikan panjang array yang baru.

Berikut adalah sintaks penggunaan `push`:

```
array.push(element1[, ...[, elementN]]);
```

Argumen `element1, ..., elementN` mewakili elemen-elemen yang akan ditambahkan ke ujung array.

Sebagai contoh:

```
let array = [1, 2, 3];
array.push(4);

console.log(array);

// Result: array = [1, 2, 3, 4]
```

Anda juga dapat menggunakan `push` untuk menambahkan elemen ke ujung objek mirip array (seperti objek arguments atau objek NodeList) dengan pertama-tama mengonversinya menjadi array menggunakan metode `Array.prototype.slice()`.

```
function printArguments() {
  let args = Array.prototype.slice.call(arguments);
  args.push("d", "e", "f");
  console.log(args);
}

printArguments("a", "b", "c"); // Result: ["a", "b", "c", "d", "e", "f"]
```

> **Catatan**: Metode `push` mengubah array asli. Ia tidak membuat array baru.

For Each

Metode `forEach` menjalankan sebuah fungsi yang disediakan sekali untuk setiap elemen dalam array. Berikut adalah sintaks untuk menggunakan `forEach` :

```
array.forEach(function (element, index, array) {
  // element: elemen saat ini yang sedang diproses dalam array
  // index: indeks elemen saat ini yang sedang diproses dalam array
  // array: array yang digunakan sebagai panggilan forEach
});
```

Sebagai contoh, katakanlah Anda memiliki sebuah array angka dan ingin mencetak dua kali lipat dari setiap angka ke konsol. Anda dapat melakukannya menggunakan `forEach` seperti ini:

```
let numbers = [1, 2, 3, 4, 5];
numbers.forEach(function (number) {
  console.log(number * 2);
});
```

Anda juga dapat menggunakan sintaks fungsi panah (arrow function) untuk mendefinisikan fungsi yang dilewatkan ke `forEach` :

```
numbers.forEach((number) => {
  console.log(number * 2);
});
```

or

```
numbers.forEach((number) => console.log(number * 2));
```

Metode `forEach` tidak memodifikasi array asli. Ia hanya mengulanginya melalui elemen-elemen dalam array dan menjalankan fungsi yang diberikan untuk setiap elemen.

> Metode `forEach()` tidak akan dieksekusi jika tidak ada pernyataan (statement) dalamnya.

Sort

Metode `sort` mengurutkan elemen-elemen dalam array dalam urutan tertentu (menaik atau menurun). Secara default, metode `sort` mengurutkan elemen-elemen sebagai string dan menyusunnya dalam urutan naik berdasarkan nilai unit kode UTF-16 mereka. Berikut adalah sintaks penggunaan `sort` :

```
array.sort([compareFunction]);
```

Argumen `compareFunction` bersifat opsional dan menentukan sebuah fungsi yang mendefinisikan urutan pengurutan. Jika diabaikan, elemen-elemen diurutkan dalam urutan naik berdasarkan representasi string mereka.

Sebagai contoh:

```
let city = ["California", "Barcelona", "Paris", "Kathmandu"];
let sortedCity = city.sort();

console.log(sortedCity);

// Hasil: ['Barcelona', 'California', 'Kathmandu', 'Paris']
```

> Angka dapat diurutkan dengan tidak benar ketika diurutkan sebagai string. Sebagai contoh, "35" lebih besar dari "100" karena "3" lebih besar dari "1".

Untuk mengatasi masalah pengurutan dalam angka, fungsi perbandingan digunakan. Fungsi perbandingan mendefinisikan urutan pengurutan dan mengembalikan nilai **negatif**, **nol**, atau **positif** berdasarkan argumen, seperti ini:

- Nilai negatif jika `a` harus diurutkan sebelum `b` .
- Nilai positif jika `a` harus diurutkan setelah `b` .
- `0` jika `a` dan `b` sama dan urutan mereka tidak masalah.

```
const points = [40, 100, 1, 5, 25, 10];
points.sort((a, b) => {
  return a - b;
});

// Hasil: [1, 5, 10, 25, 40, 100]
```

> Metode `sort()` mengganti array asli.

Indices

Indeks Jadi, Anda memiliki array elemen data, tetapi bagaimana jika Anda ingin mengakses elemen tertentu? Di situlah peran indeks masuk. Indeks merujuk pada posisi dalam array. Secara logis, indeks berkembang satu per satu, tetapi perlu diingat bahwa indeks pertama dalam sebuah array adalah 0, seperti halnya dalam kebanyakan bahasa pemrograman. Tanda kurung siku [] digunakan untuk menunjukkan bahwa Anda merujuk pada indeks dalam sebuah array.

```
// Ini adalah array string
let fruits = ["apple", "banana", "pineapple", "strawberry"];

// Kami mengatur variabel banana dengan nilai dari elemen kedua dari
// array buah. Ingatlah bahwa indeks dimulai dari 0, jadi 1 adalah
// elemen kedua. Hasil: banana = "pisang"

let banana = fruits[1];
```

Anda juga dapat menggunakan indeks array untuk mengatur nilai dari elemen dalam array:

```
let array = ["a", "b", "c", "d", "e"];
//  indices:  0    1    2    3    4
array[4] = "f";
console.log(array); // Hasil: ['a', 'b', 'c', 'd', 'f']
```

Perhatikan bahwa jika Anda mencoba mengakses atau mengatur elemen menggunakan indeks di luar batas array (misalnya, indeks yang kurang dari 0 atau lebih besar atau sama dengan panjang array), Anda akan mendapatkan nilai undefined.

```
console.log(array[5]); // Output: undefined
array[5] = "g";
console.log(array); // Hasil: ['a', 'b', 'c', 'd', 'f', undefined, 'g']
```

Reverse

Metode `reverse` dapat digunakan pada jenis array apa pun, termasuk array string, array angka, dan array objek. Sebagai contoh.

```
let users = [
  {
    name: "John Smith",
    age: 30,
  },
  {
    name: "Jane Doe",
    age: 25,
  },
];

users.reverse();

console.log(users);

// RESULT:
[
  {
    name: "Jane Doe",
    age: 25,
  },
  {
    name: "John Smith",
    age: 30,
  },
];
```

Metode `reverse` mengubah urutan elemen dari objek array yang memanggilnya, memodifikasi array tersebut, dan mengembalikan referensi ke array tersebut. Berikut contoh penggunaan `reverse` pada sebuah array:

```
const numbers = [1, 2, 3];
const newLength = numbers.reverse();
console.log(numbers); // [3, 2, 1]
```

Slice

Metode `slice` menerima dua parameter, yaitu `begin` dan `end`.

- `begin` : Parameter ini menentukan indeks awal dari mana bagian akan diekstraksi. Jika argumen ini tidak ada, metode akan menggunakan 0 sebagai nilai awal default.
- `end` : Parameter ini adalah indeks hingga mana bagian akan diekstraksi (tidak termasuk indeks akhir). Jika argumen ini tidak didefinisikan, maka array hingga akhir akan diekstraksi karena itu adalah nilai akhir default. Jika nilai akhir lebih besar dari panjang array, maka nilai akhir akan berubah menjadi panjang array.

```
function func() {
  // Array Asli
  let arr = [23, 56, 87, 32, 75, 13];
  // Array yang Diekstrak
  let new_arr = arr.slice();
  console.log(arr);
  console.log(new_arr);
}
func();

// HASIL:
[23, 56, 87, 32, 75, 13][(23, 56, 87, 32, 75, 13)];
```

Metode `slice()` menyalin referensi objek ke array baru. (Misalnya, array bersarang) Jadi jika objek yang dirujuk dimodifikasi, perubahan tersebut terlihat dalam array baru yang dikembalikan.

```
let human = {
  name: "David",
  age: 23,
};

let arr = [human, "Nepal", "Manager"];
let new_arr = arr.slice();

// objek asli
console.log(arr[0]); // { name: 'David', age: 23 }

// membuat perubahan pada objek dalam array baru
new_arr[0].name = "Levy";

// perubahan terlihat
console.log(arr[0]); // { name: 'Levy', age: 23 }
```

Bab 7

Perulangan

Perulangan adalah instruksi berulang di mana satu variabel dalam perulangan berubah. Perulangan sangat berguna jika Anda ingin menjalankan kode yang sama berulang kali, setiap kali dengan nilai yang berbeda.

Daripada menulis:

```
lakukanHal(cars[0]);
lakukanHal(cars[1]);
lakukanHal(cars[2]);
lakukanHal(cars[3]);
lakukanHal(cars[4]);
```

Anda dapat menulis:

```
for (var i = 0; i < cars.length; i++) {
  lakukanHal(cars[i]);
}
```

For

Bentuk teringan dari perulangan adalah pernyataan for. Ini memiliki sintaksis yang mirip dengan pernyataan if, tetapi dengan lebih banyak pilihan:

Sintaksis

Sintaksis perulangan `for` dalam javascript diberikan di bawah ini

```
for (inisialisasi; kondisi akhir; perubahan) {
    // lakukan itu, lakukan sekarang
}
```

Penjelasan:

- Pada bagian `inisialisasi` , yang dieksekusi sebelum iterasi pertama, menginisialisasi variabel perulangan Anda.
- Pada bagian `kondisi akhir` , letakkan kondisi yang mungkin diperiksa sebelum setiap iterasi. Saat kondisi menjadi `false` , perulangan berakhir.
- Pada bagian `perubahan` , beri tahu program cara memperbarui variabel perulangan. Mari kita lihat bagaimana mengeksekusi kode yang sama sepuluh kali menggunakan perulangan `for` :

```
for (let i = 0; i < 10; i = i + 1) {
  // lakukan kode ini sepuluh kali
}
```

> ***Catatan***: `i = i + 1` dapat ditulis sebagai `i++` .

Perulangan `for...in`

Untuk mengulangi properti yang dapat dihitung dari objek, Anda dapat menggunakan perulangan `for in` . Untuk setiap properti yang berbeda, JavaScript mengeksekusi pernyataan yang ditentukan.

Sintaksis

```
for (variabel in objek) {
  // iterasi setiap properti dalam objek
}
```

Contoh

Mari kita anggap kita memiliki objek berikut:

```
const person = {
  fname: "John",
  lname: "Doe",
  age: 25,
};
```

Maka, dengan bantuan perulangan `for in`, kita dapat mengulangi objek `person` untuk mengakses propertinya seperti `fname`, `lname`, dan `age` seperti yang ditunjukkan di bawah ini.

```
let info = "";
for (let x in person) {
  console.log(person[x]);
}
```

Hasil dari potongan kode di atas akan:

```
John
Doe
25
```

> **Catatan: Objek yang dapat diulang seperti `Array`, `String`, `Map`, `NodeList` dapat diulang menggunakan pernyataan `for in`.**

```
// Contoh dengan Array
const myArray = [1, 2, 3, 4, 5];
for (const item of myArray) {
  console.log(item);
}

// Contoh dengan String
const myString = "Halo, Dunia!";
for (const char of myString) {
  consolesole.log(char);
}

// Contoh dengan Map
const myMap = new Map();
myMap.set("nama", "John");
myMap.set("usia", 30);

for (const [kunci, nilai] of myMap) {
  console.log(kunci, nilai);
}

// Contoh dengan NodeList (elemen HTML)
const paragraf = document.querySelectorAll("p");
for (const paragraph of paragraphs) {
  console.log(paragraph.textContent);
}
```

Perulangan `for...of`

Perulangan `for...of` diperkenalkan dalam versi JavaScript ES6 yang lebih baru. Pernyataan `for...of` menjalankan perulangan yang beroperasi pada urutan nilai yang bersumber dari objek yang dapat diulang seperti Array, String, TypedArray, Map, Set, NodeList (dan koleksi DOM lainnya).

Sintaksis

Sintaksis perulangan `for...of` adalah sebagai berikut:

```
for (elemen dari iterable) {
  // tubuh dari for...of
}
```

Di sini,

- **iterable** - objek yang dapat diulang
- **elemen** - item dalam iterable

Dalam bahasa Inggris sederhana, Anda dapat membaca kode di atas sebagai: *"untuk setiap elemen dalam iterable, jalankan tubuh perulangan."*

Contoh

Mari kita anggap kita memiliki objek berikut:

```
const person = ["John Doe", "Albert", "Neo"];
```

Maka, dengan bantuan perulangan `for of` , kita dapat mengulangi objek `person` untuk mengakses elemennya seperti yang ditunjukkan di bawah ini.

```
let info = "";
for (let x of person) {
  console.log(x);
}
```

Hasil dari potongan kode di atas akan:

```
John Doe
Albert
Neo
```

Penggunaan perulangan `for...of` dengan string, map, dan nodelist diberikan di bawah ini:

```
// Contoh dengan String
const text = "Halo, Dunia!";
for (const char of text) {
  console.log(char);
}

// Contoh dengan Map
const person = new Map();
person.set("nama", "John");
person.set("usia", 30);
for (const [kunci, nilai] of person) {
  console.log(kunci, nilai);
}

// Contoh dengan NodeList (elemen HTML)
const paragraf = document.querySelectorAll("p");
for (const paragraph of paragraphs) {
  console.log(paragraph.textContent);
}
```

While

Perulangan while menjalankan berulang kali blok kode selama kondisi yang ditentukan benar. Ini menyediakan cara untuk mengotomatisasi tugas-tugas berulang dan melakukan iterasi berdasarkan evaluasi kondisi.

```
while (kondisi) {
  // lakukan selama kondisi benar
}
```

Sebagai contoh, perulangan dalam contoh ini akan menjalankan berulang kali blok kode selama variabel i kurang dari 5:

```
var i = 0,
  x = "";
while (i < 5) {
  x = x + "Nomornya adalah " + i;
  i++;
}
```

Hati-hati untuk menghindari perulangan tak terbatas jika kondisinya selalu benar!

Do...While

Pernyataan do...while membuat perulangan yang mengeksekusi pernyataan blok yang ditentukan sampai kondisi uji dievaluasi sebagai salah. Kondisi dievaluasi setelah mengeksekusi blok. Sintaksis untuk do...while adalah

```
do {
  // pernyataan
} while (ekspresi);
```

Mari sebagai contoh lihat bagaimana mencetak angka kurang dari 10 menggunakan perulangan `do...while` :

```
var i = 0;
do {
  document.write(i + " ");
  i++; // menambahkan i sebanyak 1
} while (i < 10);
```

> ***Catatan***: `i = i + 1` dapat ditulis sebagai `i++` .

Bab 8

Fungsi

Fungsi adalah salah satu konsep paling kuat dan esensial dalam pemrograman. Fungsi, seperti fungsi matematika, melakukan transformasi, mereka mengambil nilai masukan yang disebut **argumen** dan **mengembalikan** nilai keluaran.

Fungsi dapat dibuat dengan dua cara: menggunakan `deklarasi fungsi` atau `ekspresi fungsi`. *Nama fungsi* dapat dihilangkan dalam `ekspresi fungsi`, menjadikannya sebagai `fungsi anonim`. Fungsi, seperti variabel, harus dideklarasikan. Mari deklarasikan fungsi `double` yang menerima *argumen* bernama `x` dan **mengembalikan** nilai ganda dari x:

```
// contoh deklarasi fungsi
function double(x) {
  return 2 * x;
}
```

> *Catatan:* fungsi di atas **mungkin** dirujuk sebelum didefinisikan.

Fungsi juga merupakan nilai dalam JavaScript; mereka dapat disimpan dalam variabel (seperti angka, string, dll...) dan diberikan kepada fungsi lain sebagai argumen:

```
// contoh ekspresi fungsi
let double = function (x) {
  return 2 * x;
};
```

> *Catatan:* fungsi di atas **tidak** boleh dirujuk sebelum didefinisikan, sama seperti variabel lainnya.

Sebuah callback adalah fungsi yang dilewatkan sebagai argumen ke fungsi lain.

Fungsi panah adalah alternatif ringkas untuk fungsi tradisional yang memiliki beberapa perbedaan semantik dengan beberapa batasan. Fungsi ini tidak memiliki ikatan mereka sendiri dengan `this`, `arguments`, dan `super`, dan tidak dapat digunakan sebagai konstruktor. Contoh dari fungsi panah.

```
const double = (x) => 2 * x;
```

Kata kunci `this` dalam fungsi panah mewakili objek yang mendefinisikan fungsi panah tersebut.

Fungsi Orde Tinggi

Fungsi orde tinggi adalah fungsi yang memanipulasi fungsi lain. Misalnya, sebuah fungsi dapat menerima fungsi-fungsi lain sebagai argumen dan/atau menghasilkan sebuah fungsi sebagai nilai kembali. Teknik-teknik fungsional yang *kreatif* seperti ini adalah konstruk kuat yang tersedia bagi Anda dalam JavaScript dan bahasa tingkat tinggi lainnya seperti python, lisp, dll.

Sekarang kita akan membuat dua fungsi sederhana, `add_2` dan `double`, serta sebuah fungsi orde tinggi yang disebut `map`. `map` akan menerima dua argumen, `func` dan `list` (deklarasinya akan dimulai dengan `map(func, list)`), dan mengembalikan sebuah array. `func` (argumen pertama) akan menjadi sebuah fungsi yang akan diterapkan pada setiap elemen dalam array `list` (argumen kedua).

```
// Mendefinisikan dua fungsi sederhana
let add_2 = function (x) {
  return x + 2;
};
let double = function (x) {
  return 2 * x;
};

// map adalah fungsi keren yang menerima 2 argumen:
//  func    fungsi yang akan dipanggil
//  list    array nilai yang akan dipanggil oleh func
let map = function (func, list) {
  let output = []; // daftar keluaran
  for (idx in list) {
    output.push(func(list[idx]));
  }
  return output;
};

// Kami menggunakan map untuk menerapkan sebuah fungsi pada seluruh daftar masukan
// untuk "memetakkannya" menjadi daftar keluaran yang sesuai
map(add_2, [5, 6, 7]); // => [7, 8, 9]
map(double, [5, 6, 7]); // => [10, 12, 14]
```

Fungsi dalam contoh di atas sederhana. Namun, ketika digunakan sebagai argumen untuk fungsi lain, mereka dapat digabungkan dengan cara yang tidak terduga untuk membangun fungsi yang lebih kompleks.

Sebagai contoh, jika kita memperhatikan bahwa kita menggunakan pemanggilan `map(add_2, ...)` dan `map(double, ...)` sangat sering dalam kode kita, kita bisa memutuskan untuk membuat dua fungsi pemrosesan daftar yang bersifat khusus dan memiliki operasi yang diinginkan tertanam di dalamnya. Dengan menggunakan komposisi fungsi, kita bisa melakukannya sebagai berikut:

```
process_add_2 = function (list) {
  return map(add_2, list);
};
process_double = function (list) {
  return map(double, list);
};
process_add_2([5, 6, 7]); // => [7, 8, 9]
process_double([5, 6, 7]); // => [10, 12, 14]
```

Sekarang mari kita membuat fungsi yang disebut `buildProcessor` yang mengambil sebuah fungsi `func` sebagai input dan mengembalikan pemroses `func`, yaitu, sebuah fungsi yang menerapkan `func` pada setiap masukan dalam daftar.

```
// sebuah fungsi yang menghasilkan pemroses daftar yang melakukan
let buildProcessor = function (func) {
  let process_func = function (list) {
    return map(func, list);
  };
  return process_func;
};
// memanggil buildProcessor mengembalikan sebuah fungsi yang dipanggil dengan masukan daftar

// dengan menggunakan buildProcessor, kita bisa menghasilkan pemroses daftar add_2 dan double seperti ini:
process_add_2 = buildProcessor(add_2);
process_double = buildProcessor(double);

process_add_2([5, 6, 7]); // => [7, 8, 9]
process_double([5, 6, 7]); // => [10, 12, 14]
```

Mari kita lihat contoh lain. Kita akan membuat sebuah fungsi yang disebut `buildMultiplier` yang mengambil sebuah angka `x` sebagai masukan dan mengembalikan sebuah fungsi yang mengalikan argumennya dengan `x`:

```
let buildMultiplier = function (x) {
  return function (y) {
    return x * y;
  };
};

let double = buildMultiplier(2);
let triple = buildMultiplier(3);

double(3); // => 6
triple(3); // => 9
```

Bab 9

Objek

Dalam JavaScript, objek bersifat **mutable** karena kita mengubah nilai yang ditunjuk oleh objek referensi, sedangkan ketika kita mengubah nilai primitif, kita mengubah referensi yang sekarang menunjuk ke nilai baru, sehingga primitif bersifat **immutable**. Tipe primitif dalam JavaScript adalah `true`, `false`, `number`, `string`, `null`, dan `undefined`. Nilai lainnya adalah `object`. Objek berisi pasangan `namaProperti`: `nilaiProperti`. Ada tiga cara untuk membuat objek dalam JavaScript:

1. Literal

```
let objek = {};
// Ya, hanya sepasang tanda kurung kurawal!
```

> ***Catatan:*** Ini adalah cara yang **direkomendasikan**.

2. Berorientasi objek

```
let objek = new Object();
```

> ***Catatan:*** Ini hampir seperti Java.

3. Dan menggunakan `Object.create`

```
let objek = Object.create(proto[, propertiesObject]);
```

> ***Catatan:*** Ini membuat objek baru dengan objek prototipe yang ditentukan dan properti.

Properti

Properti objek adalah pasangan `namaProperti : nilaiProperti`, di mana **nama properti hanya dapat berupa string**. Jika bukan string, itu akan diubah menjadi string. Anda dapat menentukan properti **saat membuat** objek **atau nanti**. Bisa ada nol atau lebih properti yang dipisahkan oleh koma.

```
let bahasa = {
  nama: "JavaScript",
  didukungOlehBrowser: true,
  diciptakanPadaTahun: 1995,
  penulis: {
    namaDepan: "Brendan",
    namaBelakang: "Eich",
  },
  // Ya, objek bisa bersarang!
  dapatkanNamaPenulis: function () {
    return this.penulis.namaDepan + " " + this.penulis.namaBelakang;
  },
  // Ya, fungsi juga bisa menjadi nilai!
};
```

Kode berikut mendemonstrasikan cara **mendapatkan** nilai properti.

```
let variabel = bahasa.nama;
// variabel sekarang berisi string "JavaScript".
variabel = bahasa["nama"];
// Baris di atas melakukan hal yang sama. Perbedaannya adalah yang kedua memungkinkan Anda menggunakan string a
variabel = bahasa.propertiBaru;
// variabel sekarang tidak terdefinisi, karena kami belum menetapkan properti ini.
```

Contoh berikut menunjukkan cara **menambahkan** properti **atau mengubah** yang sudah ada.

```
bahasa.propertiBaru = "nilai baru";
// Sekarang objek memiliki properti baru. Jika properti sudah ada, nilainya akan diganti.
bahasa["propertiBaru"] = "nilai diubah";
// Sekali lagi, Anda bisa mengakses properti kedua cara. Yang pertama (notasi titik) lebih disarankan.
```

Mutable

Perbedaan antara objek dan nilai primitif adalah bahwa kita dapat **mengubah objek**, sementara nilai primitif adalah **immutable**.

Sebagai contoh:

```
let nilaiPrimitifSaya = "nilai pertama";
nilaiPrimitifSaya = "nilai lain";
// nilaiPrimitifSaya sekarang menunjuk ke string lain.
let objekSaya = { kunci: "nilai pertama" };
objekSaya.kunci = "nilai lain";
// objekSaya menunjuk ke objek yang sama.
```

Anda dapat menambahkan, mengubah, atau menghapus properti dari objek menggunakan notasi titik atau notasi tanda kurung siku.

```
let objek = {};
objek.foo = "bar"; // Menambahkan properti 'foo'
objek["baz"] = "qux"; // Menambahkan properti 'baz'
objek.foo = "quux"; // Mengubah properti 'foo'
delete objek.baz; // Menghapus properti 'baz'
```

> Nilai primitif (seperti angka dan string) adalah immutable, sedangkan objek (seperti array dan objek) adalah mutable.

Referensi

Objek **tidak pernah disalin**. Mereka dilewatkan dengan referensi. Referensi objek adalah nilai yang merujuk pada objek. Ketika Anda membuat objek menggunakan operator `new` atau sintaksis objek literal, JavaScript membuat objek dan menetapkan referensi padanya.

Berikut contoh pembuatan objek menggunakan sintaksis objek literal:

```
var objek = {
  foo: "bar",
};
```

Berikut contoh pembuatan objek menggunakan operator `new`:

```
var objek = new Object();
objek.foo = "bar";
```

Ketika Anda menugaskan referensi objek ke sebuah variabel, variabel tersebut hanya menyimpan referensi ke objek, bukan objek itu sendiri. Ini berarti jika Anda menugaskan referensi objek ke variabel lain, kedua variabel akan menunjuk ke objek yang sama.

Contohnya:

```
var objek1 = {
  foo: "bar",
};

var objek2 = objek1;

console.log(objek1 === objek2); // Output: true
```

Pada contoh di atas, baik `objek1` maupun `objek2` adalah variabel yang menyimpan referensi ke objek yang sama. Operator `===` digunakan untuk membandingkan referensi, bukan objek itu sendiri, dan mengembalikan `true` karena kedua variabel menunjuk ke objek yang sama.

> Anda dapat menggunakan metode `Object.assign()` untuk membuat objek baru yang merupakan salinan objek yang ada.

Berikut adalah contoh pembuatan objek melalui referensi.

```
// Bayangkan saya punya sepotong pizza
let pizzaSaya = { potongan: 5 };
// Dan saya membaginya dengan Anda
let pizzaAnda = pizzaSaya;
// Saya makan satu potong lagi
pizzaSaya.potongan = pizzaSaya.potongan - 1;
let jumlahPotonganTersisa = pizzaAnda.potongan;
// Sekarang kita memiliki 4 potong karena pizzaSaya dan pizzaAnda
// merujuk ke objek pizza yang sama.
let a = {},
  b = {},
  c = {};
// a, b, dan c masing-masing merujuk ke objek kosong yang berbeda
a = b = c = {};
// a, b, dan c semuanya merujuk ke
// objek kosong yang sama
```

Prototipe

Setiap objek terhubung ke objek prototipe dari mana objek tersebut mewarisi properti. Objek yang dibuat dari literal objek (`{}`) secara otomatis terhubung ke `Object.prototype` , yang merupakan objek bawaan dalam JavaScript.

Ketika interpreter JavaScript (modul di browser Anda) mencoba untuk menemukan properti yang ingin Anda ambil, seperti dalam kode berikut:

```
let dewasa = { usia: 26 },
  propertiDitemukan = dewasa.usia;
// Baris di atas
```

Pertama, interpreter mencari melalui setiap properti yang dimiliki objek itu sendiri. Misalnya, `dewasa` hanya memiliki satu properti sendiri — `usia` . Tetapi selain itu, sebenarnya ada beberapa properti lain, yang diwarisi dari `Object.prototype.`

```
let representasiString = dewasa.toString();
// variabel tersebut memiliki nilai '[object Object]'
```

`toString` adalah properti Object.prototype, yang diwarisi. Ini memiliki nilai fungsi, yang mengembalikan representasi string dari objek. Jika Anda ingin agar itu mengembalikan representasi yang lebih bermakna, maka Anda dapat menggantinya. Cukup tambahkan properti baru ke objek dewasa.

```
dewasa.toString = function () {
  return "Saya berusia " + this.usia;
};
```

Jika Anda memanggil fungsi `toString` sekarang, interpreter akan menemukan properti baru dalam objek itu sendiri dan berhenti.

Dengan demikian, interpreter mengambil properti pertama yang akan ditemukan dalam perjalanan dari objek itu sendiri dan lebih lanjut melalui prototipnya.

Untuk menetapkan objek khusus Anda sebagai prototipe alih-alih Object.prototype default, Anda dapat memanggil `Object.create` seperti berikut:

```
let anak = Object.create(dewasa);
/* Cara membuat objek seperti ini memungkinkan kita dengan mudah mengganti Object.prototype default dengan yang
anak.usia = 8;
/* Sebelumnya, anak tidak memiliki properti usianya sendiri, dan interpreter harus melihat lebih lanjut ke prot
Sekarang, ketika kita menetapkan usia anak sendiri, interpreter tidak akan melanjutkan pencarian.
Catatan: usia dewasa masih 26. */
let representasiString = anak.toString();
// Nilainya adalah "Saya berusia 8".
/* Catatan: kita tidak telah mengganti properti toString anak, sehingga metode dewasa akan dipanggil. Jika dewa
```

Prototipe `anak` adalah `dewasa` , yang prototipenya adalah `Object.prototype` . Urutan prototipe ini disebut sebagai **rantai prototipe**.

Operator Delete

Operator `delete` dapat digunakan untuk **menghapus properti** dari objek. Ketika sebuah properti dihapus, itu dihilangkan dari objek dan tidak dapat diakses atau dienumerasi (misalnya, properti tersebut tidak muncul dalam loop for-in).

Berikut adalah sintaks untuk menggunakan `delete` :

```
delete objek.properti;
```

Sebagai contoh:

```
let dewasa = { usia: 26 },
  anak = Object.create(dewasa);

anak.usia = 8;

delete anak.usia;

/* Menghapus properti usia dari anak, sehingga mengungkapkan usia dari prototipenya, karena properti tersebut t

let usiaPrototipe = anak.usia;
// 26, karena anak tidak memiliki properti usia sendiri.
```

> Operator `delete` hanya berfungsi pada properti sendiri dari objek, dan tidak pada properti yang diwarisi. Ini juga tidak berfungsi pada properti yang memiliki atribut `configurable` yang diatur ke `false` .

Operator `delete` tidak memodifikasi rantai prototip objek. Ini hanya menghapus properti yang ditentukan dari objek dan juga tidak benar-benar menghancurkan objek atau propertinya. Ini hanya membuat properti tidak dapat diakses. Jika Anda perlu menghancurkan objek dan melepaskan memori, Anda dapat mengatur objek menjadi `null` atau menggunakan pengumpul sampah (garbage collector) untuk mengembalikan memori.

Enumeration

Enumeration merujuk pada proses mengulang properti dari sebuah objek dan melakukan tindakan tertentu untuk setiap properti. Ada beberapa cara untuk mengenumerasi properti objek dalam JavaScript.

Salah satu cara untuk mengenumerasi properti objek adalah dengan menggunakan perulangan `for-in`. Perulangan `for-in` akan mengulang properti yang dapat dienumerasi dari objek dalam urutan sembarangan, dan untuk setiap properti, ia akan menjalankan blok kode yang diberikan.

Pernyataan `for-in` dapat mengulang semua nama properti dalam objek. Enumeration akan mencakup fungsi dan properti prototipe.

```
let buah = {
    apel: 2,
    jeruk: 5,
    pir: 1,
  },
  kalimat = "Saya punya ",
  jumlah;
for (jenis in buah) {
  jumlah = buah[jenis];
  kalimat += jumlah + " " + jenis + (jumlah === 1 ? "" : "s") + ", ";
}
// Baris berikut menghapus koma di akhir.
kalimat = kalimat.substr(0, kalimat.length - 2) + ".";
// Saya punya 2 apel, 5 jeruk, 1 pir.
```

Cara lain untuk mengenumerasi properti objek adalah dengan menggunakan metode `Object.keys()`, yang mengembalikan sebuah array dari nama properti enumerable objek tersebut.

Sebagai contoh:

```
let objek = {
  foo: "bar",
  baz: "qux",
};

let properti = Object.keys(objek);
properti.forEach(function (prop) {
  console.log(prop + ": " + objek[prop]);
});

// foo: bar
// baz: qux
```

Bab 10

Tanggal dan Waktu

Objek `Date` menyimpan tanggal dan waktu serta menyediakan metode untuk mengelolanya. Objek tanggal bersifat statis dan menggunakan zona waktu default browser untuk menampilkan tanggal sebagai string teks penuh.

Untuk membuat objek `Date`, kita menggunakan konstruktor `new Date()` dan dapat dibuat dengan cara berikut.

```
new Date()
new Date(string tanggal)
new Date(tahun, bulan)
new Date(tahun, bulan, hari)
new Date(tahun, bulan, hari, jam)
new Date(tahun, bulan, hari, jam, menit)
new Date(tahun, bulan, hari, jam, menit, detik)
new Date(tahun, bulan, hari, jam, menit, detik, ms)
new Date(milidetik)
```

> Bulan dapat ditentukan dari `0` hingga `11`, lebih dari itu akan mengakibatkan overflow ke tahun berikutnya.

Metode dan properti yang didukung oleh objek tanggal dijelaskan di bawah ini:

Nama	Deskripsi
`constructor`	Mengembalikan fungsi yang membuat prototipe objek Tanggal
`getDate()`	Mengembalikan tanggal (1-31) dari bulan
`getDay()`	Mengembalikan hari (0-6) dari minggu
`getFullYear()`	Mengembalikan tahun (4 digit)
`getHours()`	Mengembalikan jam (0-23)
`getMilliseconds()`	Mengembalikan milidetik (0-999)
`getMinutes()`	Mengembalikan menit (0-59)
`getMonth()`	Mengembalikan bulan (0-11)
`getSeconds()`	Mengembalikan detik (0-59)
`getTime()`	Mengembalikan nilai numerik dari tanggal yang ditentukan dalam milidetik sejak tengah malam 1 Januari 1970
`getTimezoneOffset()`	Mengembalikan perbedaan zona waktu dalam menit
`getUTCDate()`	Mengembalikan tanggal (1-31) dari bulan menurut waktu universal
`getUTCDay()`	Mengembalikan hari (0-6) menurut waktu universal
`getUTCFullYear()`	Mengembalikan tahun (4 digit) menurut waktu universal
`getUTCHours()`	Mengembalikan jam (0-23) menurut waktu universal
`getUTCMilliseconds()`	Mengembalikan milidetik (0-999) menurut waktu universal
`getUTCMinutes()`	Mengembalikan menit (0-59) menurut waktu universal
`getUTCMonth()`	Mengembalikan bulan (0-11) menurut waktu universal
`getUTCSeconds()`	Mengembalikan detik (0-59) menurut waktu universal
`now()`	Mengembalikan nilai numerik dalam milidetik sejak tengah malam 1 Januari 1970
`parse()`	Mengurai string tanggal dan mengembalikan nilai numerik dalam milidetik sejak tengah malam 1 Januari 1970
`prototype`	Memungkinkan penambahan properti
`setDate()`	Mengatur tanggal bulan
`setFullYear()`	Mengatur tahun
`setHours()`	Mengatur jam
`setMilliseconds()`	Mengatur milidetik
`setMinutes()`	Mengatur menit
`setMonth()`	Mengatur bulan
`setSeconds()`	Mengatur detik
`setTime()`	Mengatur waktu
`setUTCDate()`	Mengatur tanggal bulan menurut waktu universal
`setUTCFullYear()`	Mengatur tahun menurut waktu universal
`setUTCHours()`	Mengatur jam menurut waktu universal

Nama	Deskripsi
`setUTCMilliseconds()`	Mengatur milidetik menurut waktu universal
`setUTCMinutes()`	Mengatur menit menurut waktu universal
`setUTCMonth()`	Mengatur bulan menurut waktu universal
`setUTCSeconds()`	Mengatur detik menurut waktu universal
`toDateString()`	Mengembalikan tanggal dalam format yang mudah dibaca
`toISOString()`	Mengembalikan tanggal sesuai dengan format

ISO | | `toJSON()` | Mengembalikan tanggal dalam bentuk string, diformat sebagai tanggal JSON | | `toLocaleDateString()` | Mengembalikan tanggal dalam bentuk string menggunakan konvensi lokal | | `toLocaleTimeString()` | Mengembalikan waktu dalam bentuk string menggunakan konvensi lokal | | `toLocaleString()` | Mengembalikan tanggal menggunakan konvensi lokal | | `toString()` | Mengembalikan representasi string dari tanggal yang ditentukan | | `toTimeString()` | Mengembalikan bagian *waktu* dalam format yang mudah dibaca | | `toUTCString()` | Mengubah tanggal menjadi string sesuai dengan format universal | | `toUTC()` | Mengembalikan milidetik sejak tengah malam 1 Januari 1970 dalam format UTC | | `valueOf()` | Mengembalikan nilai primitif dari `Date` |

Bab 11

JSON

Java**S**cript **O**bject **N**otation (JSON) adalah format berbasis teks untuk menyimpan dan mengangkut data. Objek JavaScript dapat dengan mudah dikonversi menjadi JSON dan sebaliknya. Contohnya.

```
// objek JavaScript
let myObj = { name: "Ryan", age: 30, city: "Austin" };

// dikonversi menjadi JSON:
let myJSON = JSON.stringify(myObj);
console.log(myJSON);
// Hasil: '{"name":"Ryan","age":30,"city":"Austin"}'

// dikonversi kembali menjadi objek JavaScript
let originalJSON = JSON.parse(myJSON);
console.log(originalJSON);

// Hasil: {name: 'Ryan', age: 30, city: 'Austin'}
```

`stringify` dan `parse` adalah dua metode yang didukung oleh JSON.

Metode	Deskripsi
`parse()`	Mengembalikan objek JavaScript dari string JSON yang diurai
`stringify()`	Mengembalikan string JSON dari Objek JavaScript

Tipe data berikut didukung oleh JSON.

- string
- number
- array
- boolean
- objek dengan nilai JSON yang valid
- `null`

Ini tidak dapat berupa `function` , `date` , atau `undefined` .

Panduan Penanganan Kesalahan JavaScript

Kesalahan merupakan bagian tak terhindarkan dalam pengembangan perangkat lunak. Menangani kesalahan dengan efektif sangat penting untuk menulis kode JavaScript yang kokoh dan dapat diandalkan. Panduan ini akan memandu Anda melalui dasar-dasar penanganan kesalahan dalam JavaScript, termasuk mengapa hal ini penting, jenis kesalahan, dan cara menggunakan pernyataan `try...catch`.

Mengapa Penanganan Kesalahan Penting

Penanganan kesalahan sangat penting karena beberapa alasan:

- **Pemulihan yang Mulus**: Ini memungkinkan kode Anda pulih dengan baik dari masalah yang tidak terduga dan melanjutkan eksekusi.
- **Pengalaman Pengguna**: Penanganan kesalahan yang efektif meningkatkan pengalaman pengguna dengan memberikan pesan kesalahan yang bermakna.
- **Pemecahan Masalah**: Kesalahan yang ditangani dengan baik membuat pemecahan masalah menjadi lebih mudah karena Anda dapat mengidentifikasi masalah dengan cepat.
- **Keandalan Kode**: Penanganan kesalahan memastikan bahwa kode Anda dapat diandalkan dan kokoh, mengurangi risiko terjadinya crash.

Jenis Kesalahan

Kesalahan JavaScript dapat dikategorikan menjadi beberapa jenis, termasuk:

- **Kesalahan Sintaksis**: Kesalahan yang terjadi karena sintaksis yang salah.
- **Kesalahan Runtime**: Kesalahan yang terjadi selama eksekusi kode.
- **Kesalahan Logika**: Kesalahan yang disebabkan oleh logika yang cacat dalam kode.

Penggunaan Umum

Penanganan permintaan jaringan yang mungkin gagal. Penguraian dan validasi masukan pengguna. Pengelolaan kesalahan pihak ketiga.

Keuntungan Penanganan Kesalahan

Penanganan kesalahan yang efektif menawarkan beberapa keuntungan:

-Mencegah penghentian skrip. -Memungkinkan penanganan kesalahan yang terkendali. -Memberikan informasi kesalahan yang rinci untuk keperluan pemecahan masalah. -Meningkatkan keandalan kode dan pengalaman pengguna.

Praktik Terbaik

Untuk memanfaatkan sebaik mungkin dari penanganan kesalahan, pertimbangkan praktik terbaik ini:

-Gunakan jenis kesalahan yang spesifik bila memungkinkan. -Catat kesalahan untuk tujuan pemecahan masalah. -Sediakan pesan kesalahan yang jelas dan ramah pengguna. -Tangani kesalahan sesegera mungkin setelah mereka terjadi.

Penanganan Kesalahan try...catch:

Salah satu teknik umum penanganan kesalahan adalah blok try...catch, yang dijelaskan dalam bagian-bagian berikut ini.

- try...catch
- try...catch...finally

Kesimpulan

Penanganan kesalahan adalah aspek kritis dalam pengembangan JavaScript. Dengan memahami jenis kesalahan, dan mengikuti praktik terbaik, Anda dapat menulis aplikasi yang lebih dapat diandalkan dan ramah pengguna.

Bab 12

Penanganan Kesalahan

Dalam pemrograman, kesalahan terjadi karena berbagai alasan, beberapa di antaranya disebabkan oleh kesalahan kode, beberapa disebabkan oleh input yang salah, dan hal-hal lain yang tidak dapat diprediksi. Ketika terjadi kesalahan, kode akan berhenti dan menghasilkan pesan kesalahan biasanya terlihat di konsol.

try... catch

Daripada menghentikan eksekusi kode, kita dapat menggunakan konstruksi `try...catch` yang memungkinkan penanganan kesalahan tanpa menghentikan skrip. Konstruksi `try...catch` terdiri dari dua blok utama: `try` dan kemudian `catch` .

```
try {
  // code...
} catch (err) {
  // error handling
}
```

Pertama, kode dalam blok `try` dieksekusi. Jika tidak ada kesalahan yang terjadi, maka blok `catch` dilewati. Namun, jika terjadi kesalahan, eksekusi dalam blok `try` dihentikan, dan kendali program dipindahkan ke blok `catch` . Penyebab kesalahan ditangkap dalam variabel `err` .

```
try {
  // kode...
  alert("Selamat datang di Belajar JavaScript");
  asdk; // kesalahan asdk variable tidak didefinisikan
} catch (err) {
  console.log("Kesalahan telah terjadi");
}
```

`try...catch` bekerja untuk kesalahan saat runtime, yang berarti kode harus dapat dijalankan dan bersifat sinkron.

Untuk melempar kesalahan khusus, Anda dapat menggunakan pernyataan `throw` . Objek kesalahan yang dihasilkan oleh kesalahan memiliki dua properti utama.

- **name**: nama kesalahan
- **message**: detail tentang kesalahan

Jika kita tidak memerlukan pesan kesalahan, blok `catch` dapat diabaikan.

try...catch...finally

Kita dapat menambahkan satu konstruksi lagi ke `try...catch` yang disebut `finally`, kode ini dieksekusi dalam semua situasi. Artinya, setelah `try` ketika tidak ada kesalahan dan setelah `catch` dalam kasus kesalahan. Berikut adalah sintaks untuk `try...catch...finally`.

```
try {
  // coba eksekusi kode
} catch (err) {
  // tangani kesalahan
} finally {
  // selalu dieksekusi
}
```

Menjalankan contoh kode dunia nyata.

```
try {
  alert("try");
} catch (err) {
  alert("catch");
} finally {
  alert("finally");
}
```

Di contoh di atas, blok `try` dieksekusi terlebih dahulu, yang kemudian diikuti oleh `finally` karena tidak ada kesalahan.

Exercise

Buatlah fungsi `divideNumbers()` yang mengambil dua argumen, yaitu pembilang (numerator) dan penyebut (denominator), dan mengembalikan hasil pembagian pembilang dengan penyebut dengan pengaturan berikut.

```
function divideNumbers(pembilang, penyebut) {
try {
// pernyataan try untuk membagi pembilang dengan penyebut.
} catch (error) {
// mencetak pesan kesalahan
} finally {
// mencetak bahwa eksekusi telah selesai
}
// mengembalikan hasil
}
let jawaban = divideNumbers(10, 2);
```

Bab 13

Modul

Di dunia nyata, sebuah program berkembang secara organik untuk mengatasi kebutuhan fungsionalitas baru. Dengan pertumbuhan kode yang semakin besar, struktur dan pemeliharaan kode memerlukan pekerjaan tambahan. Meskipun akan memberikan manfaat di masa depan, terkadang sulit untuk mengabaikannya dan membiarkan program menjadi sangat rumit. Dalam kenyataannya, hal ini meningkatkan kompleksitas aplikasi, karena seseorang harus memahami sistem secara menyeluruh dan sulit untuk memahami setiap bagian secara terpisah. Kedua, seseorang harus menginvestasikan lebih banyak waktu untuk memecahnya agar dapat menggunakan fungsionalitasnya.

Modul hadir untuk menghindari masalah ini. Sebuah `modul` menentukan potongan kode mana yang menjadi dependennya, bersama dengan fungsionalitas apa yang disediakannya untuk modul lainnya. Modul yang bergantung pada modul lain disebut *dependensi*. Berbagai perpustakaan modul digunakan untuk mengorganisir kode menjadi modul dan memuatnya sesuai permintaan.

- AMD - salah satu sistem modul tertua, awalnya digunakan oleh require.js.
- CommonJS - sistem modul yang dibuat untuk server Node.js.
- UMD - sistem modul yang kompatibel dengan AMD dan CommonJS.

Modul dapat memuat satu sama lain dan menggunakan direktif khusus `import` dan `export` untuk bertukar fungsionalitas dan memanggil fungsi satu sama lain.

- `export` - memberi label pada fungsi dan variabel yang harus dapat diakses dari luar modul saat ini.
- `import` - mengimpor fungsionalitas dari modul luar.

Mari kita lihat mekanisme `import` dan `export` dalam modul. Kami memiliki fungsi `sayHi` yang diekspor dari file `sayHi.js`.

```
// 📁 sayHi.js
export const sayHi = (user) => {
  alert(`Halo, ${user}!`);
};
```

Fungsi `sayHi` digunakan dalam file `main.js` dengan bantuan direktif `import`.

```
// 📁 main.js
import { sayHi } from "./sayHi.js";

alert(sayHi); // function...
sayHi("Kelvin"); // Halo, Kelvin!
```

Di sini, direktif impor memuat modul dengan mengimpor path relatif dan mengassign variabel `sayHi`.

Modul dapat diekspor dalam dua cara: **Named** dan **Default**. Selain itu, ekspor Nama dapat diberikan secara inline atau individu.

```
// 📁 person.js

// ekspor nama bernama
export const name = "Kelvin";
export const age = 30;

// sekaligus
const name = "Kelvin";
const age = 30;
export { name, age };
```

Hanya dapat ada satu ekspor `default` dalam satu file.

```
// 📁 message.js
const message = (name, age) => {
  return `${name} berusia ${age} tahun.`;
};
export default message;
```

Berdasarkan jenis ekspor, kita dapat mengimpornya dengan dua cara. Ekspor nama dibuat menggunakan kurung kurawal, sedangkan ekspor default tidak.

```
import { name, age } from "./person.js"; // impor ekspor nama
import message from "./message.js"; // impor ekspor default
```

Ketika mengassign modul, kita harus menghindari *ketergantungan berputar*. Ketergantungan berputar adalah situasi di mana modul A bergantung pada B, dan B juga bergantung pada A secara langsung atau tidak langsung.

Bab 14

Ekspresi Reguler

Ekspresi reguler adalah objek yang dapat dibuat dengan konstruktor `RegEx` atau ditulis sebagai nilai literal dengan menutupi pola di antara karakter garis miring ke depan (`/`). Berikut adalah sintaksis untuk membuat ekspresi reguler:

```
// menggunakan konstruktor ekspresi reguler
new RegExp(pola[, tanda]);

// menggunakan literal
/pola/modifier
```

Tanda tersebut bersifat opsional saat membuat ekspresi reguler menggunakan literal. Contoh pembuatan ekspresi reguler yang identik menggunakan kedua metode di atas adalah sebagai berikut.

```
let re1 = new RegExp("xyz");
let re2 = /xyz/;
```

Kedua cara tersebut akan membuat objek regex dan memiliki metode dan properti yang sama. Ada kasus di mana kita mungkin perlu nilai dinamis untuk membuat ekspresi reguler, dalam hal ini, literal tidak akan berfungsi, dan kita harus menggunakan konstruktor.

> Dalam kasus di mana kita ingin garis miring menjadi bagian dari ekspresi reguler, kita harus melarikan diri dari garis miring (`/`) dengan menggunakan garis miring terbalik (`\`).

Berbagai modifier yang digunakan untuk melakukan pencarian yang tidak memperhatikan kasus adalah:

- `g` - pencarian global (menemukan semua kecocokan daripada berhenti setelah kecocokan pertama)
- `i` - pencarian yang tidak memperhatikan huruf besar kecil
- `m` - pencocokan beberapa baris

Kurung digunakan dalam ekspresi reguler untuk menemukan rentang karakter. Beberapa di antaranya adalah:

- `[abc]` - temukan karakter apa pun di antara tanda kurung
- `[^abc]` - temukan karakter apa pun, tidak ada di antara tanda kurung
- `[0-9]` - temukan angka apa pun di antara tanda kurung
- `[^0-9]` - temukan karakter apa pun, tidak ada di antara tanda kurung (bukan angka)
- `(x|y)` - temukan salah satu alternatif yang dipisahkan oleh |

Metakarakter adalah karakter khusus yang memiliki makna khusus dalam ekspresi reguler. Karakter-karakter ini lebih lanjut dijelaskan di bawah ini:

Metakarakter	Deskripsi
`.`	Cocokkan satu karakter kecuali karakter baris baru atau terminator
`\w`	Cocokkan karakter kata (karakter alfanumerik `[a-zA-Z0-9_]`)
`\W`	Cocokkan karakter bukan kata (sama dengan `[^a-zA-Z0-9_]`)
`\d`	Cocokkan karakter angka apa pun (sama dengan `[0-9]`)
`\D`	Cocokkan karakter bukan digit
`\s`	Cocokkan karakter spasi (spasi, tab, dll.)
`\S`	Cocokkan karakter bukan spasi
`\b`	Cocokkan di awal / akhir kata
`\B`	Cocokkan tetapi bukan di awal / akhir kata
`\0`	Cocokkan karakter `NULL`
`\n`	Cocokkan karakter baris baru
`\f`	Cocokkan karakter makanan bentuk
`\r`	Cocokkan karakter kembali karbon
`\t`	Cocokkan karakter tab
`\v`	Cocokkan karakter tab vertikal
`\xxx`	Cocokkan karakter yang ditentukan oleh bilangan oktal `xxx`
`\xdd`	Cocokkan karakter yang ditentukan oleh bilangan heksadesimal `dd`
`\udddd`	Cocokkan karakter Unicode yang ditentukan oleh bilangan heksadesimal `dddd`

Properti dan metode yang didukung oleh RegEx tercantum di bawah ini.

Nama	Deskripsi
`constructor`	Mengembalikan fungsi yang membuat prototipe objek RegExp
`global`	Memeriksa apakah pengubah `g` diatur
`ignoreCase`	Memeriksa apakah pengubah `i` diatur
`lastIndex`	Menentukan indeks di mana pencocokan berikutnya harus dimulai
`multiline`	Memeriksa apakah pengubah `m` diatur

|

| `source` | Mengembalikan teks string | | `exec()` | Uji kecocokan dan mengembalikan kecocokan pertama, jika tidak ada kecocokan maka mengembalikan `null` | | `test()` | Uji kecocokan dan mengembalikan `true` atau `false` | | `toString()` | Mengembalikan nilai string dari ekspresi reguler |

> Metode `complie()` mengompilasi ekspresi reguler dan sudah tidak digunakan lagi.

Beberapa contoh ekspresi reguler ditunjukkan di sini.

```
let text = "The best things in life are free";
let result = /e/.exec(text); // mencari kecocokan huruf e dalam string
// Hasil: e

let helloWorldText = "Hello world!";
// Mencari "Hello"
let pola1 = /Hello/g;
let hasil1 = pola1.test(helloWorldText);
// Hasil1: true

let pola1String = pola1.toString();
// pola1String : '/Hello/g'
```

Bab 15

Classes

Kelas adalah template untuk membuat objek. Ini mengkapsulasi data dengan kode untuk bekerja dengan data. Kata kunci `class` digunakan untuk membuat kelas. Dan metode khusus yang disebut `constructor` digunakan untuk membuat dan menginisialisasi objek yang dibuat dengan kelas. Berikut adalah contoh kelas mobil.

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age() {
    let date = new Date();
    return date.getFullYear() - this.year;
  }
}

let myCar = new Car("Toyota", 2021);
console.log(myCar.age()); // 1
```

> Kelas harus didefinisikan sebelum digunakan.

Di dalam tubuh kelas, metode atau konstruktor didefinisikan dan dieksekusi dalam "mode ketat" (strict mode). Sintaks yang tidak mengikuti mode ketat akan menghasilkan kesalahan.

Static

Kata kunci `static` digunakan untuk mendefinisikan metode atau properti statis dalam sebuah kelas. Metode dan properti ini dipanggil dalam kelas itu sendiri.

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
}
let myCar = new Car("Toyota");

console.log(myCar.hello()); // Ini akan menghasilkan kesalahan
console.log(Car.hello(myCar));
// Hasil: Hello Toyota
```

Seseorang dapat mengakses metode atau properti statis dari metode statis lain dalam kelas yang sama menggunakan kata kunci `this`.

Inheritance

Warisan berguna untuk tujuan penggunaan kembali kode karena memperluas properti dan metode kelas yang ada. Kata kunci `extends` digunakan untuk membuat warisan kelas.

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return "saya punya " + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ", ini adalah sebuah " + this.model;
  }
}

let myCar = new Model("Toyota", "Camry");
console.log(myCar.show()); // saya punya Camry, ini adalah sebuah Toyota.
```

Prototipe dari kelas induk harus berupa `Object` atau `null`.

Metode `super` digunakan di dalam sebuah konstruktor dan merujuk ke kelas induk. Dengan ini, Anda dapat mengakses properti dan metode kelas induk.

Access Modifiers

`public` , `private` , dan `protected` adalah tiga akses modifikasi yang digunakan dalam kelas untuk mengontrol akses dari luar. Secara default, semua anggota (properti, field, metode, atau fungsi) dapat diakses secara publik dari luar kelas.

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
}
let myCar = new Car("Toyota");
console.log(Car.hello(myCar)); // Hello Toyota
```

Anggota `private` hanya dapat diakses secara internal dalam kelas dan tidak dapat diakses dari luar. Properti `private` harus dimulai dengan karakter `#` .

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
  #present(carname) {
    return "I have a " + this.carname;
  }
}
let myCar = new Car("Toyota");
console.log(myCar.#present("Camry")); // Error
console.log(Car.hello(myCar)); // Hello Toyota
```

Bidang `protected` hanya dapat diakses dari dalam kelas dan mereka yang memperluasnya. Ini berguna untuk antarmuka internal karena kelas yang mewarisi juga mendapatkan akses ke kelas induk. Bidang yang dilindungi dengan `_` .

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  _present() {
    return "Saya punya " + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this._present() + ", ini adalah sebuah " + this.model;
  }
}
let myCar = new Model("Toyota", "Camry");
console.log(myCar.show()); // Saya punya Toyota, itu Camry
```

Bab 16

Browser Object Model (BOM)

Model objek peramban memungkinkan kita untuk berinteraksi dengan jendela peramban. Objek `window` mewakili jendela peramban dan didukung oleh semua peramban.

Objek `window` adalah objek default untuk peramban, sehingga kita dapat menentukan `window` atau langsung memanggil semua fungsi.

```
window.alert("Selamat datang di Belajar JavaScript");

alert("Selamat datang di Belajar JavaScript");
```

Dengan cara yang serupa, kita dapat memanggil properti lain di bawah objek `window` seperti `history` , `screen` , `navigator` , `location` , dan sebagainya.

Window

Objek `window` mewakili jendela peramban dan didukung oleh peramban-peramban. Variabel global, objek, dan fungsi juga merupakan bagian dari objek `window`.

Variabel global adalah **properti**, dan fungsi adalah **metode** dari objek `window`. Mari kita lihat contoh sifat layar. Ini digunakan untuk menentukan ukuran jendela peramban dan diukur dalam piksel.

- `window.innerHeight` - tinggi dalam jendela peramban
- `window.innerWidth` - lebar dalam jendela peramban

> **Catatan**: `window.document` sama dengan `document.location` karena model objek dokumen (DOM) adalah bagian dari objek window.

Beberapa contoh metode objek window:

- `window.open()` - membuka jendela baru
- `window.close()` - menutup jendela saat ini
- `window.moveTo()` - memindahkan jendela saat ini
- `window.resizeTo()` - mengubah ukuran jendela saat ini

Popup

Pop-up adalah cara tambahan untuk menampilkan informasi, mengonfirmasi pengguna, atau mengambil masukan pengguna dari dokumen tambahan. Sebuah pop-up dapat menavigasi ke URL baru dan mengirim informasi ke jendela pembuka. **Kotak peringatan (Alert box)**, **Kotak konfirmasi (Confirmation box)**, dan **Kotak prompt (Prompt box)** adalah fungsi global di mana kita dapat menampilkan informasi pop-up.

1. **alert()**: Ini menampilkan informasi kepada pengguna dan memiliki tombol "**OK**" untuk melanjutkan.

   ```
   alert("Alert message example");
   ```

2. **confirm()**: Digunakan sebagai dialog box untuk mengonfirmasi atau menerima sesuatu. Ini memiliki tombol "**Ok**" dan "**Batal**" untuk melanjutkan. Jika pengguna mengklik "**Ok**," maka ini mengembalikan `true`, jika mengklik "**Batal**," maka ini mengembalikan `false`.

   ```
   let txt;
   if (confirm("Tekan tombol!")) {
     txt = "Anda menekan OK!";
   } else {
     txt = "Anda menekan Batal!";
   }
   ```

3. **prompt()**: Mengambil nilai masukan pengguna dengan tombol "**Ok**" dan "**Batal**". Ini mengembalikan `null` jika pengguna tidak memberikan nilai masukan apa pun.

   ```
   //syntax
   //window.prompt("sometext","defaultText");
   ```

let person = prompt("Silakan masukkan nama Anda", "Harry Potter");

if (person == null || person == "") { txt = "Pengguna membatalkan prompt."; } else { txt = "Halo " + person + "! Bagaimana kabarmu hari ini?"; }

```

```

Screen

Objek `screen` berisi informasi tentang layar di mana jendela saat ini dirender. Untuk mengakses objek `screen`, kita dapat menggunakan properti `screen` dari objek `window`.

```
window.screen;
//or
screen;
```

Objek `window.screen` memiliki berbagai properti, beberapa di antaranya tercantum di sini:

Properti	Deskripsi
`height`	Mewakili tinggi piksel layar.
`left`	Mewakili jarak piksel dari sisi kiri layar saat ini.
`pixelDepth`	Properti hanya-dibaca yang mengembalikan kedalaman bit layar.
`top`	Mewakili jarak piksel dari bagian atas layar saat ini.
`width`	Mewakili lebar piksel layar.
`orientation`	Mengembalikan orientasi layar sebagaimana yang ditentukan dalam Screen Orientation API.
`availTop`	Properti hanya-dibaca yang mengembalikan piksel pertama dari atas yang tidak diambil oleh elemen sistem.
`availWidth`	Properti hanya-dibaca yang mengembalikan lebar piksel layar yang tidak termasuk elemen sistem.
`colorDepth`	Properti hanya-dibaca yang mengembalikan jumlah bit yang digunakan untuk mewakili warna.

Navigator

`window.navigator` atau `navigator` adalah properti yang **hanya dapat dibaca** dan berisi berbagai metode dan fungsi terkait dengan peramban.

Mari kita lihat beberapa contoh navigasi.

1. **navigator.appName**: Ini memberikan nama aplikasi peramban.

```
navigator.appName;
// "Netscape"
```

> ***Catatan:*** "Netscape" adalah nama aplikasi yang digunakan untuk IE11, Chrome, Firefox, dan Safari.

2. **navigator.cookieEnabled**: Mengembalikan nilai boolean berdasarkan nilai cookie dalam peramban.

```
navigator.cookieEnabled;
//true
```

3. **navigator.platform**: Memberikan informasi tentang sistem operasi peramban.

```
navigator.platform;
("MacIntel");
```

Cookies 🍪

Cookie adalah potongan-potongan informasi yang disimpan di komputer dan dapat diakses oleh peramban (browser).

Komunikasi antara peramban web dan server bersifat tanpa keadaan (stateless), yang berarti setiap permintaan diperlakukan secara independen. Ada kasus di mana kita perlu menyimpan informasi pengguna dan membuat informasi tersebut tersedia untuk peramban. Dengan cookie, informasi dapat diambil dari komputer kapan pun diperlukan.

Cookie disimpan dalam bentuk pasangan nama-nilai.

```
book = Learn JavaScript
```

Properti `document.cookie` digunakan untuk membuat, membaca, dan menghapus cookie. Membuat cookie cukup mudah, Anda perlu menyediakan nama dan nilainya.

```
document.cookie = "book=Learn JavaScript";
```

Secara default, sebuah cookie akan terhapus ketika peramban ditutup. Untuk membuatnya bertahan, kita perlu menentukan tanggal kedaluwarsa (dalam waktu UTC).

```
document.cookie =
  "book=Learn JavaScript; expires=Fri, 08 Jan 2022 12:00:00 UTC";
```

Kita dapat menambahkan parameter untuk menentukan ke path mana cookie tersebut dimiliki. Secara default, cookie tersebut milik halaman saat ini.

```
document.cookie =
  "book=Learn JavaScript; expires=Fri, 08 Jan 2022 12:00:00 UTC; path=/";
```

Berikut adalah contoh sederhana dari sebuah cookie.

```
let cookies = document.cookie;
// Cara sederhana untuk mengambil semua cookie.

document.cookie =
  "book=Learn JavaScript; expires=Fri, 08 Jan 2022 12:00:00 UTC; path=/";
// Mengatur sebuah cookie
```

History

Ketika kita membuka peramban web dan menjelajahi sebuah halaman web, itu menciptakan entri baru dalam tumpukan riwayat. Saat kita terus menavigasi ke halaman-halaman yang berbeda, entri-entri baru dimasukkan ke dalam tumpukan.

Untuk mengakses objek riwayat, kita dapat menggunakan:

```
window.history;
// or
history;
```

Untuk menavigasi antara berbagai tumpukan riwayat, kita dapat menggunakan metode `go()`, `forward()`, dan `back()` dari objek **history**.

1. **go()**: Digunakan untuk menavigasi ke URL tertentu dalam tumpukan riwayat.

   ```
   history.go(-1); // berpindah halaman ke belakang
   history.go(0); // me-refresh halaman saat ini
   history.go(); // me-refresh halaman saat ini
   history.go(1); // berpindah halaman ke depan
   ```

> ***Catatan:*** Posisi halaman saat ini dalam tumpukan riwayat adalah **0**.

1. **back()**: Untuk berpindah halaman ke belakang, kita menggunakan metode `back()`.

   ```
   history.back();
   ```

2. **forward()**: Metode `forward()` memuat entri berikutnya dalam riwayat peramban. Ini mirip dengan mengklik tombol maju (forward) di peramban.

   ```
   history.forward();
   ```

Location

Objek `location` digunakan untuk mengambil lokasi saat ini (URL) dari dokumen dan menyediakan berbagai metode untuk memanipulasi lokasi dokumen. Lokasi saat ini dapat diakses dengan menggunakan:

```
window.location;
//or
document.location;
//or
location;
```

> ***Catatan***: `window.location` dan `document.location` merujuk pada objek lokasi yang sama.

Mari kita ambil contoh dari URL berikut dan jelajahi berbagai properti dari `location`.

`http://localhost:3000/js/index.html?type=listing&page=2#title`

```
location.href; // mencetak URL dokumen saat ini
location.protocol; // mencetak protokol seperti http: atau https:
location.host; // mencetak nama host dengan port seperti localhost atau localhost:3000
location.hostname; // mencetak nama host seperti localhost atau www.example.com
location.port; // mencetak nomor port seperti 3000
location.pathname; // mencetak jalur seperti /js/index.html
location.search; // mencetak string kueri seperti ?type=listing&page=2
location.hash; // mencetak pengenal fragmen seperti #title
```

Bab 17

Peristiwa

Dalam pemrograman, *peristiwa* adalah tindakan atau kejadian dalam suatu sistem yang sistem memberi tahu Anda sehingga Anda dapat meresponsnya. Misalnya, saat Anda mengklik tombol reset, itu akan menghapus input.

Interaksi dari keyboard seperti penekanan tombol perlu terus dibaca untuk menangkap keadaan tombol sebelum dilepaskan lagi. Melakukan komputasi yang memakan waktu lain dapat membuat Anda melewatkan penekanan tombol. Ini dulu adalah mekanisme penanganan masukan dari beberapa mesin primitif. Langkah lebih lanjut adalah dengan menggunakan antrian, yaitu program yang secara berkala memeriksa antrian untuk peristiwa baru dan meresponsnya. Pendekatan ini disebut *polling*.

Kekurangan utama dari pendekatan ini adalah bahwa harus melihat antrian dari waktu ke waktu, yang menyebabkan gangguan ketika suatu peristiwa terpicu. Mekanisme yang lebih baik untuk ini adalah memberi tahu kode ketika suatu peristiwa terjadi. Inilah yang dilakukan oleh peramban modern dengan memungkinkan kita mendaftarkan fungsi sebagai *handler* untuk peristiwa tertentu.

```
<p>Klik saya untuk mengaktifkan handler.</p>
<script>
  window.addEventListener("click", () => {
    console.log("diklik");
  });
</script>
```

Di sini, `addEventListener` dipanggil pada objek `window` (objek bawaan yang disediakan oleh peramban) untuk mendaftarkan handler untuk seluruh `window`. Memanggil metode `addEventListener`-nya mendaftarkan argumen kedua untuk dipanggil setiap kali peristiwa yang dijelaskan oleh argumen pertama terjadi.

> Pendengar peristiwa hanya dipanggil ketika peristiwa terjadi dalam konteks objek yang mendaftarkannya.

Beberapa peristiwa HTML umum disebutkan di sini.

Peristiwa	Deskripsi
`onchange`	Saat pengguna mengubah atau memodifikasi nilai input formulir
`onclick`	Saat pengguna mengklik elemen
`onmouseover`	Saat kursor mouse berada di atas elemen
`onmouseout`	Saat kursor mouse meninggalkan elemen
`onkeydown`	Saat pengguna menekan dan kemudian melepaskan tombol
`onload`	Saat peramban selesai memuat

Umumnya, handler yang terdaftar pada node dengan anak-anak juga akan menerima peristiwa dari anak-anak tersebut. Misalnya, jika sebuah tombol di dalam sebuah paragraf diklik, handler yang terdaftar pada paragraf juga akan menerima peristiwa klik. Jika ada handler pada keduanya, yang ada di bawah akan dieksekusi terlebih dahulu. Peristiwa dikatakan *menyebar* ke luar, dari node yang memulai ke node induknya dan pada akar dokumen.

Handler peristiwa dapat memanggil metode `stopPropagation` pada objek peristiwa untuk mencegah handler lebih atas menerima peristiwa. Ini berguna dalam kasus seperti Anda memiliki tombol di dalam elemen yang dapat diklik dan Anda tidak ingin memicu perilaku klik elemen luar dari klik tombol.

```
<p>Sebuah paragraf dengan <button>tombol</button>.</p>
<script>
  let para = document.querySelector("p"),
      button = document.querySelector("button");
  para.addEventListener("mousedown", () => {
    console.log("Handler paragraf.");
  });
  button.addEventListener("mousedown", event => {
    console.log("Handler tombol.");
    event.stopPropagation();
  });
</script>
```

Di sini, handler *`mousedown`* didaftarkan oleh paragraf dan tombol. Saat tombol diklik, handler tombol memanggil `stopPropagation`, yang akan mencegah handler pada paragraf dijalankan.

Peristiwa dapat memiliki perilaku default. Misalnya, tautan menavigasi ke tujuan tautan saat diklik, Anda akan diarahkan ke bawah halaman saat mengklik tombol panah bawah, dan sebagainya. Perilaku default ini dapat dicegah dengan memanggil metode `preventDefault` pada objek peristiwa.

```
<a href="https://developer.mozilla.org/">MDN</a>
<script>
  let link = document.querySelector("a");
  link.addEventListener("click", event => {
    console.log("Tidak.");
    event.preventDefault();
  });
</script>
```

Di sini, perilaku default tautan saat diklik dicegah, yaitu tidak menavigasi ke tujuan tautan.

Bab 18

Promise, async/await

Bayangkan Anda adalah seorang penulis buku terkenal, dan Anda berencana untuk merilis buku baru pada suatu hari tertentu. Para pembaca yang tertarik pada buku ini menambahkannya ke daftar keinginan mereka dan diberi tahu ketika buku tersebut diterbitkan atau bahkan jika tanggal rilisnya ditunda. Pada hari rilis, semua orang mendapatkan pemberitahuan dan dapat membeli buku tersebut, membuat semua pihak senang. Ini adalah analogi kehidupan nyata yang terjadi dalam pemrograman.

1. Kode "*produksi*" adalah sesuatu yang membutuhkan waktu dan mencapai sesuatu. Di sini, ini adalah penulis buku.
2. Kode "*konsumsi*" adalah seseorang yang mengonsumsi "kode produksi" begitu sudah siap. Dalam kasus ini, itu adalah seorang "pembaca".
3. Keterkaitan antara "kode produksi" dan "kode konsumsi" dapat disebut sebagai *promise* karena itu menjamin untuk mendapatkan hasil dari "kode produksi" ke "kode konsumsi".

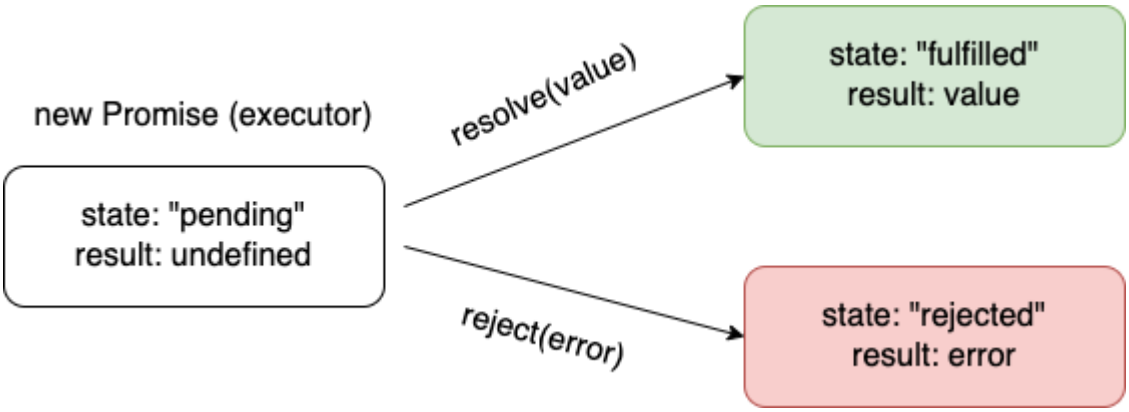
Promise

Analogi yang kita buat juga berlaku untuk objek `promise` JavaScript. Sintaks konstruktor untuk objek `promise` adalah:

```
let promise = new Promise(function (resolve, reject) {
  // executor (kode produksi, "penulis")
});
```

Di sini, sebuah fungsi dilewatkan ke `new Promise` yang juga dikenal sebagai *executor*, dan berjalan secara otomatis saat pembuatan. Ini berisi kode produksi yang memberikan hasil. `resolve` dan `reject` adalah argumen yang disediakan oleh JavaScript itu sendiri dan salah satu dari mereka dipanggil saat hasilnya ada.

- `resolve(value):` fungsi panggilan balik yang mengembalikan `value` saat hasil ada.
- `reject(error):` fungsi panggilan balik yang mengembalikan `error` saat terjadi kesalahan, mengembalikan objek kesalahan.



Properti internal objek `promise` yang dikembalikan oleh konstruktor `new Promise` adalah sebagai berikut:

- `state` - awalnya `pending`, kemudian berubah menjadi `fulfill` saat `resolve` atau `rejected` saat `reject` dipanggil

- `result` - awalnya `undefined`, kemudian berubah menjadi `value` saat `resolve` atau `error` saat `reject` dipanggil

> Tidak dapat mengakses properti `promise`: `state` dan `result`. Diperlukan metode promise untuk menangani promise.

Contoh promise.

```
let promiseSatu = new Promise(function (resolve, reject) {
  // fungsi ini dieksekusi secara otomatis saat promise dibuat

  // setelah 1 detik, sinyalkan bahwa pekerjaan selesai dengan hasil "done"
  setTimeout(() => resolve("done"), 1000);
});

let promiseDua = new Promise(function (resolve, reject) {
  // fungsi ini dieksekusi secara otomatis saat promise dibuat

  // setelah 1 detik, sinyalkan bahwa pekerjaan selesai dengan hasil "error"
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});
```

Di sini, `promiseSatu` adalah contoh "*promise yang dipenuhi*" karena berhasil memecahkan nilai, sementara `promiseDua` adalah "*promise yang ditolak*" karena ditolak. Promise yang tidak ditolak atau dipenuhi disebut *promise yang diselesaikan*, berbeda dengan promise awalnya *pending*. Fungsi konsumsi dari promise dapat didaftarkan menggunakan metode `.then` dan `.catch`. Kami juga dapat menambahkan metode `.finally` untuk melakukan pembersihan atau penyelesaian setelah metode sebelumnya selesai.

```
let promiseSatu = new Promise(function (resolve, reject) {
  setTimeout(() => resolve("done!"), 1000);
});

// resolve menjalankan fungsi pertama di .then
promiseSatu.then(
  (result) => alert(result), // menampilkan "done!" setelah 1 detik
  (error) => alert(error) // tidak dijalankan
);

let promiseDua = new Promise(function (resolve, reject) {
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// reject menjalankan fungsi kedua di .then
promiseDua.then(
  (result) => alert(result), // tidak dijalankan
  (error) => alert(error) // menampilkan "Error: Whoops!" setelah 1 detik
);

let promiseTiga = new Promise((resolve, reject) => {
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// .catch(f) sama dengan promise.then(null, f)
promiseTiga.catch(alert); // menampilkan "Error: Whoops!" setelah 1 detik
```

> Pada metode `Promise.then()`, kedua argumen panggilan balik adalah opsional.

Async/Await

Dengan promises, seseorang dapat menggunakan kata kunci `async` untuk mendeklarasikan fungsi asinkron yang mengembalikan promise, sedangkan sintaks `await` membuat JavaScript menunggu sampai promise tersebut selesai dan mengembalikan nilainya. Kata kunci-kata kunci ini membuat promises lebih mudah untuk ditulis. Contoh async ditunjukkan di bawah ini.

```
// fungsi async f
async function f() {
  return 1;
}
// promise yang di-resolve
f().then(alert); // 1
```

Contoh di atas dapat ditulis seperti berikut:

```
function f() {
  return Promise.resolve(1);
}

f().then(alert); // 1
```

`async` memastikan bahwa fungsi mengembalikan promise, dan membungkus non-promises di dalamnya. Dengan `await` , kita dapat membuat JavaScript menunggu sampai promise tersebut selesai dengan nilai yang dikembalikan.

```
async function f() {
  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("Selamat datang di Belajar JavaScript!"), 1000);
  });

  let result = await promise; // tunggu sampai promise tersebut diselesaikan (*)
  alert(result); // "Selamat datang di Belajar JavaScript!"
}

f();
```

Kata kunci `await` hanya dapat digunakan di dalam fungsi `async` .

Bab 19

Beragam

Dalam bab ini, kita akan membahas berbagai topik yang akan muncul saat menulis kode. Topik-topik tersebut tercantum di bawah ini:

- Template Literals
- Hoisting
- Currying
- Polyfills dan Transpilers
- Linked List
- Jejak Global
- Debugging
- Callback
- Web API dan AJAX
- Sifat Thread Tunggal
- ECMA Script
- Membangun dan Mendeploy Aplikasi JS
- Pengujian

Template Literal

Template literal adalah literal yang dibatasi dengan tanda kutip belakang (``) dan digunakan dalam interpolasi variabel dan ekspresi ke dalam string.

```
let teks = `Halo Dunia!`;
// template literal dengan tanda kutip tunggal dan ganda dalam satu string
let teks = `Dia sering dipanggil "Johnny"`;
// template literal dengan string beberapa baris
let teks = `Rubah coklat
cepat
melompati
anjing malas`;

// template literal dengan interpolasi variabel
const namaDepan = "John";
const namaBelakang = "Doe";

const teksSelamatDatang = `Selamat datang ${namaDepan}, ${namaBelakang}!`;

// template literal dengan interpolasi ekspresi
const harga = 10;
const PPN = 0.25;

const total = `Total: ${(harga * (1 + PPN)).toFixed(2)}`;
```

Hoisting

Hoisting adalah perilaku default dalam JavaScript yang memindahkan deklarasi ke bagian atas. Saat menjalankan kode, itu menciptakan konteks eksekusi global: pembuatan dan eksekusi. Pada tahap penciptaan, JavaScript memindahkan deklarasi variabel dan fungsi ke bagian atas halaman, yang dikenal sebagai hoisting.

```
// Hoisting variabel
console.log(counter);
let counter = 1; // menghasilkan ReferenceError: Cannot access 'counter' before initialization
```

Meskipun `counter` ada di dalam memori heap tetapi belum diinisialisasi, sehingga itu menghasilkan kesalahan. Ini terjadi karena hoisting, variabel `counter` diangkat ke sini.

```
// Hoisting fungsi
const x = 20,
  y = 10;

let result = add(x, y); // ❌ Uncaught ReferenceError: add is not defined
console.log(result);

let add = (x, y) => x + y;
```

Di sini, fungsi `add` diangkat dan diinisialisasi dengan `undefined` di dalam memori heap pada tahap penciptaan konteks eksekusi global. Oleh karena itu, menghasilkan kesalahan.

Currying

`Currying` adalah teknik canggih dalam pemrograman fungsional yang terdiri dari mengubah fungsi dengan beberapa argumen menjadi urutan fungsi bersarang. Ini mengubah fungsi dari yang dapat dipanggil `f(a, b, c)` menjadi yang dapat dipanggil seperti `f(a)(b)(c)`. Ini tidak memanggil fungsi, melainkan mengubahnya.

Untuk memahami currying lebih baik, mari buat fungsi `add` sederhana yang mengambil tiga argumen dan mengembalikan jumlah dari mereka. Kemudian, kita akan membuat fungsi `addCurry` yang mengambil satu input dan mengembalikan serangkaian fungsi dengan hasil penjumlahannya.

```
// Versi non-curried
const add = (a, b, c) => {
  return a + b + c;
};
console.log(add(2, 3, 5)); // 10

// Versi curried
const addCurry = (a) => {
  return (b) => {
    return (c) => {
      return a + b + c;
    };
  };
};
console.log(addCurry(2)(3)(5)); // 10
```

Di sini, kita dapat melihat bahwa baik versi curried maupun versi non-curried mengembalikan hasil yang sama. Currying dapat bermanfaat dalam banyak hal, beberapa di antaranya disebutkan di sini.

- Ini membantu menghindari pengiriman ulang variabel yang sama berkali-kali.
- Ini membagi fungsi menjadi potongan-potongan kecil dengan satu tanggung jawab, menjadikan fungsi kurang rentan terhadap kesalahan.
- Ini digunakan dalam pemrograman fungsional untuk membuat fungsi berorde tinggi.

Polyfills dan Transpilers

JavaScript terus berkembang. Secara berkala, proposal bahasa baru diajukan, dianalisis, dan ditambahkan ke https://tc39.github.io/ecma262/ dan kemudian diintegrasikan ke dalam spesifikasi. Mungkin ada perbedaan dalam cara implementasinya di mesin JavaScript tergantung pada browser. Beberapa mungkin mengimplementasikan proposal awal, sementara yang lain menunggu hingga seluruh spesifikasinya dirilis. Masalah kompatibilitas mundur muncul ketika hal-hal baru diperkenalkan.

Untuk mendukung kode modern di browser lama, kita menggunakan dua alat: `transpiler` dan `polyfill`.

Transpiler

Ini adalah program yang menerjemahkan kode modern dan menulis ulangnya menggunakan konstruksi sintaks yang lebih lama sehingga mesin yang lebih lama dapat memahaminya. Misalnya, operator "nullish coalescing" `??` diperkenalkan pada tahun 2020, dan browser lama tidak dapat memahaminya.

Sekarang, tugas transpiler adalah membuat operator "nullish coalescing" `??` menjadi dapat dimengerti oleh browser lama.

```
// sebelum menjalankan transpiler
height = height ?? 200;

// setelah menjalankan transpiler
height = (height !== undefined dan height !== null) ? height: 200;
```

> Babel adalah salah satu transpiler yang paling terkemuka. Dalam proses pengembangan, kita dapat menggunakan alat pembangunan seperti webpack atau parcel untuk men-transpile kode.

Polyfills

Ada saat-saat ketika fungsionalitas baru tidak tersedia di mesin browser yang ketinggalan zaman. Dalam kasus ini, kode yang menggunakan fungsionalitas baru tidak akan berfungsi. Untuk mengisi celah, kita menambahkan fungsionalitas yang hilang yang disebut `polyfill`. Misalnya, metode `filter()` diperkenalkan dalam ES5 dan tidak didukung di beberapa browser lama. Metode ini menerima sebuah fungsi dan mengembalikan sebuah array yang hanya berisi nilai-nilai dari array asli yang fungsi tersebut mengembalikan `true`

```
const arr = [1, 2, 3, 4, 5, 6];
const filtered = arr.filter((e) => e % 2 === 0); // menghapus angka genap
console.log(filtered);

// [2, 4, 6]
```

Polyfill untuk metode filter adalah.

```
Array.prototype.filter = function (callback) {
  // Simpan array baru
  const result = [];
  for (let i = 0; i < this.length; i++) {
    // panggil callback dengan elemen saat ini, indeks, dan konteks.
    // jika lulus uji maka tambahkan elemen ke dalam array baru.
    if (callback(this[i], i, this)) {
      result.push(this[i]);
    }
  }
  // kembalikan array
  return result;
};
```

caniuse menunjukkan fungsionalitas dan sintaks yang diperbarui yang didukung oleh mesin browser yang berbeda.

```
---
bab: 19
nomor halaman: 98
keterangan: Linked List adalah struktur data linear yang digunakan untuk menyimpan kumpulan elemen, yang disebu

# Linked List

Ini adalah struktur data umum yang ditemukan dalam semua bahasa pemrograman. Linked List sangat mirip dengan ar

Di sini setiap elemen dalam daftar adalah objek terpisah yang berisi tautan atau pointer ke elemen berikutnya.
```

["satu", "dua", "tiga", "empat"]

```
**Jenis-Jenis Linked List**

Ada tiga jenis linked list yang berbeda:

1. **Singly Linked Lists:** Setiap node hanya berisi satu pointer ke node berikutnya.
2. **Doubly Linked Lists:** Ada dua pointer pada setiap node, satu ke node berikutnya dan satu ke node sebelumn
3. **Circular Linked Lists:** Circular linked list membentuk lingkaran dengan node terakhir yang menunjuk ke no

# Tambah

Metode `add` dibuat di sini untuk menambahkan nilai ke linked list.

```javascript
class Node {
    constructor(data) {
        this.data = data
        this.next = null
    }
}

class LinkedList {
    constructor(head) {
        this head = head
    }
    append = (value) => {
        const newNode = new Node(value)
        let current = this.head
        if (!this.head) {
            this head = newNode
            return
        }
        while (current.next) {
            current = current.next
        }
        current.next = newNode
    }
}
```
```

Hapus

Di sini, metode `pop` dibuat untuk menghapus nilai dari linked list.

```
class Node {
    constructor(data) {
        this data = data
        this.next = null
    }
}

class LinkedList {
    constructor(head) {
        this head = head
    }
    pop = () => {
        let current = this head
        while (current.next.next) {
            current = current.next
        }
        current.next = current.next.next
    }
}
```

Sisipkan di Depan

Di sini, metode `prepend` dibuat untuk menambahkan nilai sebelum anak pertama dalam linked list.

```
class Node {
    constructor(data) {
        this data = data
        this.next = null
    }
}

class LinkedList {
    constructor(head) {
        this head = head
    }
    prepend = (value) => {
        const newNode = new Node(value)
        if (!this.head) {
            this head = newNode
        }
        else {
            newNode.next = this head
            this head = newNode
        }
    }
}
```

Geser

Di sini, metode `shift` dibuat untuk menghapus elemen pertama dari Linked List.

```
class Node {
    constructor(data) {
        this data = data
        this.next = null
    }
}

class LinkedList {
    constructor(head) {
        this head = head
    }
    shift = () => {
        this head = this.head.next
    }
}
```

Jejak Global

Jika Anda sedang mengembangkan sebuah modul, yang mungkin akan berjalan pada sebuah halaman web, yang juga menjalankan modul-modul lain, maka Anda harus waspada terhadap tumpang tindih nama variabel.

Misalkan kita sedang mengembangkan modul penghitung:

```
let myCounter = {
  number: 0,
  plusPlus: function () {
    this.number = this.number + 1;
  },
  isGreaterThanTen: function () {
    return this.number > 10;
  },
};
```

> ***Catatan:*** Teknik ini sering digunakan dengan closure, untuk membuat status internal tidak dapat diubah dari luar.

Modul ini sekarang hanya menggunakan satu nama variabel — `myCounter` . Jika modul lain pada halaman tersebut menggunakan nama seperti `number` atau `isGreaterThanTen` , maka itu aman karena kita tidak akan menimpa nilai-nilai satu sama lain.

Debugging

Dalam pemrograman, kesalahan dapat terjadi saat menulis kode. Ini bisa disebabkan oleh kesalahan sintaksis atau logika. Proses menemukan kesalahan bisa memakan waktu dan sulit, dan disebut debugging kode.

Untungnya, sebagian besar browser modern dilengkapi dengan debugger bawaan. Debugger ini dapat diaktifkan dan dinonaktifkan, memaksa kesalahan dilaporkan. Juga memungkinkan untuk mengatur breakpoint selama eksekusi kode untuk menghentikan eksekusi dan memeriksa variabel. Untuk ini, seseorang harus membuka jendela debugging dan menempatkan kata kunci `debugger` dalam kode JavaScript. Eksekusi kode dihentikan di setiap breakpoint, memungkinkan pengembang untuk memeriksa nilai-nilai JavaScript dan melanjutkan eksekusi kode.

Seseorang juga dapat menggunakan metode `console.log()` untuk mencetak nilai-nilai JavaScript di jendela debugger.

```
const a = 5,
  b = 6;
const c = a + b;
console.log(c);
// Hasil: c = 11;
```

Alat Pengembang Browser

Browser modern dilengkapi dengan alat pengembang yang kuat untuk membantu debugging JavaScript, memeriksa HTML dan CSS, serta memonitor permintaan jaringan. Berikut adalah gambaran singkat tentang beberapa alat penting:

Chrome DevTools: Alat pengembang Google Chrome menawarkan berbagai fitur untuk debugging aplikasi web.

Firefox DevTools: Alat pengembang Mozilla Firefox adalah pilihan yang sangat baik, dengan kemampuan serupa.

Microsoft Edge DevTools: Untuk pengguna Microsoft Edge, alat pengembang bawaan menyediakan fitur debugging penting.

Safari Web Inspector: Safari's Web Inspector adalah seperangkat alat yang kuat untuk debugging dan profil aplikasi web.

Menggunakan Breakpoint

Browser modern menawarkan alat pengembang dengan kemampuan debugging. Atur breakpoint untuk menghentikan eksekusi kode dan memeriksa variabel dan tumpukan panggilan. Langkah melalui kode untuk memahami alirannya. Alat Pengembang Browser

Browser menyediakan seperangkat alat pengembang yang memungkinkan Anda untuk memeriksa HTML, CSS, dan JavaScript. Anda dapat mengaksesnya dengan mengklik kanan pada halaman web dan memilih "Inspect" atau dengan menekan `F12` atau `Ctrl+Shift+I` . Fitur utama meliputi:

Konsol: Lihat dan interaksi dengan keluaran konsol.

Elemen: Periksa dan modifikasi DOM.

Sumber: Debug JavaScript dengan breakpoint dan ekspresi pantau.

Jaringan: Pantau permintaan dan respons jaringan. Menggunakan Pernyataan Debugger

Masukkan pernyataan debugger dalam kode Anda untuk membuat breakpoint secara berprogram. Ketika kode menemui debugger, itu akan menghentikan eksekusi dan membuka alat pengembang browser.

Membangun dan Mendeploy Aplikasi JavaScript

Mengembangkan dan mendeploy aplikasi JavaScript melibatkan serangkaian langkah mulai dari menyiapkan lingkungan pengembangan hingga mendeploy aplikasi di server web atau platform hosting. Berikut adalah panduan rinci untuk membantu individu melalui proses ini:

Menyiapkan Lingkungan Pengembangan

Sebelum memulai proses pengembangan, sangat penting bagi pengembang untuk memastikan bahwa Node.js dan npm (Node Package Manager) terinstal di sistem mereka. Alat penting ini dapat diperoleh dari situs web resmi Node.js Node.js. Selain itu, pengembang harus memilih editor kode atau Lingkungan Pengembangan Terpadu (IDE) yang sesuai untuk pengembangan JavaScript. Beberapa pilihan populer termasuk Visual Studio Code, Sublime Text, dan WebStorm.

Instalasi Node.js dan npm memberikan akses ke alat dan pustaka penting yang diperlukan untuk pengembangan JavaScript. Pemilihan editor kode atau IDE yang sesuai dapat secara signifikan meningkatkan produktivitas dan kualitas kode.

Memilih Kerangka Kerja atau Pustaka JavaScript

Pemilihan kerangka kerja atau pustaka JavaScript bergantung pada persyaratan khusus proyek yang sedang dikerjakan. Pengembang dapat memilih untuk bekerja dengan kerangka kerja yang sudah mapan seperti React, Angular, Vue.js, atau tetap menggunakan JavaScript murni, tergantung pada kompleksitas dan tuntutan proyek. Pemilihan ini pada dasarnya dipandu oleh kebutuhan akan struktur dan komponen yang telah ada yang dapat mempercepat proses pengembangan dan meningkatkan kemudahan pemeliharaan.

Membuat Proyek

Inisiasi proyek difasilitasi dengan menggunakan manajer paket seperti npm atau yarn untuk membuat proyek baru. Misalnya, eksekusi perintah `npm init` dapat digunakan untuk membuat proyek Node.js baru. Penggunaan manajer paket selama inisiasi proyek memastikan pembentukan struktur proyek yang terstandarisasi dan menyederhanakan manajemen dependensi. Pendekatan ini secara signifikan membantu dalam menjaga keteraturan dan kemudahan pengelolaan proyek.

Pengembangan Aplikasi

Selama proses pengkodean aplikasi JavaScript, pengembang disarankan untuk dengan cermat mengorganisasi modul dan komponen. Praktik ini penting untuk memungkinkan kemudahan pemeliharaan di masa depan. Pengembangan kode yang terorganisir dan modular sangat penting untuk memastikan aplikasi tetap mudah dipelihara dan mudah didebug. Selain itu, pendekatan ini mendorong penggunaan kembali kode dan mendorong kolaborasi antara pengembang yang bekerja pada proyek tersebut.

Pengujian Aplikasi

Pengembang didorong untuk membuat pengujian unit dan pengujian integrasi dengan menggunakan kerangka kerja pengujian seperti Jest, Mocha, atau Jasmine. Praktik ini bertujuan untuk memverifikasi bahwa aplikasi berfungsi sesuai dengan tujuan yang ditentukan. Pembuatan pengujian berfungsi sebagai langkah preventif untuk mengidentifikasi dan mengatasi potensi bug dengan antisipasi, sehingga meningkatkan kepercayaan terhadap keandalan aplikasi.

Membangun Aplikasi

Untuk mengoptimalkan kode JavaScript, CSS, dan aset untuk produksi, disarankan untuk menggunakan alat pembangunan yang sesuai seperti Webpack, Parcel, atau Rollup. Alat-alat ini menggabungkan dan mengoptimalkan kode dan aset, menghasilkan waktu pemuatan yang lebih cepat dan kinerja yang lebih baik. Selain itu, mereka berkontribusi pada organisasi kode dan memfasilitasi segregasi kepentingan dalam aplikasi.

Konfigurasi Deploy

Pengembang harus membuat keputusan yang bijaksana mengenai lokasi deploy. Pilihan deploy mencakup hosting web tradisional, layanan cloud seperti AWS atau Google Cloud, atau platform seperti Netlify, Vercel, atau GitHub Pages. Pemilihan platform deploy harus sesuai dengan persyaratan dan kendala anggaran proyek. Platform yang berbeda menawarkan tingkat skalabilitas, keamanan, dan kemudahan penggunaan yang beragam.

Membuat Build Produksi

Meng

hasilkan versi siap produksi dari aplikasi melibatkan menjalankan proses build. Hal ini umumnya melibatkan minimasi dan optimisasi kode, yang menghasilkan penggunaan bandwidth yang lebih sedikit dan pengalaman pengguna yang lebih baik. Selain itu, build produksi memastikan bahwa aplikasi berperforma optimal di lingkungan produksi.

Mendeploy Aplikasi

Proses deploy mensyaratkan kepatuhan ketat terhadap petunjuk yang diberikan oleh platform hosting. Ini mungkin melibatkan penggunaan FTP, SSH, atau alat deploy khusus platform. Mematuhi praktik terbaik selama deploy sangat penting untuk memastikan akses pengguna yang lancar ke aplikasi. Deploy dapat dicapai melalui berbagai cara, termasuk pengunggahan manual atau alur kerja deploy otomatis.

Konfigurasi Domain dan DNS (jika diperlukan)

Bagi mereka yang menggunakan domain kustom, mengonfigurasi pengaturan DNS untuk mengarahkan lalu lintas ke penyedia hosting atau server adalah langkah yang diperlukan. Konfigurasi ini memungkinkan pengguna mengakses aplikasi melalui nama domain yang ramah pengguna, meningkatkan branding dan aksesibilitas.

Integrasi Berkelanjutan dan Deploy Berkelanjutan (CI/CD)

Pengembang dapat memilih untuk mendirikan alur kerja Integrasi Berkelanjutan dan Deploy Berkelanjutan (CI/CD). Hal ini dapat dicapai melalui penggunaan alat CI/CD seperti Jenkins, Travis CI, CircleCI, atau GitHub Actions. Otomatisasi proses pengujian dan deploy sebagai tanggapan terhadap perubahan kode mengurangi potensi

kesalahan manusia dan memastikan bahwa perubahan kode mengalami pengujian yang ketat sebelum mencapai lingkungan produksi. Pendekatan ini secara signifikan meningkatkan kualitas dan keandalan kode.

Pemantauan dan Pemeliharaan

Setelah deploy, diperlukan kewaspadaan untuk memantau aplikasi terhadap kesalahan, masalah kinerja, dan kerentanan keamanan. Memperbarui dependensi secara berkala sangat penting untuk meningkatkan keamanan dan memanfaatkan fitur-fitur baru. Pendekatan proaktif ini menjamin bahwa aplikasi tetap andal, aman, dan bertenaga seiring berjalannya waktu.

Skalabilitas (jika diperlukan)

Dalam skenario di mana aplikasi mengalami pertumbuhan dan peningkatan lalu lintas dan beban kerja, skalabilitas infrastruktur mungkin menjadi penting. Penyedia layanan cloud menawarkan solusi yang dirancang untuk mengakomodasi persyaratan skalabilitas tersebut. Solusi-solusi ini memungkinkan aplikasi mengelola beban yang lebih tinggi dengan lancar sambil mempertahankan kinerja dan ketersediaan.

Cadangan dan Pemulihan Bencana (jika diperlukan)

Penerapan strategi cadangan dan pemulihan bencana sangat penting untuk melindungi data aplikasi dalam kejadian gangguan tak terduga. Strategi-strategi ini penting untuk memastikan kelangsungan bisnis dan mengurangi risiko kehilangan data selama peristiwa tak terduga.

Fungsi Callback dalam JavaScript

Fungsi callback adalah konsep dasar dalam JavaScript yang memungkinkan pemrograman asinkron dan berbasis peristiwa. Dokumen Markdown ini memberikan penjelasan mendalam tentang fungsi callback, tujuan mereka, dan cara menggunakannya dengan efektif.

Apa Itu Fungsi Callback?

- **Fungsi callback** adalah fungsi JavaScript yang diteruskan sebagai argumen ke fungsi lain.
- Biasanya dijalankan atau dieksekusi pada waktu yang lebih lambat, seringkali setelah beberapa operasi asinkron atau peristiwa.
- Callback penting untuk menangani tugas seperti pengambilan data, penanganan peristiwa, dan penanganan perilaku asinkron.

Mengapa Menggunakan Fungsi Callback?

- **Operasi Asinkron**: Callback penting untuk mengelola operasi asinkron seperti membaca file, permintaan API, dan timer.
- **Penanganan Peristiwa**: Digunakan untuk merespons peristiwa seperti klik tombol, masukan pengguna, atau respons jaringan.
- **Kode Modular**: Callback membantu menulis kode modular dan dapat digunakan kembali dengan memisahkan kepentingan dan mempromosikan prinsip tanggung jawab tunggal.

Anatomi Fungsi Callback

Sebuah fungsi callback umumnya memiliki struktur berikut:

```
function fungsiCallback(arg1, arg2, ..., callback) {
    // Lakukan beberapa operasi
    // ...

    // Panggil fungsi callback ketika selesai
    callback(hasil);
}
```

- **fungsiCallback** adalah fungsi yang mengambil callback sebagai argumen. Ini dapat melakukan beberapa operasi secara asinkron.
- Pada akhirnya, ia memanggil fungsi **callback**, mengirimkan hasil atau kesalahan.

Penanganan Kesalahan dalam Callback

Dalam JavaScript, fungsi callback dapat menangani kesalahan dengan konvensi. Umumnya, objek kesalahan dikirimkan sebagai argumen pertama atau argumen kedua digunakan untuk mewakili kesalahan. Pengembang harus memeriksa kesalahan dan menanganinya dengan benar dalam fungsi callback.

Pendekatan Alternatif untuk Callback

1. **Promises**: Promises menawarkan cara terstruktur untuk mengelola kode asinkron dan kesalahan. Mereka memiliki tiga status: tertunda, terpenuhi, dan ditolak. Promises menggunakan metode `.then()` dan `.catch()` untuk menangani skenario keberhasilan dan kesalahan.
2. **Async/Await**: Async/await adalah tambahan yang lebih baru dalam JavaScript. Ini menyederhanakan kode asinkron dengan memungkinkan pengembang menulisnya dengan gaya yang lebih bersifat sinkron. Ini dibangun di atas Promises dan sangat berguna untuk menangani operasi asinkron dengan alur kode yang lebih linier.
3. **Pengirim Acara**: Di Node.js, kelas `EventEmitter` memungkinkan Anda untuk membuat arsitektur berbasis peristiwa khusus untuk menangani tugas-tugas asinkron.

Callback Hell (Piramida Callback) dan Contoh

Callback hell, juga dikenal sebagai "piramida malapetaka," adalah masalah umum dalam JavaScript saat bekerja dengan fungsi callback yang bersarang dalam fungsi lain. Fenomena ini terjadi ketika beberapa operasi asinkron dihubungkan satu sama lain, membuat kode sulit dibaca dan dipelihara. Dokumen Markdown ini menjelaskan callback hell dan memberikan contoh sederhana.

Apa Itu Callback Hell?

- **Callback hell** terjadi ketika fungsi asinkron bersarang satu sama lain, menghasilkan struktur kode yang sangat mendalam.
- Ini membuat kode lebih sulit dipahami, didebug, dan dipelihara karena tingkat indentasi yang berlebihan.
- Callback hell seringkali muncul saat menangani beberapa operasi asinkron secara berurutan, seperti membuat permintaan API atau membaca/menulis file.

Contoh Callback Hell

```
operasiAsinkron1(function (hasil1) {
  // Callback 1
  operasiAsinkron2(hasil1, function (hasil2) {
    // Callback 2
    operasiAsinkron3(hasil2, function (hasil3) {
      // Callback 3
      operasiAsinkron4(hasil3, function (hasil4) {
        // Callback 4
        operasiAsinkron5(hasil4, function (hasil5) {
          // Callback 5
          // ... dan seterusnya
        });
      });
    });
  });
});
```

Masalah dengan Callback Hell

- **Keterbacaan**: Callback hell mengakibatkan kode yang sangat bersarang, membuatnya sulit dibaca dan dipahami. Ini dapat menghambat ulasan kode dan kolaborasi.

- **Kemudahan Pemeliharaan**: Saat lebih banyak operasi asinkron ditambahkan, callback hell membuat kode sulit dipelihara. Memodifikasi fungsionalitas yang ada atau menambah fitur baru menjadi berisiko kesalahan.
- **Penanganan Kesalahan**: Mengelola kesalahan menjadi kompleks dalam fungsi callback bersarang. Menangani pengecualian dan menyebar kesalahan ke tingkat yang lebih tinggi bisa menjadi tantangan.

Mengurangi Callback Hell

1. Fungsi Beranama

- Pecah fungsi callback menjadi fungsi berbeda yang berdiri sendiri. Ini meningkatkan keterbacaan kode dengan memberikan nama yang bermakna pada setiap fungsi.

2. Promises

- Promises menyediakan cara yang lebih terstruktur untuk mengelola kode asinkron. Mereka memungkinkan Anda untuk mengaitkan operasi asinkron, membuat kode lebih linear dan lebih mudah dibaca.

# #

1. Async/Await
2. Async/await adalah tambahan yang lebih baru dalam JavaScript. Ini menyederhanakan kode asinkron dengan memungkinkan Anda menulisnya dengan gaya yang lebih bersifat sinkron. Ini dibangun di atas Promises dan sangat berguna untuk menangani operasi asinkron dengan alur kode yang lebih linier.

4. Modularisasi

- Susun kode ke dalam modul yang lebih kecil dan dapat digunakan kembali. Ini mengurangi kompleksitas fungsi individu dan memudahkan pengelolaan operasi asinkron.

Kesimpulan

Penanganan kesalahan yang efektif sangat penting dalam pemrograman asinkron. Callback dapat menangani kesalahan sesuai konvensi, tetapi pendekatan alternatif seperti Promises, async/await, dan pengirim acara memberikan cara yang lebih terstruktur dan mudah dibaca untuk mengelola kode asinkron. Pilihan pendekatan mana yang digunakan tergantung pada persyaratan khusus dan preferensi gaya pemrograman. Callback hell adalah masalah umum dalam JavaScript saat bekerja dengan fungsi callback yang bersarang dalam fungsi lain untuk menangani operasi asinkron. Ini dapat menghasilkan kode yang sulit dibaca, dipelihara, dan didebug. Strategi mitigasi, seperti penggunaan fungsi beranama, Promises, async/await, atau modularisasi, dapat secara signifikan meningkatkan struktur dan keterbacaan kode saat menangani tugas asinkron, membuat kode Anda lebih mudah dipelihara dan tahan kesalahan.

API Web dan AJAX

Dalam bagian ini, kita akan membahas API dan AJAX, dua teknologi penting yang memungkinkan aplikasi berinteraksi dengan server dan mengirim atau mengambil data tanpa perlu me-reload halaman secara keseluruhan.

API (Application Programming Interface)

Sebuah **API** (Application Programming Interface) adalah seperangkat aturan dan protokol yang memungkinkan aplikasi perangkat lunak yang berbeda berkomunikasi satu sama lain. Ini mendefinisikan metode dan format data yang dapat digunakan oleh aplikasi untuk meminta dan bertukar informasi.

Poin-Poin Kunci

- API memungkinkan pengembang mengakses fungsionalitas komponen, layanan, atau platform perangkat lunak lain tanpa perlu memahami kerja internalnya.
- API memberikan cara bagi aplikasi untuk mengirim permintaan dan menerima respons, memungkinkan mereka berinteraksi dan berbagi data dengan mulus.
- Jenis-jenis umum dari API meliputi **API web** yang memungkinkan aplikasi web berkomunikasi melalui internet, **API pustaka** yang menyediakan fungsi kode yang dapat digunakan kembali, dan **API sistem operasi** yang memungkinkan interaksi dengan sistem operasi yang mendasarinya.
- API sangat penting untuk membuat integrasi, membangun perangkat lunak di atas platform yang ada, dan memungkinkan interoperabilitas antara sistem yang berbeda.
- Dokumentasi API, seringkali disediakan oleh pengembang atau penyedia layanan, menjelaskan cara menggunakan API, termasuk titik akhir yang tersedia, metode permintaan, dan format respons.
- Contoh-contoh API populer meliputi API media sosial (misalnya, Facebook Graph API), API gateway pembayaran (misalnya, PayPal API), dan API layanan cloud (misalnya, AWS API).

Manfaat dari API

- **Interoperabilitas:** API memungkinkan sistem perangkat lunak yang berbeda bekerja bersama, mempromosikan kompatibilitas dan pertukaran data.
- **Efisiensi:** Pengembang dapat memanfaatkan API yang ada untuk menghemat waktu dan usaha, fokus pada membangun fungsi khusus.
- **Skalabilitas:** API memungkinkan perluasan layanan dan fitur dengan mengintegrasikan alat dan layanan pihak ketiga.
- **Inovasi:** API mendorong inovasi dengan membuka peluang bagi pengembang untuk membuat aplikasi dan layanan baru.
- **Keamanan:** API seringkali mencakup mekanisme otentikasi dan otorisasi untuk memastikan akses yang aman ke data dan layanan.

AJAX (Asynchronous JavaScript and XML)

AJAX, singkatan dari **Asynchronous JavaScript and XML**, adalah teknologi dasar dalam pengembangan web. Ini memungkinkan aplikasi web untuk membuat permintaan asinkron ke server, mengambil data, dan memperbarui bagian dari halaman web tanpa memerlukan me-reload halaman sepenuhnya. Meskipun nama tersebut mencantumkan XML, AJAX dapat bekerja dengan berbagai format data, dengan JSON menjadi yang paling umum.

Apa Itu AJAX?

Pada intinya, AJAX adalah teknik yang memungkinkan halaman web berkomunikasi dengan server secara latar belakang, tanpa mengganggu interaksi pengguna dengan halaman. Perilaku asinkron ini dicapai menggunakan JavaScript dan memungkinkan pengembangan aplikasi web yang lebih interaktif dan responsif.

Bagaimana AJAX Bekerja?

1. **JavaScript**: AJAX sangat bergantung pada JavaScript untuk memulai permintaan dan menangani respons secara asinkron.
2. **Objek XMLHttpRequest (XHR)**: Meskipun secara historis objek `XMLHttpRequest` digunakan, pengembangan web modern sering menggunakan API `fetch` untuk permintaan AJAX, yang memberikan pendekatan yang lebih intuitif dan fleksibel.
3. **Komunikasi dengan Server**: Ketika pengguna memicu suatu peristiwa, seperti mengklik tombol, JavaScript mengirim permintaan HTTP ke server. Permintaan ini bisa berupa GET (untuk mengambil data) atau POST (untuk mengirim data ke server).
4. **Pemrosesan Asinkron**: Permintaan AJAX bersifat asinkron, yang berarti bahwa browser dapat terus menjalankan kode lain saat menunggu respons. Ini mencegah antarmuka pengguna membeku.
5. **Penanganan Respons**: Begitu server memproses permintaan, ia mengirim respons kembali ke klien. JavaScript kemudian menangani respons ini, biasanya dengan memperbarui Model Objek Dokumen (DOM) dengan data baru.
6. **Penguraian**: Konten yang diperbarui di-render pada halaman web tanpa memerlukan me-reload halaman sepenuhnya, menghasilkan pengalaman pengguna yang lebih mulus.

Manfaat dari AJAX

- **Pengalaman Pengguna yang Ditingkatkan**: AJAX memungkinkan aplikasi web untuk mengambil dan menampilkan data tanpa perlu me-reload halaman sepenuhnya, membuat pengalaman pengguna lebih mulus dan interaktif.
- **Efisiensi**: Permintaan AJAX ringan dan hanya mentransfer data yang diperlukan, mengurangi penggunaan bandwidth dan meningkatkan kinerja aplikasi.
- **Pembaruan Real-time**: AJAX sangat penting untuk membangun fitur real-time seperti aplikasi obrolan, pemberitahuan langsung, dan pembaruan konten dinamis.
- **Pemuatan Dinamis**: Konten dapat dimuat berdasarkan permintaan, menghasilkan waktu pemuatan halaman awal yang lebih cepat dan aplikasi yang lebih responsif.

Kasus Penggunaan Umum

Beberapa skenario umum di mana AJAX digunakan meliputi:

- **Pengiriman Formulir**: Mengirim formulir tanpa me-reload halaman sepenuhnya untuk validasi dan pengiriman data.
- **Guliran Tanpa Batas**: Memuat konten tambahan saat pengguna menggulir ke bawah halaman, memberikan pengal

aman penjelajahan yang berkelanjutan.

- **Saran Otomatis**: Memberikan saran pencarian real-time saat pengguna mengetikkan pertanyaan pencarian.
- **Pembaruan Konten**: Memperbarui konten secara dinamis seperti umpan berita, informasi cuaca, atau skor olahraga tanpa me-reload halaman secara manual.

Mengambil Data Cuaca dengan API OpenWeatherMap menggunakan AJAX

Dalam contoh ini, kami akan menunjukkan cara menggunakan AJAX (Asynchronous JavaScript and XML) untuk mengambil informasi cuaca dari API OpenWeatherMap dan menampilkannya di halaman web.

Pengantar

API OpenWeatherMap adalah alat yang kuat untuk mengambil informasi cuaca untuk lokasi di seluruh dunia. Contoh ini menunjukkan cara menggunakan API untuk mengambil data cuaca saat ini untuk sebuah kota tertentu dan menampilkannya dalam aplikasi atau dokumentasi Anda.

Kunci API

Sebelum memulai, Anda perlu mendaftar untuk mendapatkan kunci API dari OpenWeatherMap untuk mengakses API data cuaca mereka. Gantilah `'YOUR_API_KEY'` dalam kode di bawah dengan kunci API Anda yang sebenarnya.

```
const apiKey = "YOUR_API_KEY";
```

Aplikasi Cuaca Sederhana HTML

Dalam contoh ini, kami akan memberikan struktur HTML untuk aplikasi cuaca sederhana yang mengambil dan menampilkan data cuaca dari API OpenWeatherMap.

Struktur HTML

```html
<!DOCTYPE html>
<html lang="id">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Aplikasi Cuaca</title>
  </head>
  <body>
    <h1>Laporan Cuaca</h1>
    <button id="tombolAmbil">Ambil Data</button>
    <div id="info-cuaca">
      <!-- Data akan ditampilkan di sini -->
    </div>
    <script src="script.js"></script>
  </body>
</html>
```

JavaScript (script.js)

Buat file JavaScript dengan nama `script.js` untuk menangani permintaan AJAX dan memperbarui data cuaca di halaman:

```javascript
// Titik Akhir API dan Lokasi
const apiUrl = "https://api.openweathermap.org/data/2.5/weather";
const lokasi = "New York"; // Gantilah dengan lokasi yang Anda inginkan

// Ambil Data Cuaca
const xhr = new XMLHttpRequest();
xhr.open("GET", `${apiUrl}?q=${lokasi}&appid=${apiKey}`, true);

xhr.onload = function () {
  if (xhr.status === 200) {
    const infoCuaca = document.getElementById("info-cuaca");
    const data = JSON.parse(xhr.responseText);
    const suhu = (data.main.temp - 273.15).toFixed(2); // Konversi dari Kelvin ke Celsius

    const html = `
      <p><strong>Lokasi:</strong> ${data.name}, ${data.sys.country}</p>
      <p><strong>Suhu:</strong> ${suhu} °C</p>
      <p><strong>Cuaca:</strong> ${data.weather[0].description}</p>
    `;

    infoCuaca.innerHTML = html;
  } else {
    const infoCuaca = document.getElementById("info-cuaca");
    infoCuaca.innerHTML = "<p>Gagal mengambil data cuaca.</p>";
  }
};

xhr.send();
```

Hasil

Ketika Anda membuka file HTML ini di peramban web, itu akan menampilkan informasi cuaca untuk lokasi yang ditentukan (New York dalam kasus ini). Pastikan untuk menggantikan `'YOUR_API_KEY'` dengan kunci API OpenWeatherMap Anda yang sebenarnya.

Contoh ini menunjukkan bagaimana mengambil data cuaca dari API OpenWeatherMap menggunakan AJAX dan menampilkannya di halaman web sederhana.

Jangan lupa untuk meng-host file HTML dan JavaScript Anda di server web jika Anda berencana mengakses API dari situs web langsung.

Itu dia! Anda telah berhasil mengambil dan menampilkan data cuaca menggunakan API OpenWeatherMap dan AJAX.

Kesimpulan

API memainkan peran penting dalam pengembangan perangkat lunak modern dengan memungkinkan aplikasi berkolaborasi dan berbagi data dengan efektif. Memahami cara menggunakan API dan mengintegrasikannya ke dalam proyek Anda adalah hal mendasar untuk membangun perangkat lunak yang kaya fitur dan terhubung.

AJAX adalah teknologi dasar dalam pengembangan web modern yang memberdayakan pengembang untuk membuat aplikasi web yang dinamis dan responsif. Meskipun namanya mencantumkan XML, AJAX kompatibel dengan berbagai format data, menjadikannya alat serbaguna untuk meningkatkan pengalaman pengguna dan membuat aplikasi web interaktif.

Sifat Tunggal JavaScript

JavaScript adalah bahasa pemrograman tunggal, menjalankan kode secara berurutan dalam satu utas utama. Ia mengandalkan pola asinkron non-blokir untuk menangani tugas-tugas dengan efisien tanpa memblokir utas utama, memastikan responsif dalam aplikasi web. Sementara menyederhanakan konkurensi, ini memerlukan penggunaan yang efektif dari callback dan pemrograman berbasis peristiwa.

Memahami JavaScript yang Bersifat Tunggal

Berikut adalah beberapa poin kunci untuk memahami eksekusi yang bersifat tunggal dalam JavaScript:

1. **Satu Utas, Satu Tugas:** JavaScript beroperasi dalam satu utas eksekusi tunggal, yang berarti ia hanya dapat melakukan satu tugas pada satu waktu. Utas ini sering disebut sebagai "utas utama" atau "loop peristiwa."
2. **Blokir vs. Non-Blokir:** Kode JavaScript secara alami bersifat non-blokir. Ini berarti ketika operasi yang memakan waktu (seperti permintaan jaringan atau membaca file) dihadapi, JavaScript tidak menunggu hingga selesai. Sebaliknya, ia menugaskan tugas tersebut ke bagian lain dari lingkungan (misalnya, peramban atau runtime Node.js) dan melanjutkan eksekusi kode lainnya.
3. **Pemrograman Asinkron:** Untuk menangani operasi yang memakan waktu tanpa memblokir utas utama, JavaScript sangat mengandalkan pola-pola pemrograman asinkron. Fungsi-fungsi seperti callback, Promises, dan async/await memungkinkan pengembang untuk bekerja dengan operasi asinkron dengan efektif.
4. **Berbasis Peristiwa:** JavaScript sering dijelaskan sebagai "berbasis peristiwa." Ini berarti ia mendengarkan dan merespons peristiwa, seperti interaksi pengguna (klik, penekanan tombol), timer, atau respon jaringan. Ketika peristiwa terjadi, fungsi callback yang sesuai dieksekusi.
5. **Model Konkurensi:** Meskipun JavaScript berjalan dalam satu utas, model konkurensi memungkinkan eksekusi kode bersamaan. Hal ini dicapai melalui mekanisme seperti loop peristiwa, yang mengelola eksekusi tugas-tugas asinkron dengan cara yang memastikan responsivitas dan perilaku non-blokir.
6. **Interaksi dengan Peramban dan Lingkungan:** Dalam pengembangan web, JavaScript berinteraksi dengan Model Objek Dokumen (DOM) peramban dan API peramban lainnya. Untuk menjaga antarmuka pengguna yang responsif, kode JavaScript harus dieksekusi dengan cepat dan efisien serta menugaskan operasi yang memakan waktu ke utas terpisah jika diperlukan.

Contoh Asinkron Tunggal JavaScript

```
// Simulasi operasi asinkron dengan callback
function simulasiOperasiAsinkron(callback) {
  setTimeout(function () {
    console.log("Operasi asinkron selesai.");
    callback();
  }, 2000); // Mensimulasikan penundaan selama 2 detik
}

console.log("Awal program");

// Memulai operasi asinkron
simulasiOperasiAsinkron(function () {
  console.log("Callback dieksekusi: Menangani hasil.");
});

consolelog("Akhir program");
```

Dalam contoh ini, kami menunjukkan sifat tunggal JavaScript dan bagaimana ia menangani operasi asinkron menggunakan callback.

Penjelasan Kode:

- Kami mendefinisikan fungsi `simulasiOperasiAsinkron` yang mensimulasikan operasi asinkron menggunakan `setTimeout`. Fungsi ini mengambil callback sebagai argumen, yang akan dieksekusi ketika operasi asinkron selesai.
- Kami memulai program dengan mencatat "Awal program."
- Kami memulai operasi asinkron menggunakan `simulasiOperasiAsinkron`, dengan menyertakan fungsi callback. Fungsi ini akan dieksekusi setelah penundaan selama 2 detik.
- Segera setelah memulai operasi asinkron, kami mencatat "Akhir program."

Alur Eksekusi:

- Ketika Anda menjalankan kode ini, Anda akan melihat bahwa meskipun operasi asinkron memakan waktu 2 detik untuk selesai, program tidak memblokir. Pesan "Akhir program" dicatat segera setelah memulai operasi asinkron, menunjukkan perilaku non-blokir JavaScript.
- Setelah penundaan 2 detik, pesan "Operasi asinkron selesai." dicatat, diikuti oleh "Callback dieksekusi: Menangani hasil," yang menunjukkan bahwa fungsi callback dieksekusi ketika operasi asinkron selesai.

Informasi Penting:

- JavaScript beroperasi dalam satu utas, dan operasi asinkron ditangani melalui callback.
- Sifat tunggal memungkinkan JavaScript tetap responsif bahkan selama tugas yang memakan waktu.
- Callback adalah mekanisme fundamental untuk bekerja dengan kode asinkron dalam JavaScript.

Manfaat dan Tantangan

Manfaat:

- Kesederhanaan: Eksekusi tunggal menyederhanakan model pemrograman dan mengurangi risiko bug yang terkait dengan konkurensi yang kompleks.
- Prediktabilitas: Sifat tunggal memudahkan untuk memahami urutan eksekusi dan status program Anda.

Tantangan:

- Operasi yang Memblock: Operasi yang berjalan lama dapat memblokir utas utama, menyebabkan pengalaman pengguna yang buruk, terutama dalam aplikasi web.
- Callback Hell: Penggunaan berlebihan callback (sering disebut sebagai "callback hell") dapat membuat kode sulit dibaca dan dipelihara.
- Kendala Konkurensi: Tugas yang mengikat CPU tidak dapat sepenuhnya memanfaatkan prosesor multi-inti karena JavaScript berjalan dalam satu utas.

Secara ringkas, sifat tunggal JavaScript adalah fitur

yang menentukan dari bahasa ini. Sementara itu menyederhanakan beberapa aspek pemrograman, juga menimbulkan tantangan dalam menangani tugas asinkron dan memastikan aplikasi responsif. Penggunaan yang efektif dari pola asinkron dan pemahaman model berbasis peristiwa adalah penting bagi pengembang JavaScript.

ECMAScript (ES)

ECMAScript, biasanya disingkat sebagai ES, adalah spesifikasi bahasa scripting yang telah distandardisasi. Ini berfungsi sebagai dasar bagi beberapa bahasa pemrograman, dengan JavaScript menjadi implementasi yang paling terkenal dan luas digunakan. Dokumen Markdown ini memberikan gambaran tentang ECMAScript, sejarahnya, fitur-fiturnya, dan peranannya dalam pengembangan web.

Apa Itu ECMAScript?

- **Bahasa yang Distandardisasi**: ECMAScript adalah spesifikasi bahasa scripting yang telah distandardisasi oleh ECMA International. Ini mendefinisikan sintaksis dan semantik bahasa untuk memastikan konsistensi dan interoperabilitas.
- **Implementasi JavaScript**: JavaScript adalah implementasi paling terkenal dari ECMAScript, tetapi bahasa lain seperti ActionScript juga menggunakan spesifikasi ini sebagai dasar.

Sejarah ECMAScript

- **ES1 (ECMAScript 1)**: Dirilis pada tahun 1997, ES1 membentuk dasar JavaScript seperti yang kita kenal saat ini.
- **ES3 (ECMAScript 3)**: Dirilis pada tahun 1999, ES3 memperkenalkan perbaikan signifikan dan dianggap sebagai versi yang membawa JavaScript ke pengembangan web mainstream.
- **ES5 (ECMAScript 5)**: Dirilis pada tahun 2009, ES5 menambahkan fitur-fitur baru dan meningkatkan yang sudah ada, membuat JavaScript lebih kuat.
- **ES6 (ECMAScript 2015)**: Dirilis pada tahun 2015, ES6 adalah tonggak besar, memperkenalkan perubahan signifikan seperti fungsi panah, kelas, modul, dan lainnya.
- **ESNext**: Merujuk pada perkembangan terus-menerus ECMAScript, di mana fitur-fitur dan perbaikan baru terus diusulkan dan ditambahkan.

Mengapa ECMAScript (ES) Distandardisasi untuk JavaScript

Bagian dokumen ini menjelaskan mengapa ECMAScript sangat penting untuk JavaScript, peranannya dalam standardisasi, dan manfaatnya bagi bahasa.

Kebutuhan Standarisasi

- **Konsistensi Bahasa**: JavaScript, sebagai bahasa pemrograman yang banyak digunakan dalam pengembangan web, membutuhkan spesifikasi yang distandardisasi untuk memastikan konsistensi di berbagai implementasi dan lingkungan.
- **Interoperabilitas**: Berbagai browser web dan mesin mungkin memiliki interpretasi mereka sendiri tentang JavaScript. Standar membantu memastikan bahwa kode JavaScript berperilaku konsisten di semua platform.

Peran ECMAScript

- **Mendefinisikan Bahasa**: ECMAScript mendefinisikan fitur-fitur inti JavaScript, termasuk sintaksisnya, tipe data, fungsi, dan objek dasar.
- **Badan Standarisasi**: ECMAScript dipelihara dan dikembangkan oleh ECMA International (European Computer Manufacturers Association), sebuah organisasi standar. Organisasi ini memastikan bahwa JavaScript tetap menjadi bahasa yang terdefinisi dengan baik dan stabil.
- **Evolusi Versi**: ECMAScript memperkenalkan fitur-fitur bahasa baru dan perbaikan di setiap versi baru, menjaga agar JavaScript selalu sesuai dengan kebutuhan pengembangan web modern.

Manfaat Standarisasi ECMAScript

- **Konsistensi**: Standarisasi memastikan bahwa JavaScript berperilaku konsisten di berbagai platform dan browser, mengurangi masalah kompatibilitas.
- **Interoperabilitas**: Pengembang dapat menulis kode JavaScript dengan keyakinan, tahu bahwa itu akan berfungsi seperti yang diharapkan di berbagai lingkungan.
- **Inovasi**: Pengembangan terus-menerus ECMAScript memungkinkan pengenalan fitur-fitur bahasa baru dan perbaikan, memungkinkan JavaScript berkembang seiring dengan lanskap web yang terus berubah.
- **Pengembangan lintas platform**: Standarisasi memudahkan pengembang untuk menulis kode yang bekerja di lingkungan sisi klien dan sisi server.

Implementasi JavaScript

- **Browser Utama**: Browser web populer seperti Chrome, Firefox, Safari, dan Edge semua menerapkan ECMAScript untuk mengeksekusi kode JavaScript.
- **Node.js**: Node.js, runtime JavaScript sisi server, juga mengikuti standar ECMAScript, memungkinkan JavaScript digunakan untuk pemrograman sisi server.

Fitur Utama ECMAScript

- **Fungsi Panah**: Memberikan sintaksis ringkas untuk mendefinisikan fungsi dan ikatan leksikal `this`.
- **Kelas**: Memperkenalkan sintaksis kelas untuk pemrograman berorientasi objek.
- **Modul**: Menambahkan dukungan asli untuk impor dan ekspor modul.
- **Janji**: Memperkenalkan Janji untuk penanganan operasi asinkron yang lebih baik.
- **Async/Await**: Mempermudah kode asinkron dengan pengenalan fungsi async.
- **let dan const**: Variabel berbatas blok dengan `let` dan konstan dengan `const`.
- **Dekonstruksi**: Memungkinkan ekstraksi nilai dari array dan objek dengan mudah.
- **Literal Template**: Memperkenalkan literal template untuk interpolasi string yang lebih fleksibel.

Peran ECMAScript dalam Pengembangan Web

- **Pemrograman di Sisi Klien**: ECMAScript adalah dasar untuk pemrograman di sisi klien dalam pengembangan web. Ini menggerakkan aplikasi web interaktif.
- **Kompatibilitas**: Meskipun browser modern mendukung fitur-fitur ECMAScript terbaru, pengembang perlu mempertimbangkan kompatibilitas ke belakang untuk browser yang lebih lama.
- **Transpiler**: Alat seperti Babel dapat mengonversi kode ECMAScript terbaru menjadi versi yang lebih lama untuk mendukung browser yang lebih luas.
- **TypeScript**: TypeScript, sebagai superset dari ECMAScript, menambahkan tipe statis untuk alat dan keamanan kode yang lebih baik.

Kesimpulan

ECMAScript adalah bagian fundamental dari pengembangan web, membentuk cara kita membuat aplikasi web dinamis dan interaktif. Tetap terinformasi tentang fitur-fitur ECMAScript terbaru adalah penting untuk pengembangan

JavaScript modern. ECMAScript memainkan peran penting dalam menyediakan dasar yang distandardisasi untuk JavaScript, memastikan konsistensi, interoperabilitas, dan perbaikan yang berkelanjutan dari bahasa. Standarisasi ini memungkinkan pengembang menulis kode JavaScript dengan keyakinan, tahu bahwa itu akan bekerja dengan andal di berbagai platform dan lingkungan yang berbeda.

Pengujian Unit

Pengujian unit adalah praktik dasar dalam pengembangan web. Ini melibatkan pengujian komponen atau fungsi individu untuk memastikan bahwa mereka berfungsi seperti yang diharapkan. Praktik ini dapat mendeteksi bug lebih awal, meningkatkan kualitas kode, dan membuat refaktoring lebih aman. Pengujian unit penting dengan alasan berikut:

- Ini memverifikasi bahwa bagian-bagian individu dalam kode Anda berfungsi dengan benar.
- Ini menyediakan jaringan keamanan saat melakukan refaktoring atau membuat perubahan.
- Ini membantu mendokumentasikan perilaku yang diharapkan dari fungsi dan komponen.

Kerangka Kerja Pengujian

Kerangka kerja pengujian menyederhanakan proses penulisan dan pelaksanaan pengujian. Dua kerangka kerja yang populer adalah Jest dan Mocha.

Jest

Jest adalah kerangka kerja pengujian populer yang tidak memerlukan konfigurasi awal. Cocok untuk pengujian unit dan integrasi. Mari kita lihat cara memulai dengan Jest.

Instalasi Jest menggunakan npm atau yarn:

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

Buat file pengujian (misalnya, `myFunction.test.js`) untuk fungsi yang ingin Anda uji.

Tulis kasus pengujian menggunakan fungsi pengujian Jest:

```
const myFunction = require("./myFunction");

test("seharusnya mengembalikan hasil penjumlahan dua angka", () => {
  expect(myFunction(2, 3)).toBe(5);
});
```

Jalankan pengujian menggunakan perintah jest:

```
npx jest
```

Mocha

Mocha adalah kerangka kerja pengujian yang fleksibel. Ini menyediakan struktur untuk menjalankan pengujian tetapi memerlukan perpustakaan tambahan untuk asersi dan pembuatan tiruan (mocking).

Memulai dengan Mocha

Instalasi Mocha dan perpustakaan asersi seperti Chai:

```
npm install --save-dev mocha chai
```

Buat direktori `test` dan tambahkan file pengujian Anda.

Tulis kasus pengujian menggunakan fungsi describe dan it dari Mocha dan fungsi asersi Chai.

```
const chai = require("chai");
const expect = chai.expect;
const myFunction = require("./myFunction");

describe("myFunction", () => {
  it("seharusnya mengembalikan hasil penjumlahan dua angka", () => {
    expect(myFunction(2, 3)).to.equal(5);
  });
});
```

Kesimpulan

Dalam bab ini, kita telah menjelajahi dasar-dasar pengujian dalam pengembangan web dan membahas pentingnya pengujian unit serta kerangka kerja dan alat pengujian lain yang sangat penting bagi pengembang web mana pun. Dengan praktik yang konsisten dan akses ke seperangkat alat yang tepat, seseorang dapat menulis kode yang dapat diandalkan dan memastikan bahwa aplikasi berjalan dengan optimal.

Bab 20

Kode Sisi Server

Kode sisi server merujuk pada kode yang berjalan pada *web server* daripada pada peramban web pengguna. Kode ini bertanggung jawab untuk memproses permintaan dari klien (biasanya peramban web) dan menghasilkan halaman web dinamis atau menyediakan data kepada klien.

Kode sisi klien merujuk pada kode yang berjalan di *peramban web* pengguna daripada di server web. Kode ini bertanggung jawab untuk menghasilkan antarmuka pengguna dan menangani interaksi pengguna. Kode sisi klien biasanya ditulis dalam bahasa JavaScript dan dieksekusi oleh peramban.

Mengapa kita memerlukan kode sisi server?

Kode sisi server sangat penting dalam pengembangan web karena beberapa alasan:

- **Keamanan**: Kode sisi server tidak terlihat oleh pengguna, sehingga lebih aman daripada kode sisi klien.
- **Kinerja**: Kode sisi server dapat digunakan untuk melakukan tugas yang membutuhkan komputasi, seperti pemrosesan data, tanpa memengaruhi pengalaman pengguna.
- **Penyimpanan Data**: Kode sisi server dapat digunakan untuk menyimpan data dalam database, yang kemudian dapat diakses oleh kode sisi klien.
- **Autentikasi Pengguna**: Kode sisi server dapat digunakan untuk mengautentikasi pengguna dan membatasi akses ke bagian-bagian tertentu dari situs web.
- **Konten Dinamis**: Kode sisi server dapat digunakan untuk menghasilkan halaman web dinamis, yang dapat disesuaikan untuk setiap pengguna.

Kode Sisi Server vs. Kode Sisi Klien

Perbedaan-perbedaan ini dirangkum dalam tabel di bawah:

Kode Sisi Server	Kode Sisi Klien
Berjalan pada server web	Berjalan di peramban web
Memiliki akses ke sumber daya server (sistem file, database, dll.).	Memiliki akses ke sumber daya klien (cookie, penyimpanan lokal, dll.).
Dapat ditulis dalam berbagai bahasa (PHP, Python, Ruby, Java, C#, dll.).	Hanya dapat ditulis dalam JavaScript.
Mungkin menggunakan rendering sisi server (SSR) untuk menghasilkan HTML di server.	Menggunakan rendering sisi klien (CSR) untuk menghasilkan HTML di peramban.
Lebih baik untuk SEO karena kontennya langsung tersedia untuk mesin pencari.	Lebih buruk untuk SEO karena kontennya tidak langsung tersedia untuk mesin pencari.
Dapat memanfaatkan pengecualian dan Jaringan Pengiriman Konten (CDN) untuk kinerja.	Kontrol terbatas atas pengecualian, mengandalkan cache peramban.

Mengapa menggunakan JavaScript untuk kode sisi server?

Berbeda dengan kode sisi klien, yang hanya dapat ditulis dalam JavaScript, kode sisi server dapat ditulis dalam berbagai bahasa, termasuk PHP, Python, Ruby, Java, C#, dan banyak lagi. Jadi mengapa menggunakan JavaScript untuk kode sisi server? Ada beberapa alasan:

- **Bahasa Tunggal**: Pengembang dapat menggunakan bahasa yang sama dan paradigma pemrograman sepanjang tumpukan aplikasi, yang dapat mengarah pada reuse kode dan kolaborasi yang lebih mudah antara pengembang sisi depan dan sisi belakang.
- **Ekosistem Besar**: JavaScript memiliki ekosistem yang luas dari perpustakaan dan paket yang tersedia melalui npm (Node Package Manager). Ekosistem yang kaya ini menyederhanakan proses pengembangan dengan menyediakan modul yang telah dibangun sebelumnya untuk berbagai fungsionalitas, mulai dari routing server hingga konektivitas basis data.
- **JSON**: JavaScript Object Notation (JSON) adalah format data populer yang digunakan untuk mengirimkan data antara server dan aplikasi web. JSON didasarkan pada JavaScript, sehingga mudah untuk bekerja dengan data JSON dalam JavaScript.

Selanjutnya, kita akan belajar cara menggunakan JavaScript untuk kode sisi server dengan Node.js dan cara menggunakan Server Side Rendering (SSR) untuk menghasilkan HTML di server.

Node.js

Node.js adalah lingkungan runtime JavaScript yang memungkinkan pengembang menjalankan kode JavaScript di luar peramban web. Ini dibangun di atas mesin JavaScript V8, yang merupakan mesin yang sama yang digunakan oleh Google Chrome. Node.js adalah sumber terbuka dan lintas platform, yang berarti itu dapat berjalan di Windows, macOS, dan Linux.

Node.js bukan bahasa pemrograman

Banyak orang keliru percaya bahwa Node.js adalah bahasa pemrograman. Ini tidak benar. Node.js adalah lingkungan runtime JavaScript, yang berarti ia menyediakan lingkungan untuk menjalankan kode JavaScript. Ini bukan bahasa pemrograman itu sendiri.

Node.js **bukan** satu-satunya lingkungan runtime JavaScript. Ada banyak lainnya, termasuk Deno, Nashorn, dan yang paling baru, Bun. Tetapi Node.js adalah jauh yang paling populer dan paling banyak digunakan lingkungan runtime JavaScript.

Memulai dengan Node.js

Untuk memulai dengan Node.js, Anda perlu menginstalnya di komputer Anda. Anda dapat mengunduh versi terbaru Node.js dari situs web resmi di nodejs.org. Setelah Anda mengunduh dan menginstal Node.js, Anda dapat memverifikasi instalasi dengan menjalankan perintah berikut di terminal Anda:

```
node --version
```

Ini seharusnya mencetak nomor versi Node.js seperti ini:

```
v20.7.0
```

Menulis program Node.js pertama Anda

Sekarang setelah Anda menginstal Node.js, mari tulis program Node.js pertama kita. Buat file baru bernama `hello.js` dan tambahkan kode berikut:

```
console.log("Halo Dunia!");
```

Untuk menjalankan program ini, buka terminal Anda dan navigasikan ke direktori di mana Anda menyimpan file `hello.js`. Kemudian jalankan perintah berikut:

```
node hello.js
```

Ini seharusnya mencetak keluaran berikut:

```
Halo Dunia!
```

Menulis server web sederhana menggunakan Express dan Node.js

Express adalah kerangka web populer untuk Node.js. Ini menyediakan API yang sederhana dan elegan untuk membangun aplikasi web. Mari gunakan Express untuk membuat server web sederhana yang akan merespons permintaan HTTP dengan pesan "Halo Dunia!".

Pertama, kita perlu menginstal Express. Untuk melakukannya, jalankan perintah berikut di terminal Anda:

```
npm install express
```

Ini akan menginstal Express dan semua dependensinya. Setelah instalasi selesai, buat file baru bernama `server.js` dan tambahkan kode berikut:

```
const express = require("express");
const app = express();

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

app.listen(3000, () => {
  console.log("Server mendengarkan di port 3000");
});
```

Kode ini membuat aplikasi Express baru dan menentukan rute untuk jalur akar (`/`). Ketika permintaan dibuat ke rute ini, server akan merespons dengan pesan "Halo Dunia!".

Untuk menjalankan program ini, buka terminal Anda dan navigasikan ke direktori di mana Anda menyimpan file `server.js` . Kemudian jalankan perintah berikut:

```
node server.js
```

Ini seharusnya mencetak keluaran berikut:

```
Server mendengarkan di port 3000
```

Sekarang buka peramban web Anda dan buka http://localhost:3000. Anda seharusnya melihat pesan "Halo Dunia!".

Server Side Rendering (SSR)

Secara normal, ketika seorang pengguna mengunjungi sebuah situs web, peramban mengirim permintaan ke server, yang memberikan respons dalam bentuk HTML, CSS, dan JavaScript. Namun, dengan perpustakaan seperti *React* dan *Vue*, server hanya mengirimkan halaman HTML kosong bersama dengan file JavaScript. File JavaScript kemudian merender halaman di peramban. Ini disebut **Client Side Rendering (CSR)**.

Server Side Rendering (SSR) adalah sebuah teknik di mana server memproses permintaan dan menghasilkan HTML di server dari komponen React atau Vue. Kemudian, server mengirimkan HTML yang dihasilkan ke peramban, yang kemudian dapat merender halaman tanpa harus menunggu JavaScript untuk dimuat.

Mengapa menggunakan SSR?

Ada beberapa keuntungan menggunakan SSR dibandingkan dengan CSR:

- **Lebih baik untuk SEO**: Mesin pencari dapat menjelajahi dan mengindeks konten situs web Anda dengan lebih mudah jika kontennya dirender di server. Ini dapat menghasilkan peringkat mesin pencari yang lebih baik dan lebih banyak lalu lintas dari mesin pencari.
- **Waktu muat halaman awal yang lebih cepat**: Karena HTML dihasilkan di server, peramban tidak perlu menunggu JavaScript dimuat sebelum merender halaman. Ini dapat menghasilkan waktu muat halaman awal yang lebih cepat.
- **Kinerja yang lebih baik pada perangkat berdaya rendah**: Karena HTML dihasilkan di server, peramban tidak perlu melakukan banyak pekerjaan untuk merender halaman. Ini dapat menghasilkan kinerja yang lebih baik pada perangkat berdaya rendah, seperti ponsel dan tablet.

Kerugian SSR

Ada juga beberapa kerugian dalam menggunakan SSR:

- **Proses pengembangan yang lebih kompleks**: SSR membutuhkan lebih banyak pekerjaan di sisi server, yang dapat membuat proses pengembangan menjadi lebih kompleks.
- **Sumber daya server yang lebih banyak**: SSR membutuhkan lebih banyak sumber daya server, yang dapat mengakibatkan biaya hosting yang lebih tinggi.
- **Fungsionalitas sisi klien yang terbatas**: SSR tidak memungkinkan Anda menggunakan perpustakaan sisi klien, seperti jQuery atau Bootstrap, karena perpustakaan tersebut tidak tersedia di server.

Cara mengimplementasikan SSR?

Setiap perpustakaan memiliki cara tersendiri untuk mengimplementasikan SSR. Misalnya, untuk React, Anda dapat menggunakan Next.js atau Gatsby. Untuk Vue, Anda dapat menggunakan Nuxt.js. Untuk Svelte, Anda dapat menggunakan SvelteKit.

Kesimpulan

Dalam bab ini, kita mempelajari tentang Server Side Rendering (SSR) dan bagaimana itu dapat meningkatkan kinerja situs web Anda. Kita juga mempelajari tentang kelebihan menggunakan SSR dibandingkan dengan CSR dan bagaimana mengimplementasikan SSR dengan React, Vue, dan Svelte.

Bab 21

Latihan

Pada bab ini, kita akan melakukan latihan untuk menguji pengetahuan kita dalam JavaScript. Latihan yang akan kita lakukan tercantum di bawah ini:

- Konsol
- Perkalian
- Variabel Input Pengguna
- Konstanta
- Konkatenasi
- Fungsi
- Pernyataan Kondisional
- Objek
- Masalah FizzBuzz
- Dapatkan Judulnya!

Konsol

Dalam JavaScript, kita menggunakan `console.log()` untuk menulis pesan (konten dari variabel, string tertentu, dll.) di `konsol`. Ini digunakan terutama untuk tujuan debugging, idealnya untuk meninggalkan jejak konten variabel selama eksekusi program.

Contoh:

```
console.log("Selamat datang di Learn JavaScript Beginners Edition");
let usia = 30;
console.log(usia);
```

📝 Tugas:

- [ ] Tulis program untuk mencetak `Hello World` di konsol. Jangan ragu untuk mencoba hal lain juga!
- [ ] Tulis program untuk mencetak variabel ke `konsol`.
    1. Deklarasikan variabel `hewan` dan berikan nilainya berupa "naga".
    2. Cetak variabel `hewan` ke `konsol`.

💡 Petunjuk:

- Kunjungi bab variabel untuk memahami lebih lanjut tentang variabel.

Perkalian

Dalam JavaScript, kita dapat melakukan perkalian dua angka dengan menggunakan operator aritmatika asterisk `(*)`.

Contoh:

```
let nilaiHasil = 3 * 2;
```

Di sini, kita menyimpan hasil dari `3 * 2` ke dalam variabel `nilaiHasil`.

📝 Tugas:

- [ ] Tulis program untuk menyimpan hasil dari `23` kali `41` dan mencetak nilainya.
- [ ] Tulis program yang menghasilkan tabel perkalian untuk suatu angka tertentu. Program ini harus mengambil angka sebagai input dan kemudian menampilkan tabel perkalian untuk angka tersebut, dari 1 hingga 10.

💡 Petunjuk:

- Kunjungi bab Operator Dasar untuk memahami operasi matematika.

Variabel Input Pengguna

Di JavaScript, kita dapat mengambil input dari pengguna dan menggunakannya sebagai variabel. Kita tidak perlu tahu nilai mereka untuk bekerja dengan mereka.

📝 Tugas:

- [ ] Tulis program untuk mengambil input dari pengguna dan menambahkan `10` kepadanya, dan cetak hasilnya.

💡 Petunjuk:

- Isi dari variabel ditentukan oleh masukan pengguna. Metode `prompt()` menyimpan nilai masukan sebagai string.
- Anda perlu memastikan bahwa nilai string dikonversi menjadi bilangan bulat untuk perhitungan.
- Kunjungi bab Operator Dasar untuk konversi tipe `string` menjadi `int`.

Konstan

Konstan diperkenalkan dalam ES6 (2015), dan menggunakan kata kunci `const`. Variabel yang dideklarasikan dengan `const` tidak dapat diubah nilainya atau dideklarasikan kembali.

Contoh:

```
const VERSI = "1.2";
```

📝 Tugas:

- [ ] Jalankan program berikut dan perbaiki kesalahan yang Anda lihat di konsol. Pastikan hasil kode adalah `0.9` setelah diperbaiki di konsol.

  ```
  const VERSI = "0.7";
  VERSI = "0.9";
  console.log(VERSI);
  ```

- [ ] Tulis program yang meminta pengguna memasukkan suhu dalam *derajat Celsius*, lalu gunakan konstanta `FAKTOR_KONVERSI` yang sama dengan `9/5` untuk mengonversi ke *derajat Fahrenheit*.

  ```
  const FAKTOR_KONVERSI = 9 / 5;

  /* Mulai kode Anda dari sini*/
  ```

💡 Petunjuk:

- Kunjungi bab Variabel untuk informasi lebih lanjut tentang `const`, dan cari "*TypeError assignment to constant variable*" di mesin pencari untuk mempelajari cara memperbaikinya.

Penggabungan

Dalam bahasa pemrograman apa pun, penggabungan string hanya berarti menambahkan satu atau lebih string ke string lain. Misalnya, ketika string "*World*" dan "*Good Afternoon*" digabungkan dengan string "*Hello*", mereka membentuk string "*Hello World, Good Afternoon*". Kita dapat menggabungkan string dengan beberapa cara di JavaScript.

Contoh:

```
const icon = "👋";

// menggunakan Template Strings
`hi ${icon}`;

// menggunakan Metode join()
["hi", icon].join(" ");

// menggunakan Metode concat()
"".concat("hi ", icon);

// menggunakan operator +
"hi " + icon;

// HASIL
// hi 👋
```

📝 Tugas:

- [ ] Tulis program untuk mengatur nilai `str1` dan `str2` sehingga kode mencetak '*Hello World*' ke konsol.
- [ ] Tulis program yang meminta pengguna untuk memasukkan nama depan mereka (`first_name`) dan nama belakang (`last_name`). Kemudian, gunakan penggabungan string untuk membuat dan menampilkan nama lengkap mereka (`full_name`).
- [ ] Tulis program yang meminta pengguna untuk memasukkan nama mereka. Kemudian, gunakan penggabungan string untuk membuat pesan sambutan yang mencakup nama mereka. Contohnya: `Selamat pagi, Aman` .

💡 Petunjuk:

- Kunjungi bab penggabungan dari strings untuk informasi lebih lanjut tentang penggabungan string.

Fungsi

Sebuah fungsi adalah blok kode yang dirancang untuk melakukan tugas tertentu dan dieksekusi ketika "sesuatu" memanggilnya. Informasi lebih lanjut tentang fungsi dapat ditemukan di bab fungsi.

📝 Tugas:

- [ ] Tulis program untuk membuat sebuah fungsi yang diberi nama `isGanjil` yang menerima angka `45345` sebagai argumen dan menentukan apakah angka tersebut ganjil atau tidak.
- [ ] Panggil fungsi ini untuk mendapatkan hasil. Hasilnya harus dalam format boolean dan harus mengembalikan `true` di `konsol`.
- [ ] Tulis program untuk membuat sebuah fungsi yang diberi nama `hitungLuasPersegiPanjang` yang mengambil dua parameter `lebar` dan `tinggi` dari persegi panjang dan harus mengembalikan `luas` dari persegi panjang.

💡 Petunjuk:

- Kunjungi bab fungsi untuk memahami fungsi dan cara membuatnya.

Pernyataan Kondisional

Logika kondisional sangat penting dalam pemrograman karena memastikan bahwa program berfungsi tanpa memandang data apa yang Anda masukkan dan juga memungkinkan percabangan. Latihan ini berkaitan dengan implementasi logika kondisional dalam masalah dunia nyata.

📝 Tugas:

- [ ] Tulis program untuk membuat sebuah prompt "*Berapa kilometer yang tersisa untuk pergi?*" dan berdasarkan pengguna serta kondisi berikut, cetak hasilnya di `konsol`.
  - Jika tersisa lebih dari 100 km, cetak: `"Masih ada sedikit perjalanan yang harus Anda lakukan"`.
  - Jika tersisa lebih dari 50 km, tetapi kurang atau sama dengan 100 km, cetak: `"Saya akan tiba dalam 5 menit"`.
  - Jika tersisa kurang dari atau sama dengan 50 km, cetak: `"Saya sedang parkir. Saya akan segera menemui Anda"`.
- [ ] Tulis program yang memeriksa apakah seseorang memenuhi syarat untuk memilih atau tidak berdasarkan usia mereka.
  - Jika usia pengguna adalah 18 tahun atau lebih, maka cetak `Anda berhak memilih`
  - Jika usia pengguna kurang dari 18 tahun, maka cetak `Anda tidak memenuhi syarat untuk memilih`.

  Catatan: `usia` ***dapat berkisar antara*** `1` ***hingga*** `100`***.***

💡 Petunjuk:

- Kunjungi bab logika kondisional untuk memahami bagaimana menggunakan logika kondional dan pernyataan kondisional.

Objek

Objek adalah kumpulan pasangan `kunci` , `nilai` , dan setiap pasangan kunci-nilai dikenal sebagai properti. Di sini, properti dari kunci dapat berupa `string` sedangkan nilainya dapat berupa nilai apa pun.

📝 Tugas:

Diberikan keluarga Doe yang terdiri dari dua anggota, di mana informasi setiap anggota disediakan dalam bentuk objek.

```
let orang = {
  nama: "John", //String
  namaBelakang: "Doe",
  usia: 35, //Number
  jenisKelamin: "laki-laki",
  nomorKeberuntungan: [7, 11, 13, 17], //Array
  pasangan: orang2, //Objek
};

let orang2 = {
  nama: "Jane",
  namaBelakang: "Doe",
  usia: 38,
  jenisKelamin: "perempuan",
  nomorKeberuntungan: [2, 4, 6, 8],
  pasangan: orang,
};

let keluarga = {
  namaBelakang: "Doe",
  anggota: [orang, orang2], //Array of objects
};
```

- [ ] Temukan cara untuk mencetak nama anggota pertama dari keluarga Doe di dalam `konsol` .
- [ ] Ubah nilai keempat `nomorKeberuntungan` dari anggota kedua keluarga Doe menjadi `33` .
- [ ] Tambahkan anggota baru ke keluarga dengan membuat anggota baru (`Jimmy` `Doe` , `13` , `laki-laki` , `[1, 2, 3, 4]` , `null`) dan perbarui daftar anggota.
- [ ] Cetak `JUMLAH` dari nomor keberuntungan keluarga Doe di dalam `konsol` .

💡 Petunjuk:

- Kunjungi bab objek untuk memahami cara kerja objek.
- Anda dapat mendapatkan `nomorKeberuntungan` dari setiap objek orang di dalam objek keluarga.
- Begitu Anda mendapatkan setiap array, cukup lakukan perulangan atasnya, menambahkan setiap elemen, dan kemudian tambahkan setiap hasil penjumlahan dari 3 anggota keluarga.

Dapatkan Judul!

Masalah *Dapatkan Judul!* adalah masalah menarik di mana kita harus mendapatkan judul dari daftar buku. Ini adalah latihan yang baik untuk implementasi array dan objek.

📝 Tugas:

Diberikan sebuah array objek yang mewakili buku dengan penulis.

```
const buku = [
  {
    judul: "Eloquent JavaScript, Edisi Ketiga",
    penulis: "Marijn Haverbeke"
  },
  {
    judul: "Practical Modern JavaScript",
    penulis: "Nicolás Bevacqua"
  }
]
```

- [ ] Tulis program untuk membuat fungsi `dapatkanJudul` yang mengambil array tersebut dan mengembalikan array judul dan mencetak nilainya di `konsol` .

💡 Petunjuk:

- Kunjungi bab array dan objek untuk memahami cara kerja array dan objek.

Bab 22

Pertanyaan wawancara

Bab ini membahas berbagai pertanyaan untuk lebih mempersiapkan calon dalam pemahaman mereka tentang JavaScript. Ini dibagi menjadi tiga bagian: tingkat dasar, tingkat menengah, dan tingkat lanjut.

Pertanyaan Wawancara JavaScript Dasar

1. Sejarah dan Mendefinisikan Variabel.

1.1. Apa itu JavaScript?

Jawaban: JavaScript adalah bahasa pemrograman berlevel tinggi yang digunakan secara umum untuk pengembangan web guna menambahkan interaktivitas dan perilaku dinamis pada situs web.

1.2. Siapa yang menciptakan/Mengembangkan JavaScript?

Jawaban: JavaScript diciptakan oleh *Brendan Eich* saat bekerja di **Netscape Communications Corporation**. Dia mengembangkan bahasa ini dalam waktu hanya sepuluh hari pada bulan Mei 1995. Awalnya, JavaScript dinamai "*Mocha*" namun kemudian diubah menjadi "*LiveScript*" dan akhirnya "*JavaScript*" sebagai bagian dari kolaborasi pemasaran dengan **Sun Microsystems** (sekarang **Oracle Corporation**), yang saat itu sedang populer dengan bahasa pemrograman **Java**. Meskipun mirip dalam nama, *JavaScript* dan *Java* adalah bahasa pemrograman yang berbeda dengan tujuan dan karakteristik yang berbeda.

1.3. Bagaimana cara mendeklarasikan variabel dalam JavaScript?

Jawaban: Anda dapat mendeklarasikan variabel menggunakan `var`, `let`, atau `const`:

- `var` (berumur fungsi)
- `let` (berumur blok)
- `const` (berumur blok, untuk konstanta)

1.4. Apa perbedaan antara `let`, `var`, dan `const`?

Jawaban:

- `var` berumur fungsi, sedangkan `let` dan `const` berumur blok.
- `let` memungkinkan perubahan nilai variabel, sementara `const` digunakan untuk konstanta.
- Variabel yang dideklarasikan dengan `var` diangkat (hoisted), sementara `let` dan `const` tidak diangkat.

1.5. Apakah JavaScript adalah bahasa yang berjenis data statis atau berjenis data dinamis?

Jawaban: JavaScript adalah bahasa yang berjenis data dinamis. Dalam bahasa berjenis data dinamis, tipe variabel diperiksa selama runtime, berbeda dengan bahasa berjenis data statis, di mana tipe variabel diperiksa selama compile-time.

1.6. Apa saja jenis kesalahan dalam JavaScript?

Jawaban: Ada dua jenis kesalahan dalam JavaScript.

1. **Kesalahan sintaksis**: Kesalahan sintaksis adalah kesalahan atau masalah pengejaan dalam kode yang menyebabkan program tidak dijalankan sama sekali atau berhenti di tengah jalan. Biasanya, pesan kesalahan disertakan.

2. **Kesalahan logika**: Kesalahan logika terjadi ketika sintaksis benar tetapi logika atau programnya salah. Dalam kasus ini, program dapat berjalan tanpa masalah, tetapi hasilnya tidak akurat. Kesalahan logika kadang-kadang lebih sulit diperbaiki daripada kesalahan sintaksis karena aplikasi ini tidak menampilkan sinyal kesalahan untuk masalah logika.

1.7. Sebutkan beberapa keunggulan JavaScript.

Jawaban: Ada banyak keunggulan JavaScript. Beberapa di antaranya adalah:

- JavaScript dapat dieksekusi di sisi klien maupun sisi server. Ada berbagai Kerangka Kerja Frontend yang dapat Anda pelajari dan gunakan. Namun, jika Anda ingin menggunakan JavaScript di sisi belakang, Anda perlu mempelajari NodeJS. Saat ini, NodeJS adalah satu-satunya kerangka JavaScript yang dapat digunakan di sisi belakang.
- JavaScript adalah bahasa yang mudah dipelajari.
- Halaman web memiliki lebih banyak fungsionalitas berkat JavaScript.
- Dalam pandangan pengguna akhir, JavaScript sangat cepat.

1.8. Apa itu kata kunci 'this' dalam JavaScript?

Jawaban:

Kata kunci 'this' dalam JavaScript digunakan untuk memanggil objek saat ini sebagai konstruktor untuk memberikan nilai ke properti objek.

2. Fungsi

2.1. Bagaimana cara membuat fungsi dalam JavaScript?

Jawaban:

Anda dapat membuat fungsi menggunakan kata kunci `function` atau dengan menggunakan fungsi panah (`=>`):

Contoh:

```
function myFunction() {
  // Isi fungsi
}

const myArrowFunction = () => {
  // Isi fungsi
};
```

2.2. Apa yang dimaksud dengan Callback?

Jawaban: Callback adalah fungsi yang akan dieksekusi setelah fungsi lain selesai dieksekusi. Dalam JavaScript, fungsi diperlakukan sebagai warga pertama, yang berarti mereka dapat digunakan sebagai argumen fungsi lain, dikembalikan oleh fungsi lain, dan digunakan sebagai properti objek.

Fungsi yang digunakan sebagai argumen untuk fungsi lain disebut fungsi callback. **Contoh**:

```
function bagiDua(jumlah) {
  console.log(Math.floor(jumlah / 2));
}

function kaliDua(jumlah) {
  console.log(jumlah * 2);
}

function operasiPadaJumlah(num1, num2, operasi) {
  var jumlah = num1 + num2;
  operasi(jumlah);
}

operasiPadaJumlah(3, 3, bagiDua); // Output 3

operasiPadaJumlah(5, 5, kaliDua); // Output 20
```

- Dalam kode di atas, kami melakukan operasi matematika pada jumlah dua angka. Fungsi `operasiPadaJumlah` mengambil 3 argumen: angka pertama, angka kedua, dan operasi yang akan dilakukan pada jumlah mereka (fungsi callback).
- Baik `bagiDua` maupun `kaliDua` digunakan sebagai fungsi callback dalam kode di atas.
- Fungsi callback ini akan dieksekusi hanya setelah fungsi `operasiPadaJumlah` dieksekusi.
- Oleh karena itu, sebuah callback adalah fungsi yang akan dieksekusi setelah fungsi lainnya selesai dieksekusi.

2.3. Jelaskan Scope dan Scope Chain dalam JavaScript.

Jawaban: Scope dalam JavaScript menentukan aksesibilitas variabel dan fungsi pada berbagai bagian kode.

Sec

ara umum, scope akan memberi tahu kita pada bagian kode tertentu, variabel dan fungsi mana yang dapat atau tidak dapat diakses.

Ada tiga jenis scope dalam JavaScript:

- Global Scope
- Local atau Function Scope
- Block Scope

Global Scope: Variabel atau fungsi yang dideklarasikan dalam ruang nama global memiliki scope global, yang berarti semua variabel dan fungsi yang memiliki scope global dapat diakses dari mana saja dalam kode.

```
var variabelGlobal = "Halo dunia";

function kirimPesan() {
  return variabelGlobal; // dapat mengakses variabelGlobal karena ditulis dalam ruang global
}
function kirimPesan2() {
  return kirimPesan(); // Dapat mengakses fungsi kirimPesan karena ditulis dalam ruang global
}
kirimPesan2(); // Mengembalikan "Halo dunia"
```

Function Scope: Variabel atau fungsi yang dideklarasikan dalam fungsi memiliki scope lokal/fungsi, yang berarti semua variabel dan fungsi yang dideklarasikan dalam fungsi hanya dapat diakses dari dalam fungsi tersebut dan tidak di luar.

```
function fungsiKeren() {
  var a = 2;

  var perkalianDua = function () {
    console.log(a * 2); // Dapat mengakses variabel "a" karena keduanya ditulis dalam fungsi yang sama
  };
}
console.log(a); // Menimbulkan kesalahan referensi karena "a" ditulis dalam scope lokal dan tidak dapat diakses

perkalianDua(); // Menimbulkan kesalahan referensi karena perkalianDua ditulis dalam scope lokal
```

Block Scope: `Block scope` terkait dengan variabel yang dideklarasikan menggunakan let dan const. Variabel yang dideklarasikan dengan var tidak memiliki block scope. Block scope memberi tahu kita bahwa variabel yang dideklarasikan dalam blok `{ }` hanya dapat diakses di dalam blok itu dan tidak di luarnya.

```
{
  let x = 45;
}

console.log(x); // Menimbulkan kesalahan referensi karena "x" tidak dapat diakses di luar blok

for (let i = 0; i < 2; i++) {
  // lakukan sesuatu
}

console.log(i); // Menimbulkan kesalahan referensi karena "i" tidak dapat diakses di luar blok loop
```

Scope Chain: Mesin JavaScript juga menggunakan Scope untuk mencari variabel. Mari pahami ini dengan contoh:

```
var y = 24;

function fungsiFavorit() {
  var x = 667;
  var fungsiFavoritLainnya = function () {
    console.log(x); // Tidak menemukan "x" di dalam fungsiFavoritLainnya, jadi mencari variabel dalam fungsiFav
  };

  var fungsiFavoritLainnyaLagi = function () {
    console.log(y); // Tidak menemukan "y" di dalam fungsiFavoritLainnyaLagi, jadi mencari variabel dalam fungs
  };

  fungsiFavoritLainnya();
  fungsiFavoritLainnyaLagi();
}
fungsiFavorit();
```

Seperti yang dapat dilihat dalam kode di atas, jika mesin JavaScript tidak menemukan variabel dalam scope lokal, ia mencoba mencari variabel dalam scope yang lebih tinggi (outer scope). Jika variabel tersebut tidak ada dalam ruang lingkup global, maka akan menghasilkan kesalahan referensi.

2.4. Jelaskan Higher Order Functions dalam JavaScript.

Jawaban: Fungsi yang beroperasi pada fungsi lain, baik dengan mengambilnya sebagai argumen atau dengan mengembalikannya, disebut *higher-order functions*.

Higher-order functions adalah hasil dari fungsi menjadi *warga pertama* (first-class citizens) dalam JavaScript.

Contoh higher-order functions:

```
function higherOrder(fn) {
  fn();
}

higherOrder(function () {
  console.log("Halo dunia");
});

function higherOrder2() {
  return function () {
    return "Lakukan sesuatu";
  };
}

var x = higherOrder2();
x(); // Mengembalikan "Lakukan sesuatu"
```

2.5. Apa yang dimaksud dengan Self Invoking Functions dalam JavaScript?

Jawaban:

Self-invoking expression adalah ekspresi yang secara otomatis dijalankan (diinisiasi) tanpa diminta. Jika ekspresi fungsi diikuti oleh (), maka akan dieksekusi secara otomatis. Deklarasi fungsi tidak dapat dijalankan sendiri.

Biasanya, kita mendeklarasikan fungsi dan kemudian memanggilnya, namun fungsi anonim dapat digunakan untuk menjalankan fungsi secara otomatis saat didefinisikan dan tidak akan dipanggil lagi. Dan tidak ada nama untuk jenis fungsi ini.

2.6. Apa perbedaan antara metode `exec()` dan `test()` dalam JavaScript?

Jawaban:

- `test()` dan `exec()` adalah metode ekspresi Reguler dalam JavaScript.
- Kami akan menggunakan `exec()` untuk mencari string untuk pola tertentu dan jika menemukannya, akan mengembalikan pola tersebut secara langsung; jika tidak, akan mengembalikan hasil 'kosong'.
- Kami akan menggunakan `test()` untuk mencari string untuk pola tertentu. Ini akan mengembalikan nilai Boolean 'true' jika menemukan teks yang diberikan; jika tidak, akan mengembalikan 'false'.

2.7. Apa perbedaan antara deklarasi fungsi (Function declaration) dan ekspresi fungsi (Function expression) dalam JavaScript?

Jawaban

- **Deklarasi Fungsi (Function Declaration)**: a) Dinyatakan sebagai pernyataan terpisah dalam kode JavaScript utama. b) Dapat dipanggil sebelum fungsi didefinisikan. c) Menawarkan keterbacaan kode yang lebih baik dan organisasi kode yang lebih baik. d) Contoh:

  ```
  function abc() {
    return 5;
  }
  ```

- **Ekspresi Fungsi (Function Expression)**: a) Dibuat dalam suatu ekspresi atau konstruksi lain. b) Dibuat saat titik eksekusi mencapai ekspresi tersebut; hanya dapat digunakan setelah itu. c) Dig

unakan ketika diperlukan deklarasi fungsi kondisional. d) Contoh:

```
var a = function abc() {
  return 5;
};
```

2.8. Apa itu fungsi anak panah (Arrow functions) dalam JavaScript?

Jawaban

- Fungsi anak panah adalah cara penulisan singkat dan ringkas untuk menulis fungsi dalam JavaScript. Sintaks umum fungsi anak panah adalah sebagai berikut:

```
const helloWorld = () => {
  console.log("Halo dunia!");
};
```

2.9. Dalam konteks "Passed by value" dan "Passed by reference":

Jawaban

- **Passed By Value adalah Tipe Data Primitive**. Pertimbangkan contoh berikut:

  Di sini, `a = 432` adalah tipe data primitif, yaitu tipe number yang memiliki nilai yang diberikan oleh operator. Ketika kode `var b = a` dijalankan, nilai dari `var a` mengembalikan alamat baru untuk `var b` dengan mengalokasikan ruang baru di memori, sehingga `var b` akan beroperasi di lokasi yang baru.

  Contoh:

```
var a = 432;
var b = a;
```

- **Passed by Reference adalah Tipe Data Non-primitive**. Pertimbangkan contoh berikut:

  Referensi dari objek variabel pertama, yaitu `var obj` , dilewatkan melalui lokasi variabel lainnya, yaitu `var obj2` , dengan bantuan operator yang diberikan.

  Contoh:

```
var obj = { name: "Raj", surname: "Sharma" };
var obj2 = obj;
```

3. Tipe Data dan Operator

3.1. Apa saja tipe data yang ada dalam JavaScript?

Jawaban:

1. **Tipe data Primitif**
   - `String` - Ini mewakili serangkaian karakter dan ditulis dengan tanda kutip. Sebuah string dapat direpresentasikan dengan menggunakan tanda kutip tunggal atau ganda.

     Contoh:

```
var str = "Vivek Singh Bisht"; //menggunakan tanda kutip ganda
var str2 = "John Doe"; //menggunakan tanda kutip tunggal
```

- `Number` - Ini mewakili sebuah angka dan dapat ditulis dengan atau tanpa desimal.

  Contoh:

  ```
  var x = 3; //tanpa desimal
  var y = 3.6; //dengan desimal
  ```

- `BigInt` - Tipe data ini digunakan untuk menyimpan angka yang melebihi batasan tipe data Number. Ini dapat menyimpan angka besar dan direpresentasikan dengan menambahkan "n" ke literal angka.

  Contoh:

  ```
  var bigInteger = 234567890123456789012345678901234567890n;
  ```

- `Boolean` - Ini mewakili entitas logis dan hanya dapat memiliki dua nilai: true atau false. Boolean biasanya digunakan untuk pengujian kondisional.

  Contoh:

  ```
  var a = 2;
  var b = 3;
  var c = 2;
  (a == b)(
    // mengembalikan false
    a == c
  ); // mengembalikan true
  ```

- `Undefined` - Saat variabel dideklarasikan tetapi tidak diisi, nilainya adalah undefined dan tipenya juga undefined.

  Contoh:

  ```
  var x; // nilai x adalah undefined
  var y = undefined; // kita juga bisa mengatur nilai variabel sebagai undefined
  ```

- `Null` - Ini mewakili nilai yang tidak ada atau tidak valid.

  Contoh:

  ```
  var z = null;
  ```

- `Symbol` - Ini adalah tipe data baru yang diperkenalkan dalam versi ES6 JavaScript. Digunakan untuk menyimpan nilai anonim dan unik.

  Contoh:

  ```
  var symbol1 = Symbol("symbol");
  ```

- **Array** - Ini adalah kumpulan data yang dipesan yang digunakan untuk menyimpan sejumlah nilai dalam satu variabel.

  Contoh:

```
var array1 = [5, "Halo", true, 4.1];
```

Note: Dalam JavaScript, tipe data primitif hanya dapat menyimpan satu nilai. Untuk menyimpan nilai yang lebih kompleks, kita menggunakan tipe data non-primitif.

2. **Tipe data Non-Primitif**
   - **Object** - Digunakan untuk menyimpan koleksi data.

     Contoh:

```
// Koleksi data dalam pasangan nama-nilai
var obj1 = {
  x: 43,
  y: "Halo dunia!",
  z: function () {
    return this.x;
  },
};
```

3.2. Apa perbedaan antara operator `==` dan `===` ?

Jawaban: Kedua operator tersebut adalah operator perbandingan. Perbedaan antara keduanya adalah bahwa `==` digunakan untuk membandingkan nilai, sedangkan `===` digunakan untuk membandingkan baik nilai maupun tipe data.

Contoh:

```
var x = 2;
var y = "2";

x == y; // Mengembalikan true karena nilainya sama, meskipun tipe datanya berbeda
x === y; // Mengembalikan false karena selain nilai, tipe datanya juga harus sama
```

````
### 3.3. Apa itu properti NaN dalam JavaScript?

**Jawaban:**
Properti NaN mewakili nilai "Not-a-Number" yang disengaja. Ini menunjukkan nilai yang bukan angka yang sah.

Hasil dari `typeof NaN` akan mengembalikan "number". Untuk memeriksa apakah suatu nilai adalah NaN, kita menggu

**Contoh**:

```js
isNaN("Halo"); // Mengembalikan true
isNaN(345); // Mengembalikan false
isNaN("1"); // Mengembalikan false, karena '1' diubah menjadi tipe data Number, yang menghasilkan 0 (angka)
isNaN(true); // Mengembalikan false, karena true diubah menjadi tipe data Number, yang menghasilkan 1 (angka)
isNaN(false); // Mengembalikan false
isNaN(undefined); // Mengembalikan true
```

### 3.4. Metode apa yang digunakan untuk mengambil karakter dari indeks tertentu?

**Jawaban:**
Fungsi `charAt()` dalam JavaScript digunakan untuk mencari karakter pada indeks tertentu. Angka indeks dimulai

Contoh:

```js
var kata = "Halo dunia";
var karakter = kata.charAt(3); // Mengembalikan karakter "o" (indeks 3)
```

````md
# 4. Beberapa Konsep Penting

## 4.1. Apa yang dimaksud dengan Mode Ketat (Strict Mode) dalam JavaScript?

**Jawaban:**
Mode Ketat adalah fitur baru dalam ECMAScript 5 yang memungkinkan Anda untuk menempatkan program atau fungsi da

## 4.2. Mengapa kita menggunakan kata "debugger" dalam JavaScript?

**Jawaban:**
Kata kunci "debugger" digunakan untuk membuat breakpoint dalam kode. Ketika peramban menemukan kata kunci "debu

## 4.3. Apa itu currying dalam JavaScript?

**Jawaban:**
Currying adalah teknik lanjutan untuk mengubah fungsi dengan argumen n menjadi n fungsi dengan satu atau lebih

Contoh dari fungsi yang dicurrying:

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

tambah(3)(4); // Hasilnya adalah 7
```
````

Dengan menggunaan teknik currying, kita tidak mengubah fungsionalitas fungsi, hanya cara memanggilnya.

4.4. Apa keuntungan menggunakan JavaScript Eksternal?
````

Jawaban: JavaScript Eksternal adalah kode JavaScript (script) yang ditulis dalam file terpisah dengan ekstensi .js, dan kemudian kita menghubungkan file tersebut dalam elemen atau file HTML tempat kode tersebut ditempatkan.

Beberapa keuntungan dari penggunaan JavaScript eksternal adalah:

- Memungkinkan desainer web dan pengembang untuk berkolaborasi pada file HTML dan JavaScript.
- Kode dapat digunakan kembali.
- Kode menjadi lebih mudah dibaca dalam JavaScript eksternal.

4.5. Apa itu closure dalam JavaScript?

Jawaban: Closure adalah fungsi yang memiliki akses ke cakupan fungsi induknya bahkan setelah fungsi induknya sudah selesai. Ini berarti closure dapat mengingat dan mengakses variabel dan argumen dari fungsi induknya bahkan setelah fungsi tersebut selesai.

Dalam singkat, closure adalah fungsi yang memiliki akses ke variabel dari lingkup luar (enclosing function) bahkan setelah fungsi tersebut selesai dijalankan.

4.6. Apa itu DOM dalam JavaScript?

Jawaban: Document Object Model (DOM) adalah antarmuka pemrograman untuk dokumen HTML dan XML. DOM merepresentasikan halaman sehingga program dapat mengubah struktur, gaya, dan kontennya. DOM menggambarkan dokumen sebagai simpul dan objek.

4.7. Apa yang dimaksud dengan delegasi acara?

Jawaban: Delegasi acara adalah teknik untuk mendengarkan acara di mana elemen induk (biasanya yang lebih tinggi dalam hirarki dokumen) bertindak sebagai pemantau acara untuk elemen-elemen anaknya. Acara-acara yang dipancarkan oleh elemen anak akan ditangani oleh pemantau acara elemen induk.

4.8. Bagaimana cara melakukan permintaan AJAX dalam JavaScript?

Jawaban: AJAX singkatan dari Asynchronous JavaScript and XML. Ini adalah seperangkat teknik pengembangan web yang menggunakan banyak teknologi web di sisi klien untuk membuat aplikasi web asinkron. Dengan AJAX, aplikasi web dapat mengirim dan mengambil data dari server secara asinkron (di latar belakang) tanpa mengganggu tampilan dan perilaku halaman yang ada.

Anda dapat melakukan permintaan AJAX menggunakan objek XMLHttpRequest atau dengan menggunakan API fetch. Berikut adalah contoh menggunakan fetch:

```
fetch("https://contoh.com/api/data")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error(error));
```

4.9. Apa itu promise dalam JavaScript?

Jawaban: Promise adalah objek yang dapat menghasilkan satu nilai tunggal di masa depan, baik itu nilai yang dipenuhi atau alasan penolakan. Promise dapat berada dalam salah satu dari 3 keadaan: terpenuhi, ditolak, atau masih dalam keadaan tertunda. Pengguna promise dapat melampirkan callback untuk menghandle nilai yang dipenuhi atau alasan penolakan.

4.10. Mengapa kita memerlukan promise dalam JavaScript?

Jawaban: Promise digunakan untuk mengatasi operasi asinkron. Promise memberikan pendekatan alternatif terhadap callback dengan mengurangi kompleksitas callback dan menulis kode yang lebih bersih.

5. Objek

5.1. Apa cara-cara yang mungkin digunakan untuk membuat objek dalam JavaScript?

Jawaban: Ada beberapa cara untuk membuat objek dalam JavaScript, antara lain:

1. **Konstruktor Objek:**
   - Cara termudah untuk membuat objek adalah dengan menggunakan konstruktor objek, seperti `new Object()`. Namun, ini sekarang tidak disarankan.

```
var obj = new Object();
```

2. **Metode `create` Objek:**
   - Metode `create` dari objek `Object` digunakan untuk membuat objek baru dengan objek prototipe yang ditentukan.

```
var obj = Object.create(null);
```

3. **Notasi Objek Literal:**
   - Notasi objek literal (atau inisialisasi objek) adalah kumpulan pasangan nama-nilai yang dipisahkan oleh koma dan dikelilingi oleh kurung kurawal.

```
var obj = {
  nama: "Sudheer",
  usia: 34,
};
```

4. **Konstruktor Fungsi:**
   - Anda dapat membuat fungsi konstruktor yang menghasilkan objek menggunakan operator `new`.

```
function Person(nama) {
  this.nama = nama;
}
var obj = new Person("Sudheer");
```

5. **Konstruktor Fungsi dengan Prototype:**
   - Ini mirip dengan konstruktor fungsi, tetapi menggunakan prototipe untuk properti dan metode.

```
function Person() {}
Person.prototype.nama = "Sudheer";
var obj = new Person();
```

6. **Notasi Kelas ES6:**
   - Dalam ES6, Anda dapat menggunakan notasi kelas untuk membuat objek.

```
class Person {
  constructor(nama) {
    this.nama = nama;
  }
}
var obj = new Person("Sudheer");
```

6. Berbagai Hal Lain

6.1. Apa itu mode ketat (Strict Mode) dalam JavaScript?

Jawaban: Mode Ketat adalah fitur baru dalam ECMAScript 5 yang memungkinkan Anda untuk menempatkan program, atau fungsi, dalam konteks operasi "ketat". Ini menghalangi beberapa tindakan tertentu dan menghasilkan lebih banyak pengecualian. Ekspresi harfiah "use strict"; memberi tahu peramban untuk menggunakan kode JavaScript dalam Mode Ketat.

6.2. Apa itu nilai null dalam JavaScript?

Jawaban: Nilai null mewakili ketiadaan objek yang disengaja. Ini adalah salah satu nilai primitif dalam JavaScript. Tipe dari nilai null adalah object. Anda dapat mengosongkan variabel dengan mengatur nilainya menjadi null.

```
var user = null;
console.log(typeof user); // object
```

6.3. Apa itu eval dalam JavaScript?

Jawaban: F

ungsi eval() dalam JavaScript digunakan untuk mengevaluasi kode JavaScript yang direpresentasikan sebagai string. String tersebut dapat berisi ekspresi JavaScript, variabel, pernyataan, atau rangkaian pernyataan.

```
console.log(eval("1 + 2")); // 3
```

6.4. Apakah JavaScript bahasa yang dikompilasi atau diinterpretasi?

Jawaban: JavaScript adalah bahasa yang diinterpretasi, bukan dikompilasi. Interpreter di peramban membaca kode JavaScript, menguraikannya, dan mengeksekusinya. Saat ini, peramban modern menggunakan teknologi yang disebut kompilasi Just-In-Time (JIT) untuk mengkompilasi JavaScript menjadi kode bytecode yang dapat dijalankan.

6.5. Apakah JavaScript adalah bahasa yang peka huruf besar-kecil (case-sensitive)?

Jawaban: Ya, JavaScript adalah bahasa yang peka huruf besar-kecil (case-sensitive). Kata kunci bahasa, variabel, nama fungsi, dan identifier lainnya harus selalu ditulis dengan kapitalisasi huruf yang konsisten.

6.6. Apa perbedaan antara metode `exec()` dan `test()`?

Jawaban:

- `exec()` : Ini adalah metode ekspresi reguler yang digunakan untuk mencari string dengan pola tertentu. Ketika itu ditemukan, metode ini akan mengembalikan hasil pola tersebut, atau jika tidak ditemukan, akan mengembalikan hasil "kosong".
- `test()` : Ini adalah metode ekspresi reguler yang digunakan untuk mencari string dengan pola tertentu atau teks. Jika itu ditemukan, metode ini akan mengembalikan nilai boolean `true`, jika tidak ditemukan, akan mengembalikan nilai `false`.

Pertanyaan Wawancara JavaScript Tingkat Menengah

1. Perulangan (Loops)

1.1. Apa definisi iterasi dalam perulangan JavaScript?

Jawaban:

Iterasi dalam perulangan JavaScript merujuk pada setiap eksekusi individu dari tubuh perulangan, biasanya sesuai dengan satu siklus perulangan.

1.2. Apa saja struktur perulangan dalam JavaScript?

Jawaban:

Perulangan while: Perulangan while adalah pernyataan aliran kendali yang memungkinkan kode dieksekusi secara berulang berdasarkan kondisi Boolean yang diberikan. Perulangan while dapat dianggap sebagai pernyataan if berulang.

Perulangan for: Perulangan for memberikan cara ringkas untuk menulis struktur perulangan. Berbeda dengan perulangan while, pernyataan for mengonsumsi inisialisasi, kondisi, dan penambahan/pengurangan dalam satu baris, sehingga memberikan struktur perulangan yang lebih singkat, mudah didebug.

Do while: Perulangan do-while mirip dengan perulangan while dengan satu perbedaan, yaitu bahwa ia memeriksa kondisi setelah mengeksekusi pernyataan, dan oleh karena itu merupakan contoh dari Perulangan Kontrol Keluar.

1.3. Bagaimana cara kerja pernyataan break dalam perulangan?

Jawaban:

Pernyataan break mengakhiri perulangan atau pernyataan switch saat ini dan mengalihkan kendali program ke pernyataan yang mengikuti pernyataan yang diakhiri. Ini juga dapat digunakan untuk melompati pernyataan berlabel saat digunakan dalam pernyataan berlabel.

1.4. Bagaimana cara kerja pernyataan continue dalam perulangan?

Jawaban:

Direktif continue adalah "versi ringan" dari pernyataan break. Ini tidak menghentikan seluruh perulangan; sebaliknya, itu menghentikan iterasi saat ini dan memaksa perulangan memulai yang baru (jika kondisinya memungkinkan).

2. Pernyataan Switch (Switch Statement)

2.1. Apa itu pernyataan switch dalam JavaScript?

Jawaban:

Pernyataan switch dalam JavaScript adalah pernyataan aliran kendali yang mengevaluasi suatu ekspresi dan mengeksekusi blok kode tertentu berdasarkan kasus yang cocok.

2.2. Apa keuntungan menggunakan pernyataan switch?

Jawaban:

Pernyataan switch dapat menggantikan beberapa pemeriksaan, lebih deskriptif, dan lebih mudah dibaca. Pernyataan switch meningkatkan keterbacaan kode, memberikan kinerja yang lebih baik, menyederhanakan kondisi yang kompleks, meningkatkan pemeliharaan, dan mendukung sintaks yang lebih bersih.

2.3. Apakah urutan pernyataan case penting dalam pernyataan switch?

Jawaban:

Urutan pernyataan case penting dalam pernyataan switch, terutama saat menggunakan perilaku jatuh. Kasus dievaluasi secara berurutan, sehingga kasus yang cocok yang ditemukan lebih awal akan mencegah pengujian kasus selanjutnya, yang memengaruhi eksekusi dan kinerja.

3. Cookie JavaScript

3.1. Apa itu Cookie JavaScript?

Jawaban:

Cookie adalah file kecil yang disimpan di komputer pengguna. Mereka digunakan untuk menyimpan sejumlah kecil data yang khusus untuk klien dan situs web tertentu, dan dapat diakses baik oleh server web maupun komputer klien. Ketika cookie ditemukan, mereka awalnya berisi informasi tentang preferensi pengguna. Misalnya, saat Anda memilih bahasa tampilan situs web, situs web akan menyimpan informasi tersebut dalam cookie di komputer Anda, sehingga ketika Anda mengunjungi situs web tersebut kembali, itu dapat membaca cookie yang disimpan sebelumnya.

3.2. Bagaimana cara membuat cookie menggunakan JavaScript?

Jawaban:

Untuk membuat cookie menggunakan JavaScript, Anda hanya perlu menetapkan nilai string ke objek document.cookie.

```
document.cookie = "key1 = value1; key2 = value2; expires = date";
```

3.3. Bagaimana cara membaca cookie menggunakan JavaScript?

Jawaban:

Nilai dari document.cookie digunakan untuk membuat cookie. Kapan pun Anda ingin mengakses cookie, Anda dapat menggunakan string ini. String document.cookie menyimpan daftar pasangan nama = nilai yang dipisahkan oleh titik koma, di mana nama adalah nama cookie dan nilai adalah nilai stringnya.

3.4. Bagaimana cara menghapus cookie menggunakan JavaScript?

Jawaban:

Menghapus cookie jauh lebih mudah daripada membuat atau membaca cookie. Anda hanya perlu mengatur expires = "waktu yang lalu" dan pastikan satu hal menentukan jalur cookie yang benar, kecuali beberapa tidak akan memungkinkan Anda menghapus cookie.

4. Kotak Pop-up di JavaScript

4.1. Apa jenis kotak pop-up yang tersedia dalam JavaScript?

Jawaban:

Ada tiga jenis kotak pop-up yang tersedia dalam JavaScript: Alert, Confirm, Prompt.

4.2. Apa perbedaan antara kotak peringatan (alert box) dan kotak konfirmasi (confirmation box)?

Jawaban:

Kotak peringatan (alert box) hanya akan menampilkan satu tombol yaitu tombol OK. Ini digunakan untuk memberi tahu pengguna tentang persetujuan yang harus disetujui. Tetapi kotak konfirmasi (confirmation box) menampilkan dua tombol, OK dan Batal, di mana pengguna dapat memutuskan untuk setuju atau tidak.

5. Fungsi Panah (Arrow Functions)

5.1. Apa definisi fungsi panah (arrow function)?

Jawaban:

Fungsi panah adalah sintaksis ringkas untuk mendefinisikan fungsi anonim dalam JavaScript, menggunakan notasi panah. Ini menawarkan sintaksis yang lebih pendek, lingkup leksikal "this," dan tidak dapat digunakan sebagai konstruktor.

5.2. Bagaimana fungsi pan

ah berbeda dari ekspresi fungsi?

Jawaban:

Fungsi panah memberikan sintaksis yang lebih pendek, tidak memiliki "this," "arguments," "super," atau "new.target" mereka sendiri, dan tidak dapat digunakan sebagai konstruktor, berbeda dari ekspresi fungsi.

5.3. Apa yang dimaksud dengan ikatan "this" leksikal dalam fungsi panah?

Jawaban:

Ikatan "this" leksikal dalam fungsi panah berarti bahwa mereka tidak membuat konteks "this" mereka sendiri; sebaliknya, mereka mewarisi "this" dari cakupan yang mengelilingi mereka, mengurangi masalah umum terkait "this."

5.4. Apa keuntungan menggunakan fungsi panah?

Jawaban:

Keuntungan menggunakan fungsi panah dalam JavaScript meliputi sintaksis yang lebih pendek, pengembalian implisit, dan ikatan 'this' leksikal.

5.5. Apa kasus penggunaan umum untuk fungsi panah?

Jawaban:

Fungsi panah umumnya digunakan untuk metode objek, penangan acara, pemanggilan kembali, dan fungsi lain yang memerlukan sintaksis yang lebih pendek dan lebih ringkas.

Pertanyaan Wawancara JavaScript Tingkat Lanjut

1. Closures dan Ruang Lingkup

1.1. Apa itu closure dalam JavaScript? Berikan contoh di mana penggunaan closure dapat bermanfaat.

Jawaban:

Closure dalam JavaScript adalah fungsi yang memiliki akses ke variabel-variabel dalam ruang lingkup yang mengelilinginya, bahkan setelah fungsi luar selesai dieksekusi. Mekanisme ini memungkinkan fungsi untuk mempertahankan status antara eksekusi.

Contoh: Salah satu penggunaan umum dari closure adalah untuk membuat fungsi pabrik atau variabel privat. Misalnya, jika kita ingin menghasilkan nilai ID unik untuk elemen:

1.2. Bagaimana keterkaitan antara closure dengan ruang lingkup dan masa pakai variabel?

Jawaban:

Closure memungkinkan suatu fungsi untuk mengakses semua variabel, serta fungsi, yang berada dalam ruang lingkup leksikalnya, bahkan setelah fungsi luar selesai dieksekusi. Hal ini mengakibatkan variabel-variabel tersebut tetap ada dalam memori, yang efektif memungkinkan variabel memiliki masa pakai yang lebih lama dibandingkan dengan variabel lokal standar yang biasanya akan dihapuskan setelah fungsi utamanya dieksekusi.

1.3. Berikan beberapa contoh penggunaan closure dalam JavaScript.

Jawaban:

Berikut beberapa contoh penggunaan closure:

- Pola Desain Modul.
- Currying.
- Memoisasi

2. Pewarisan Prototipal

2.1. Jelaskan perbedaan antara pewarisan klasik dan pewarisan prototipal.

Jawaban:

Pewarisan klasik adalah konsep yang paling sering ditemukan dalam bahasa pemrograman berorientasi objek tradisional seperti Java atau C++, di mana sebuah kelas dapat mewarisi properti dan metode dari kelas induk. Pewarisan prototipal, di sisi lain, unik untuk JavaScript. Dalam JavaScript, setiap objek dapat memiliki objek lain sebagai prototipnya, dan dapat mewarisi properti dari prototipnya.

Perbedaan utama adalah bahwa pewarisan klasik berbasis kelas, sedangkan pewarisan prototipal berbasis objek. Meskipun ES6 memperkenalkan kata kunci `class` ke JavaScript, ini hanyalah gula sintaksis di atas pewarisan prototipal yang ada.

2.2. Bagaimana Anda dapat memperluas objek JavaScript bawaan?

Jawaban:

Untuk memperluas objek JavaScript bawaan, kita dapat menambahkan metode atau properti ke prototip mereka. Namun, umumnya tidak disarankan untuk memodifikasi prototip bawaan karena dapat mengakibatkan masalah kompatibilitas dan perilaku yang tidak terduga, terutama jika ada perubahan di spesifikasi JavaScript di masa depan.

3. JavaScript Asynchronous

3.1. Jelaskan loop acara dalam JavaScript. Bagaimana hubungannya dengan tumpukan panggilan?

Jawaban:

Loop acara adalah konsep dasar dalam JavaScript yang bertanggung jawab untuk mengelola eksekusi beberapa bagian program dari waktu ke waktu, masing-masing dieksekusi hingga selesai. Ini bekerja sebagai loop yang terus memeriksa apakah ada tugas yang menunggu dalam antrean pesan. Jika ada tugas dan tumpukan panggilan utama (tumpukan panggilan) kosong, maka tugas tersebut diambil dari antrean dan dieksekusi.

Tumpukan panggilan adalah struktur data yang melacak eksekusi fungsi dalam program. Ketika sebuah fungsi dipanggil, ia ditambahkan ke tumpukan panggilan, dan ketika selesai dieksekusi, ia dihapus dari tumpukan. Dalam konteks JavaScript, loop acara terus-menerus memeriksa tumpukan panggilan untuk menentukan apakah tumpukan tersebut kosong. Jika tumpukan kosong dan ada fungsi callback yang menunggu dalam antrean pesan, callback tersebut dieksekusi.

3.2. Apa itu promise, dan bagaimana perbedaannya dengan callback dalam mengelola operasi asinkron?

Jawaban:

Promise adalah objek yang mewakili penyelesaian (atau kegagalan) dari operasi asinkron serta hasilnya. Sebuah `Promise` dapat berada dalam salah satu dari tiga keadaan:

- `pending` : keadaan awal, belum terpenuhi atau terpenuhi.
- `fulfilled` : artinya operasi asinkron telah selesai dan `Promise` memiliki hasilnya.
- `rejected` : artinya operasi asinkron gagal, dan `Promise` tidak memiliki hasil.

Callback adalah fungsi yang diberikan ke fungsi lain sebagai argumen dan dieksekusi setelah fungsi luar selesai. Meskipun baik promise dan callback dapat mengelola operasi asinkron, promise menyediakan cara yang lebih kuat dan terstruktur untuk mengelola operasi tersebut.

Perbedaan utama antara keduanya meliputi:

- Promise memungkinkan rantai operasi asinkron yang lebih baik.
- Callback dapat mengarah pada callback hell atau pyramid of doom, di mana kode menjadi sulit dibaca dan dikelola karena callback bertingkat.
- Promise memiliki mekanisme penanganan kesalahan yang terstandarisasi menggunakan `.then` dan `.catch` .

3.3. Jelaskan async/await. Bagaimana cara membuat kode asinkron lebih sederhana?

Jawaban:

`async/await` adalah fitur sintaksis yang diperkenalkan di ES8 (atau ES2017) untuk mengelola kode asinkron dengan cara yang lebih mirip dengan kode sinkron. Ini memungkinkan penulisan operasi asinkron dengan cara linier tanpa menggunakan callback, menghasilkan kode yang lebih bersih dan mudah dibaca.

Kata kunci `async` digunakan untuk mendeklarasikan fungsi asinkron, yang memastikan bahwa fungsi mengembal

ikan promise. Kata kunci `await` digunakan dalam fungsi asinkron untuk menghentikan eksekusi hingga promise diselesaikan atau ditolak.

Menggunakan `async/await` menyederhanakan penanganan kesalahan, karena kita dapat menggunakan blok `try/catch` tradisional alih-alih `.catch` dengan promise.

4. Metode Array Lanjutan

4.1. Jelaskan fungsi `map`, `reduce`, dan `filter`. Berikan contoh penggunaan praktis untuk masing-masing.

Jawaban:

- `map`: Fungsi ini mengubah setiap elemen dalam array berdasarkan sebuah fungsi, mengembalikan array baru dengan panjang yang sama. **Contoh:** Menggandakan setiap angka dalam array.

```
const angka = [1, 2, 3, 4];
const hasilGanda = angka.map((num) => num * 2); // [2, 4, 6, 8]
```

4.2. Apa keterbatasan atau potensi kesalahan saat menggunakan fungsi panah?

Jawaban:

Fungsi panah memperkenalkan cara ringkas untuk menulis fungsi dalam JavaScript, namun mereka memiliki beberapa keterbatasan:

1. **Tidak Ada Pengikatan `this`**: Fungsi panah tidak mengikat `this` mereka sendiri. Mereka mewarisi pengikatan `this` dari ruang lingkup sekitarnya. Ini dapat menjadi masalah, terutama saat menggunakannya sebagai metode dalam objek atau sebagai penangan acara.
2. **Tidak Ada Objek `arguments`**: Fungsi panah tidak memiliki objek `arguments` sendiri. Jika kita perlu mengakses objek `arguments`, kita harus menggunakan fungsi ekspresi biasa.
3. **Tidak Bisa Digunakan Sebagai Konstruktor**: Fungsi panah tidak dapat digunakan sebagai konstruktor dengan kata kunci `new` karena mereka tidak memiliki metode internal `[[Construct]]`.
4. **Tidak Ada Properti `prototype`**: Berbeda dengan fungsi biasa, fungsi panah tidak memiliki properti `prototype`.
5. **Kurang Mudah Dibaca untuk Logika yang Kompleks**: Untuk operasi sederhana, sintaks yang ringkas adalah keuntungan. Namun, untuk fungsi yang mengandung logika yang kompleks, sintaks yang ringkas bisa membuat kode kurang mudah dibaca.

5. Kata Kunci "this"

5.1. Jelaskan perilaku kata kunci `this` dalam konteks yang berbeda, seperti dalam sebuah metode, fungsi mandiri, fungsi panah, dan penangan acara.

Jawaban: Kata kunci `this` dapat berbeda tergantung pada konteksnya:

- Dalam Metode: Merujuk ke objek yang metode tersebut dipanggil padanya.

```
const person = {
  name: "Alice",
  sayHello: function () {
    console.log(`Hello, my name is ${this.name}`);
  },
};
person.sayHello(); // Output: Hello, my name is Alice
```

- Dalam Fungsi Mandiri: Perilaku `this` tergantung pada cara fungsi tersebut dipanggil. Jika fungsi dipanggil dalam lingkup global, `this` merujuk ke objek global (misalnya, `window` di peramban web).

```
function greet() {
  console.log(`Hello, my name is ${this.name}`);
}

const name = "Alice";
greet(); // Output: Hello, my name is Alice
```

- Dalam Fungsi Panah: Fungsi panah menangkap nilai `this` dari lingkup leksikalnya, tidak memiliki `this` sendiri.

```
const person = {
  name: "Bob",
  sayHello: () => {
    console.log(`Hello, my name is ${this.name}`);
  },
};
person.sayHello(); // Output: Hello, my name is undefined
```

5.2. Bagaimana Anda dapat memastikan bahwa suatu fungsi menggunakan objek tertentu sebagai nilai `this` -nya?

Jawaban:

Anda dapat memastikan bahwa suatu fungsi menggunakan objek tertentu sebagai nilai `this` dengan menggunakan metode seperti bind, fungsi panah, call, apply, atau dengan mendefinisikan metode dalam kelas ES6. Teknik-teknik ini memungkinkan kita mengendalikan konteks di mana fungsi beroperasi dan memastikan bahwa ia mengakses properti dan metode objek yang diinginkan.

6. Pengelolaan Memori

6.1. Apa yang dimaksud dengan kebocoran memori dalam JavaScript? Diskusikan penyebab potensial dan cara untuk mencegahnya.

Jawaban:

Kebocoran memori dalam JavaScript terjadi saat program dengan tidak sengaja mempertahankan referensi ke objek yang tidak lagi diperlukan, mengakibatkan penggunaan memori yang berlebihan dan masalah potensial dalam aplikasi. Penyebab umum kebocoran meliputi variabel yang tidak terpakai, closure, listener acara, dan referensi berulang. Untuk mencegah kebocoran memori, pengembang harus dengan jelas menghapus referensi yang tidak diperlukan, mengelola listener acara, menghindari dependensi berulang, menggunakan referensi lemah, melakukan profil memori, melakukan pengujian dan peninjauan kode, dan menggunakan linter dan alat analisis statis untuk mendeteksi masalah potensial sejak awal dalam proses pengembangan.

6.2. Jelaskan perbedaan antara salinan dangkal dan salinan dalam JavaScript. Bagaimana cara mencapainya?

Jawaban:

Salinan dangkal dan salinan dalam adalah metode untuk menggandakan objek atau array dalam JavaScript.

Salinan dangkal menggandakan struktur paling atas dan nilai-nilai objek atau array tetapi mempertahankan referensi ke objek atau array yang bersarang. Perubahan pada struktur bersarang akan memengaruhi objek asli dan salinannya.

Salinan dalam menciptakan objek atau array baru dan menggandakan semua tingkatan struktur bersarang secara rekursif. Ini memastikan bahwa perubahan pada salinan tidak memengaruhi objek asli.

Untuk mencapai salinan dangkal, kita dapat menggunakan metode seperti operator penyebaran atau slice(). Untuk salinan dalam, kita perlu menggunakan logika k

ustom atau pustaka seperti lodash's cloneDeep karena JavaScript tidak memiliki metode bawaan untuk menggandakan objek secara dalam.

7. ES6 dan Lebih Lanjut

7.1. Jelaskan tujuan dan penggunaan destrukturisasi dalam JavaScript.

Jawaban:

Destrukturisasi dalam JavaScript adalah fitur yang menyederhanakan pengambilan nilai dari objek atau array dengan cara yang lebih konsis dan mudah dibaca. Ini memungkinkan kita untuk mengaitkan nilai-nilai tersebut dengan variabel berdasarkan nama properti atau posisi dalam objek atau array. Destrukturisasi juga dapat digunakan dalam parameter fungsi dan mendukung sintaksis istimewa seperti rest untuk menangkap elemen yang tersisa.

7.2. Jelaskan pentingnya modul JavaScript dan sintaksis `import/export` ES6.

Jawaban:

Modul JavaScript, bersama dengan sintaksis `import/export` ES6, penting untuk pengembangan JavaScript modern. Mereka memungkinkan pengembang untuk mengatur, mengulang, dan mempertahankan kode dengan efektif. Modul mengemas kode yang berkaitan, mempromosikan reusabilitas kode, mengelola dependensi, dan meningkatkan skalabilitas kode. Sintaksis `import/export` ES6 memberikan cara standar untuk mendeklarasikan dan menggunakan modul, membuat lebih mudah mengorganisasi dan berbagi kode dengan cara yang bersih dan mudah dipelihara. Fitur-fitur ini telah menjadi penting dalam membangun aplikasi JavaScript yang modular dan mudah dipelihara, baik di sisi klien maupun server.

7.3. Bagaimana templat literal meningkatkan manipulasi string dalam ES6? Berikan contoh.

Jawaban:

Templat literal dalam ES6 meningkatkan manipulasi string dengan memungkinkan pengembang untuk membuat string dengan ekspresi tersemat dan konten yang melintasi beberapa baris dengan cara yang lebih mudah dibaca dan fleksibel. Mereka mendukung interpolasi variabel, string multi-baris, evaluasi ekspresi, panggilan fungsi, dan bahkan penggunaan lanjutan seperti templat yang diberi tag. Fitur ini meningkatkan keterbacaan dan pemeliharaan kode saat bekerja dengan string kompleks yang melibatkan konten dinamis atau ekspresi.

8. Pemrograman Fungsional

8.1. Bagaimana pemrograman fungsional berbeda dari pemrograman imperatif dalam JavaScript?

Jawaban:

Pemrograman fungsional dan pemrograman imperatif adalah dua paradigma pemrograman dominan.

- **Pemrograman Imperatif**: Paradigma ini berfokus pada memberi tahu komputer "bagaimana" melakukan sesuatu dan mengandalkan pernyataan yang mengubah status program. Ini sering melibatkan penggunaan loop, kondisional, dan pernyataan yang mengubah variabel.
- **Pemrograman Fungsional (FP)**: FP lebih berfokus pada memberi tahu komputer "apa" yang harus dilakukan daripada "bagaimana" melakukannya. Ini memperlakukan tugas komputasi sebagai evaluasi fungsi matematika dan menghindari penggunaan data mutabel dan perubahan status. Dalam konteks JavaScript, fitur-fitur FP termasuk fungsi murni, data yang tidak berubah, dan fungsi tingkat tinggi seperti map dan reduce.

8.2. Jelaskan fungsi kelas pertama dan pentingnya dalam pemrograman fungsional.

Jawaban:

Dalam JavaScript dan banyak bahasa pemrograman lainnya, fungsi dianggap sebagai "warga negara kelas pertama." Ini berarti fungsi dapat:

- Diberikan kepada variabel.
- Diteruskan sebagai argumen ke fungsi lain.
- Dikembalikan dari fungsi lain sebagai nilai.
- Disimpan dalam struktur data seperti array dan objek.

Berikut contoh sederhana yang menunjukkan properti-properti ini:

```
// Memberikan fungsi kepada variabel
const sapa = function(nama) {
  return "Halo, " + nama;
}
// Mengirimkan fungsi sebagai argumen ke fungsi lain
function jalankanFungsi(fn, nilai) {
  return fn(nilai);
}
jalankanFungsi(sapa, 'John'); // Mengembalikan: "Halo, John"
// Mengembalikan fungsi dari fungsi lain
function pengganda(faktor) {
  return function(angka) {
    return angka \* faktor;
  }
}
const duaKali = pengganda(2);
duaKali(5); // Mengembalikan: 10
// Menyimpan fungsi dalam array
const fungsi = [sapa, duaKali];
```

9. Menyimpan Data di Browser

A. Penyimpanan Lokal dan Penyimpanan Sesi

9.1 Apa perbedaan utama antara Penyimpanan Lokal dan Penyimpanan Sesi?

Jawaban:

Penyimpanan Web adalah API web yang menyediakan dua mekanisme untuk menyimpan data di browser web: Penyimpanan Lokal dan Penyimpanan Sesi. Perbedaan utamanya adalah:

- Masa Hidup: Data Penyimpanan Lokal tetap ada bahkan setelah browser ditutup, sementara data Penyimpanan Sesi hanya tersedia selama sesi halaman berlangsung.
- Ruang Lingkup: Data Penyimpanan Lokal dapat diakses di berbagai jendela dan tab dari asal yang sama, sedangkan data Penyimpanan Sesi terbatas pada halaman atau tab saat ini.
- Batasan Penyimpanan: Penyimpanan Lokal biasanya memiliki batasan penyimpanan yang lebih besar (sekitar 5-10 MB) dibandingkan dengan Penyimpanan Sesi (sekitar 5-10 MB juga).

9.2 Bagaimana cara menyimpan data di Penyimpanan Lokal dan Penyimpanan Sesi menggunakan JavaScript?

Jawaban:

Anda dapat menggunakan objek localStorage dan sessionStorage untuk menyimpan data. Berikut contoh menyimpan data di Penyimpanan Lokal:

`localStorage.setItem('username', 'JohnDoe');`

Untuk menyimpan data di Penyimpanan Sesi, gantilah localStorage dengan `sessionStorage.`

9.3 Bagaimana cara menghapus atau menghapus data dari Penyimpanan Lokal dan Penyimpanan Sesi?

Jawaban:

Anda dapat menghapus item dari penyimpanan menggunakan metode removeItem. Untuk menghapus semua item, Anda dapat menggunakan metode clear. Contohnya:

Hapus item : `localStorage.removeItem('username');`

Hapus semua item : `localStorage.clear();`

9.4 Jelaskan kekhawatiran keamanan yang terkait dengan Penyimpanan Web.

Jawaban:

Penyimpanan Web bersifat domain-spesifik, artinya data hanya dapat diakses dari domain yang sama yang menyimpannya. Namun, ada kekhawatiran keamanan terkait dengan menyimpan informasi sensitif di Penyimpanan Web. Data tidak dienkripsi, dan rentan terhadap serangan cross-site scripting (XSS), di mana skrip jahat dapat mengakses dan memodifikasi data yang disimpan.

B. IndexDB

IndexedDB bisa dianggap sebagai "localStorage versi canggih". Ini adalah database berbasis kunci-nilai sederhana, cukup kuat untuk aplikasi offline, namun mudah digunakan.

9.5 Apa itu IndexDB, dan bagaimana perbedaannya dengan Penyimpanan Web (Penyimpanan Lokal dan Penyimpanan Sesi)?

Jawaban:

IndexedDB adalah database berbasis JavaScript tingkat rendah untuk menyimpan jumlah data yang besar. Ini berbeda dari Penyimpanan Web dalam beberapa hal:

- Struktur Data: IndexedDB menyimpan data terstruktur, sementara Penyimpanan Web menyimpan pasangan kunci-nilai.
- Batasan Penyimpanan: IndexedDB biasanya menawarkan batasan penyimpanan yang lebih besar (seringkali dalam megabita) dibandingkan dengan penyimpanan terbatas Penyimpanan Web.
- Kompleksitas API: IndexedDB memiliki API yang lebih kompleks, mengharuskan pengembang untuk menentukan skema database dan bekerja dengan transaksi.

9.6 Bagaimana cara membuka database dan membuat toko objek di IndexedDB menggunakan JavaScript?

Jawaban:

Anda dapat membuka database dan membuat toko objek seperti ini:

Buka database (atau buat jika belum ada) :

```
const request = indexedDB.open('myDatabase', 1);
```

Buat toko objek :

```
request.onupgradeneeded = (event) => {
  const db = event.target.result;
  db.createObjectStore('myStore', { keyPath: 'id' });
};
```

Bab 23

Pola Desain

Pola desain adalah solusi berorientasi objek yang dapat Anda implementasikan untuk mengatasi masalah pemrograman umum yang mungkin muncul selama proses pengembangan. Pola desain bukanlah hal yang sama dengan algoritma, melainkan konsep pemrograman yang dapat Anda gunakan untuk menyelesaikan jenis masalah tertentu.

Ada total 23 pola desain, dan ini tidak unik untuk bahasa tertentu. Ini adalah konsep dasar yang dapat diterapkan dalam berbagai bahasa pemrograman. Hari ini, kita akan belajar tentang setiap pola desain dan bagaimana mengimplementasikannya menggunakan JavaScript. Pola desain dapat digolongkan ke dalam salah satu dari tiga kategori berikut:

- Pola Penciptaan
- Pola Struktural
- Pola Perilaku

Pola Desain Kreasional

Pola desain kreasional berfokus pada mekanisme penciptaan objek

1. Metode Pabrik (Factory Method)

Sebuah fungsi pabrik hanyalah sebuah fungsi yang menciptakan objek dan mengembalikannya. Ini adalah pola desain kreasional yang memungkinkan Anda untuk membuat objek tanpa menentukan kelas atau konstruktor yang tepat untuk digunakan. Ini mengkonsolidasikan logika penciptaan objek, memungkinkan fleksibilitas dalam penciptaan berbagai jenis objek. Katakanlah Anda memiliki situs web dan ingin membuat metode yang memungkinkan Anda dengan mudah membuat objek HTML dan menambahkannya ke DOM. Pabrik adalah solusi yang sempurna untuk ini, dan berikut adalah bagaimana kita dapat mengimplementasikannya.

1.1. Komponen Metode Pabrik

Pencipta (Creator)

Ini adalah metode yang diimplementasikan dalam Pabrik yang menciptakan produk baru.

Produk Abstrak (Abstract Product)

Antarmuka untuk produk yang sedang dibuat.

Produk Konkret (Concrete Product)

Ini adalah objek sebenarnya yang sedang dibuat.

1.2. Manfaat Metode Pabrik

Abstraksi Penciptaan Objek

Ini menghilangkan kompleksitas penciptaan objek, memungkinkan kode klien hanya fokus pada objek yang dibuat.

Fleksibilitas dan Kustomisasi

Pabrik memungkinkan kustomisasi proses penciptaan objek, memungkinkan variasi dalam objek yang dibuat.

Enkapsulasi Logika Penciptaan

Logika penciptaan terenkapsulasi dalam pabrik, membuatnya lebih mudah dimodifikasi atau diperluas tanpa memengaruhi kode klien.

Penciptaan Objek yang Kompleks

Pabrik berguna ketika penciptaan objek rumit, melibatkan beberapa langkah, atau memerlukan kondisi tertentu.

1.3. Contoh

```
function elementFactory(type, text, color) {
  const newElement = document.createElement(type);
  newElement.innerText = text;
  newElement.style.color = color;
  document.body.append(newElement);

  function setText(newText) {
    newElement.innerText = newText;
  }

  function setColor(newColor) {
    newElement.innerText = newColor;
  }

  return {
    newElement,
    setText,
    setColor,
  };
}

const h1Tag = elementFactory("h1", "Teks Awal", "Biru");

h1Tag.setText("Halo dunia");

h1Tag.setColor("Merah");
```

2. Metode Pabrik Abstrak

Pabrik abstrak adalah pola desain pembuatan lainnya. Tujuannya utama adalah menyediakan antarmuka untuk menciptakan keluarga objek terkait atau bergantung tanpa menentukan kelas konkret mereka. Pola ini memastikan bahwa objek yang diciptakan kompatibel dan bekerja bersama.

2.1. 4 Komponen dari Pabrik Abstrak

Pabrik Abstrak

Ini mendefinisikan antarmuka untuk membuat produk abstrak, yang merupakan keluarga objek terkait (misalnya, komponen UI). Pabrik abstrak mendeklarasikan metode penciptaan untuk setiap jenis produk dalam keluarga.

Pabrik Konkrit

Ini adalah kelas yang mengimplementasikan antarmuka pabrik abstrak, menyediakan implementasi khusus untuk menciptakan produk konkret. Setiap pabrik konkret menciptakan keluarga produk terkait (misalnya, pabrik UI mungkin menciptakan tombol atau kotak centang).

Produk Abstrak

Ini adalah antarmuka atau kelas dasar untuk produk yang dibuat oleh pabrik abstrak. Setiap jenis produk dalam keluarga memiliki definisi produk abstraknya sendiri (misalnya, Tombol, Kotak Centang).

Produk Konkrit

Ini adalah implementasi aktual dari produk abstrak. Setiap pabrik konkret menciptakan seperangkat produk konkretnya sendiri. Produk konkrit mengimplementasikan antarmuka produk abstrak yang didefinisikan untuk keluarganya (misalnya, HTMLButton, WindowsButton).

2.2. Manfaat dari Pabrik Abstrak

Konsistensi

Ini memastikan bahwa objek yang dibuat kompatibel dan mengikuti tema atau gaya yang konsisten.

Pemisahan Tanggung Jawab

Ini mengisolasi penciptaan objek dari kode klien, mempromosikan pemisahan yang bersih antara perhatian.

Fleksibilitas dan Skalabilitas

Ini memungkinkan penambahan mudah keluarga produk baru tanpa mengubah kode klien yang ada.

2.3. Contoh

```
// Pabrik Abstrak: UIFactory
class UIFactory {
    createButton() {
        throw new Error('metode createButton harus di-override');
    }

    createCheckbox() {
        throw new Error('metode createCheckbox harus di-override');
    }
}

// Pabrik Konkret: WindowsUIFactory
class WindowsUIFactory extends UIFactory {
    createButton() {
        return new WindowsButton();
    }

    createCheckbox() {
        return new WindowsCheckbox();
    }
}

// Pabrik Konkret: MacUIFactory
class MacUIFactory extends UIFactory {
    createButton() {
        return new MacButton();
    }

    createCheckbox() {
        return new MacCheckbox();
    }
}

// Produk Abstrak: Tombol (Button)
class Button {
    render() {
        throw an Error('metode render harus di-override');
    }
}

// Produk Konkret: WindowsButton
class WindowsButton extends Button {
    render() {
        console.log('Merender tombol Windows');
    }
}

// Produk Konkret: MacButton
class MacButton extends Button {
    render() {
        console.log('Merender tombol Mac');
    }
}

// Produk Abstrak: Checkbox
class Checkbox {
    render() {
        throw new Error('metode render harus di-override');
    }
}

// Produk Konkret: WindowsCheckbox
class WindowsCheckbox extends Checkbox {
    render() {
        console.log('Merender checkbox Windows');
    }
}
```

```
// Produk Konkret: MacCheckbox
class MacCheckbox extends Checkbox {
    render() {
        console.log('Merender checkbox Mac');
    }
}

// Penggunaan
const windowsFactory = new WindowsUIFactory();
const macFactory = new MacUIFactory();

const windowsButton = windowsFactory.createButton();
windowsButton.render();  // Output: Merender tombol Windows

const macCheckbox = macFactory.createCheckbox();
macCheckbox.render();  // Output: Merender checkbox Mac
```

3. Pembangun (Builder)

Tujuan dari pembangun adalah untuk memisahkan konstruksi objek dari representasinya. Apa yang dilakukan pola pembangun adalah memungkinkan klien untuk membangun objek kompleks hanya dengan melewati jenis dan konten objek. Klien tidak perlu khawatir tentang detail konstruksi.

3.1. 4 Komponen dari Pembangun

Pembangun

Biasanya berisi serangkaian metode untuk membangun berbagai bagian objek.

Pembangun Konkrit

Mengimplementasikan metode dari antarmuka pembangun untuk membangun bagian-bagian objek.

Direktur (Opsional)

Ini tidak selalu diperlukan tetapi dapat membantu dalam membangun objek akhir menggunakan proses konstruksi tertentu.

Objek

Representasi produk akhir. Berisi bagian-bagian yang dibangun oleh pembangun.

3.2. Manfaat dari Pola Pembangun

Pemisahan Perhatian

Pola Pembangun memisahkan konstruksi objek kompleks dari representasinya, memungkinkan berbagai implementasi pembangun untuk bervariasi dalam representasi internal.

Penciptaan Objek yang Fleksibel

Ini memungkinkan penciptaan konfigurasi yang berbeda dari objek kompleks dengan menggunakan proses konstruksi yang umum. Pembangun dapat disesuaikan untuk menciptakan variasi objek tertentu.

Meningkatkan Keterbacaan

Menggunakan pembangun dapat meningkatkan keterbacaan kode dengan jelas menguraikan langkah-langkah yang diperlukan untuk membangun objek. Mudah memahami kontribusi setiap langkah pada objek akhir.

Konstruksi Berparameter

Pembangun memungkinkan Anda untuk membangun objek dengan melewati parameter ke langkah-langkah konstruksi, memberikan kontrol terperinci atas penciptaan dan konfigurasi objek.

Penggunaan Kembali

Pembangun dapat digunakan kembali untuk membuat beberapa instance objek kompleks dengan konfigurasi yang berbeda, mempromosikan penggunaan ulang kode dan meminimalkan duplikasi logika konstruksi.

3.3. Contoh

```
// Pembangun
class ComputerBuilder {
    buildCPU() {}
    buildRAM() {}
    buildStorage() {}
    getResult() {}
}

// Pembangun Konkret
class GamingComputerBuilder extends ComputerBuilder {
    // Implementasikan langkah-langkah khusus untuk membangun komputer gaming
}

class OfficeComputerBuilder extends ComputerBuilder {
    // Implementasikan langkah-langkah khusus untuk membangun komputer kantor
}

// Kelas Objek
class Computer {
    constructor() {
        this.parts = [];
    }

    addPart(part) {
        this.parts.push(part);
    }
}

// Direktur (Opsional)
class ComputerAssembler {
    constructor(builder) {
        this.builder = builder;
    }

    assembleComputer() {
        this.builder.buildCPU();
        this.builder.buildRAM();
        this.builder.buildStorage();
        return this.builder.getResult();
    }
}
```

4. Singleton

Sebuah singleton adalah objek yang hanya bisa diinstansiasi sekali. Singleton berguna ketika tindakan sistem secara menyeluruh perlu diselaraskan dari satu tempat pusat tunggal. Singleton mengurangi kebutuhan untuk variabel global, yang sangat penting dalam JavaScript karena membatasi polusi ruang nama.

4.1. Komponen dari Singleton

Fungsi Anonim

Sebuah singleton diimplementasikan menggunakan fungsi anonim.

Fungsi getInstance

Ini adalah fungsi yang mengembalikan objek yang diinstansiasi secara unik.

Konstruktor (Opsional)

Dalam JavaScript, konstruktor tidak diperlukan untuk mengimplementasikan pola singleton, tetapi memiliki konstruktor umum karena memungkinkan Anda untuk mengkonfigurasi singleton dan menambahkan logika inisialisasi.

4.2. Manfaat dari Singleton

Mengurangi Variabel Global

Singleton dapat membantu mengurangi jumlah variabel global yang diperlukan dalam program Anda, mempromosikan organisasi kode yang lebih baik dan pemeliharaan.

Efisien Memori

Karena Singleton memastikan hanya ada satu instans yang ada pada satu waktu, memori disimpan karena Anda menghindari memiliki beberapa instans dari kelas yang sama.

Akses Global

Singleton menyediakan titik akses global ke instans. Ini memungkinkan bagian lain dari program untuk mengakses dan menggunakan instans yang sama tanpa perlu meneruskannya.

Berbagi Sumber Daya

Singleton sangat berguna ketika menangani tugas seperti mengelola sumber daya bersama. Singleton dapat digunakan untuk mengelola koneksi basis data, penangan berkas, dan bahkan grup benang, memastikan bahwa sumber daya ini dibagikan secara efisien di seluruh aplikasi.

4.3. Contoh

```
class Singleton {
  constructor() {
    const privateVariable = "Ini adalah variabel pribadi";

    function privateMethod() {
      console.log("Ini adalah metode pribadi.");
    }

    return {
      publicMethod: function () {
        console.log("Ini adalah metode publik.");
      },
      publicVariable: "Saya adalah publik",
    };
  }

  static getInstance() {
    if (!Singleton.instance) {
      Singleton.instance = new Singleton();
    }
    return Singleton.instance;
  }
}

const singletonInstance1 = Singleton.getInstance();
const singletonInstance2 = Singleton.getInstance();

console.log(singletonInstance1 === singletonInstance2); // Menghasilkan: true
```

5. Prototype

Pola prototipe adalah cara alternatif untuk mengimplementasikan warisan, tetapi perbedaan utamanya adalah daripada mewarisi properti dari sebuah kelas, objek mewarisi properti dari objek prototipe. Pola prototipe juga disebut pola properti, dan JavaScript mendukung prototipe secara alami. Dalam JavaScript, setiap objek memiliki prototipe (referensi ke objek lain). Ketika Anda mencoba mengakses properti yang tidak ada dalam objek itu sendiri, JavaScript akan mencarinya dalam prototipe objek dan terus naik ke prototipe hingga menemukannya atau mencapai ujung rantai prototipe.

5.1. Komponen dari Pola Prototipe

Objek Prototipe

Mengandung properti dan metode yang akan diwarisi oleh semua instansi baru.

Klien

Klien bertanggung jawab untuk membuat objek baru berdasarkan prototipe. Klien dapat membuat instansi baru berdasarkan prototipe dan memodifikasi properti-properti mereka sesuai kebutuhan.

Mekanisme Klona/Pembuatan

Mekanisme yang digunakan untuk membuat objek baru berdasarkan prototipe. Dalam JavaScript, ini dapat dicapai dengan menggunakan fungsi `Object.create()`.

5.2. Manfaat dari Pola Prototipe

Penciptaan Instansi yang Efisien

Membuat instansi baru dari Prototipe lebih efisien daripada menggunakan kelas dan konstruktor tradisional. Objek dibuat dengan mengklon prototipe, yang mengurangi kebutuhan untuk mengatur kelas dan logika inisialisasi.

Penggunaan Kode Kembali

Pola Prototipe memungkinkan Anda untuk mendefinisikan seperangkat properti dan metode default dalam objek prototipe. Ini memungkinkan beberapa instansi untuk berbagi perilaku dan struktur yang sama tanpa menggandakan kode. Ini juga mengurangi penggunaan memori karena setiap instansi tidak perlu menyimpan salinan properti prototipe.

Penciptaan Objek yang Fleksibel

Objek yang dibuat menggunakan Pola Prototipe dapat dengan mudah disesuaikan dengan memodifikasi properti mereka atau menambahkan properti baru yang khusus untuk instansi.

Perubahan Runtime yang Dinamis

Perubahan yang dibuat pada objek prototipe selama runtime tercermin dalam semua instansi yang didasarkan pada prototipe. Perilaku ini memungkinkan pembaruan dan modifikasi pada prototipe, yang memengaruhi semua instansi yang berbagi prototipe yang sama.

5.3. Contoh

```
const cameraPrototype = {
  model: "default",
  make: "default",
  shutter: function () {
    console.log(`Kamera ${this.make} ${this.model} telah mengambil foto`);
  },
};

const camera1 = Object.create(cameraPrototype);
camera1.model = "X-Pro 3";
camera1.make = "Fujifilm";

const camera2 = Object.create(cameraPrototype);
camera2.model = "R5";
camera2.make = "Canon";
```

Pola Desain Struktural

Fokus pada bagaimana kelas dan objek disusun untuk membentuk struktur yang lebih besar

1. Adapter

Adapter adalah pola desain struktural yang memungkinkan Anda untuk membuat antarmuka yang berbeda dengan metode yang berbeda bekerja bersama tanpa mengubah kode mereka. Tujuan dari Adapter adalah membuat dua antarmuka yang tidak kompatibel dapat bekerja bersama dengan lancar.

1.1 Komponen dari Adapter

Antarmuka/Kelas Target

Ini adalah antarmuka atau kelas yang akan digunakan oleh klien. Ini berisi semua metode dan properti yang akan digunakan oleh kode klien.

Adaptee

Adaptee adalah antarmuka/kelas lama yang berisi properti dan metode yang tidak kompatibel dengan antarmuka/kelas baru.

Adapter

Adapter adalah yang menghubungkan kesenjangan antara Adaptee dan antarmuka/kelas Target.

1.2 Manfaat dari Adapter

Integrasi yang Mudah

Adapter menciptakan cara yang mudah bagi kode atau sistem baru untuk berinteraksi dengan yang sudah ada. Dengan menggunakan Adapter, integrasi kode baru menjadi lebih lancar dan kurang rentan terhadap kesalahan.

Kompatibilitas dan Daur Ulang

Adapter mendorong penggunaan kembali kode dan memperluas kegunaan kode yang sudah ada dengan membuat kode lama kompatibel dengan kode baru.

Integrasi Sistem Bertahap

Dalam situasi di mana sistem baru perlu diimplementasikan secara bertahap, Adapter dapat berfungsi sebagai perantara, memungkinkan fitur-fitur baru masuk secara perlahan sambil tetap menjaga kompatibilitas dengan sistem yang sudah ada.

Peningkatan Uji Coba

Adapter memudahkan pengujian dengan memungkinkan pemalsuan atau penggantian Adaptee selama pengujian kode klien. Ini meningkatkan uji coba kode klien dan membantu dalam menulis pengujian unit yang lebih mudah dimengerti.

1.3 Contoh

```
// Adaptee: Pengisi daya EU
class PengisiDayaEU {
  chargeWithEUPlug() {
    console.log("Mengisi daya dengan steker EU");
  }
}

// Adaptee: Pengisi daya AS
class PengisiDayaAS {
  chargeWithUSPlug() {
    console.log("Mengisi daya dengan steker AS");
  }
}

// Target: Antarmuka pengisian daya yang diharapkan oleh klien
class AntarmukaPengisianDaya {
  charge() {
    console.log("Mengisi daya...");
  }
}

// Adapter untuk pengisi daya EU
class AdapterPengisiDayaEU extends AntarmukaPengisianDaya {
  constructor(pengisiDayaEU) {
    super();
    this.pengisiDayaEU = pengisiDayaEU;
  }

  charge() {
    this.pengisiDayaEU.chargeWithEUPlug();
  }
}

// Adapter untuk pengisi daya AS
class AdapterPengisiDayaAS extends AntarmukaPengisianDaya {
  constructor(pengisiDayaAS) {
    super();
    this.pengisiDayaAS = pengisiDayaAS;
  }

  charge() {
    this.pengisiDayaAS.chargeWithUSPlug();
  }
}

// Klien
function mengisiPerangkat(antarmukaPengisianDaya) {
  antarmukaPengisianDaya.charge();
}

// Penggunaan
const pengisiDayaEU = new PengisiDayaEU();
const adapterEU = new AdapterPengisiDayaEU(pengisiDayaEU);

const pengisiDayaAS = new PengisiDayaAS();
const adapterAS = new AdapterPengisiDayaAS(pengisiDayaAS);

console.log("Mengisi daya dengan pengisi daya EU:");
mengisiPerangkat(adapterEU);

console.log("Mengisi daya dengan pengisi daya AS:");
mengisiPerangkat(adapterAS);
```

2. Bridge (Jembatan)

Jembatan adalah pola desain struktural yang dirancang untuk memisahkan kelas yang sangat besar menjadi dua hierarki terpisah yang dapat dikembangkan secara independen. Kedua hierarki ini disebut sebagai tingkat Abstraksi (Abstraction) dan tingkat Implementasi (Implementation). Pada dasarnya, jika Anda memiliki kelas yang memiliki variasi dari beberapa fungsionalitas, Anda dapat menggunakan pola Jembatan untuk membagi dan mengorganisasi kelas menjadi dua hierarki yang lebih mudah dipahami.

2.1. Komponen dari Jembatan

Abstraksi

Ini adalah antarmuka atau abstraksi tingkat tinggi. Ini mendefinisikan fungsionalitas abstrak yang akan digunakan oleh klien.

Abstraksi Yang Diperinci

Ini adalah subclass atau perluasan dari tingkat abstraksi. Ini memberikan fitur tambahan atau penyempurnaan. Ini memperluas fungsionalitas yang didefinisikan oleh abstraksi.

Implementor

Ini adalah antarmuka yang mendefinisikan metode implementasi, biasanya tidak mencerminkan antarmuka abstraksi, tetapi merupakan antarmuka tingkat lebih rendah yang digunakan oleh abstraksi.

Implementor Konkret

Kelas konkret yang mengimplementasikan antarmuka implementor. Kelas-kelas ini memberikan implementasi spesifik dari metode yang didefinisikan oleh antarmuka implementor.

2.2. Manfaat dari Pola Jembatan

Mengurai Abstraksi dari Implementasi

Manfaat utama dari pola Jembatan adalah memisahkan tingkat abstraksi dari tingkat implementasi. Ini memungkinkan kedua bagian tersebut untuk berkembang secara independen, membuat basis kode lebih mudah dimodifikasi.

Meningkatkan Keterjagaan

Karena basis kode dibagi menjadi dua bagian, membuat perubahan pada salah satu bagian sistem kemungkinan besar tidak akan berdampak pada bagian lain. Ini membuat pemeliharaan basis kode lebih mudah dan efisien.

Meningkatkan Pengujian

Pengujian menjadi lebih mudah saat Anda memiliki pola jembatan dalam basis kode Anda karena Anda dapat fokus pada pengujian tingkat abstraksi secara terpisah dari pengujian tingkat implementasi. Ini memungkinkan pengujian yang lebih mudah dan lebih terfokus.

Meningkatkan Keterbacaan

Pola Jembatan menciptakan hierarki yang jelas dalam basis kode. Mengorganisasi basis kode dengan cara ini membantu memahami hubungan antara berbagai bagian sistem.

2.3. Contoh

```
// Abstraksi
class Bentuk {
  constructor(warna) {
    this.warna = warna;
  }

  gambar() {
    console.log(`Menggambar bentuk dengan warna ${this.warna}`);
  }
}

// Implementasi
class WarnaMerah {
  aplikasiWarna() {
    console.log("Menerapkan warna merah");
  }
}

class WarnaBiru {
  aplikasiWarna() {
    console.log("Menerapkan warna biru");
  }
}

// Jembatan (Bridge)
class BentukDenganWarna extends Bentuk {
  constructor(warna, implementasiWarna) {
    super(warna);
    this.implementasiWarna = implementasiWarna;
  }

  gambar() {
    super.gambar();
    this.implementasiWarna.aplikasiWarna();
  }
}

// Penggunaan
const bentukMerah = new BentukDenganWarna("merah", new WarnaMerah());
const bentukBiru = new BentukDenganWarna("biru", new WarnaBiru());

bentukMerah.gambar(); // Output: Menggambar bentuk dengan warna merah, Menerapkan warna merah
bentukBiru.gambar(); // Output: Menggambar bentuk dengan warna biru, Menerapkan warna biru
```

3. Komposit (Composite)

Pola desain komposit memungkinkan pembuatan objek dengan properti yang merupakan item primitif atau kumpulan objek. Bayangkan struktur seperti pohon, di mana Anda memiliki objek tunggal (simpul daun) atau kelompok objek (cabang). Pola desain komposit memungkinkan Anda membuat jenis struktur ini dan dapat melakukan operasi pada setiap tingkatan dengan cara yang konsisten.

3.1 Komponen Komposit

Komponen

Ini adalah antarmuka/kelas yang mewakili baik simpul daun (elemen individu) maupun simpul komposit (kumpulan elemen). Komponen mendefinisikan operasi yang dapat diterapkan pada kedua jenis simpul.

Daun (Leaf)

Ini mewakili objek individu dalam pohon yang tidak memiliki anak. Mereka mengimplementasikan operasi yang didefinisikan dalam antarmuka komponen.

Komposit (Composite)

Ini mewakili komposit atau kontainer yang dapat menampung koleksi simpul daun atau simpul komposit lainnya.

3.2 Manfaat Komposit

Keseragaman dan Konsistensi

Pola desain Komposit menyediakan cara yang seragam untuk memperlakukan baik objek individu maupun komposisi objek. Klien memiliki satu antarmuka umum untuk mengoperasikan objek-objek ini yang menyederhanakan basis kode dan interaksi objek.

Fleksibilitas

Pola desain ini memungkinkan fleksibilitas dalam menambahkan jenis komponen baru atau memodifikasi yang sudah ada tanpa memengaruhi kode klien. Anda dapat dengan mudah memperkenalkan jenis objek daun atau komposit yang baru.

Sederhana Kode Klien

Kode klien tidak perlu membedakan antara komponen individu dan komposit, sehingga membuatnya lebih sederhana dan intuitif untuk bekerja dengan struktur ini.

3.3 Contoh

```
class SatuBlok {
  constructor(nama) {
    this.nama = nama;
  }

  tampilkan() {
    console.log("Blok:", this.nama);
  }
}

class KoleksiBlok {
  constructor(nama) {
    this.nama = nama;
    this.blok = [];
  }

  tambah(blok) {
    this.blok.push(blok);
  }

  hapus(blok) {
    this.blok = this.blok.filter((b) => b !== blok);
  }

  tampilkan() {
    console.log("Koleksi Blok:", this.nama);

    for (const blok of this.blok) {
      blok.tampilkan();
    }
  }
}

// Penggunaan
const blok1 = new SatuBlok("Blok 1");
const blok2 = new SatuBlok("Blok 2");
const blok3 = new SatuBlok("Blok 3");

const grupBlok1 = new KoleksiBlok("Grup Blok 1");
grupBlok1.tambah(blok1);
grupBlok1.tambah(blok2);

const grupBlok2 = new KoleksiBlok("Grup Blok 2");
grupBlok2.tambah(blok3);

const megaStruktur = new KoleksiBlok("Mega Struktur");
megaStruktur.tambah(grupBlok1);
megaStruktur.tambah(grupBlok2);

megaStruktur.tampilkan();
```

4. Dekorator (Decorator)

Pola desain Dekorator dapat digunakan untuk memodifikasi perilaku objek baik secara statis maupun dinamis tanpa memengaruhi perilaku objek lain dari kelas yang sama. Dekorator sangat berguna ketika Anda ingin menambahkan fitur ke objek secara fleksibel dan dapat digunakan ulang.

4.1 Komponen Dekorator

Antarmuka Komponen (Component Interface)

Ini mendefinisikan logika untuk objek yang dapat memiliki tanggung jawab ditambahkan secara dinamis.

Komponen Konkret (Concrete Components)

Ini adalah objek awal yang dapat ditambahkan fungsionalitas tambahan.

Dekorator (Decorator)

Ini adalah antarmuka yang memperluas fungsionalitas komponen konkret. Ini memiliki referensi ke instansi komponen dan mengimplementasikan antarmuka komponen.

Dekorator Konkret (Concrete Decorators)

Ini adalah implementasi konkret dari kelas dekorator, yang menambahkan perilaku tertentu ke komponen yang diinginkan dengan memperluas kelas dekorator.

4.2 Manfaat Dekorator

Ekstensibilitas dan Fleksibilitas

Dekorator memungkinkan Anda untuk menambahkan fungsionalitas atau perilaku baru ke objek secara dinamis saat runtime. Ini mempromosikan ekstensibilitas tanpa mengubah basis kode yang ada dan memberikan fleksibilitas dalam cara Anda dapat menggabungkan dan menggunakan fungsionalitas tambahan ini.

Modularitas

Dekorator memungkinkan pendekatan kode yang lebih modular dengan memecah fungsionalitas menjadi unit yang lebih kecil dan mudah dikelola. Unit-unit ini dapat digabungkan dan digunakan kembali dengan berbagai cara.

Konfigurasi Saat Runtime

Dekorator memungkinkan Anda untuk mengonfigurasi objek secara dinamis saat runtime. Ini memungkinkan Anda untuk menambahkan atau menghapus fungsionalitas tanpa memengaruhi komponen inti atau perlu kompilasi ulang kode.

Mengurangi Pembuatan Subkelas (Subclassing)

Tanpa Dekorator, memperluas fungsionalitas sering melibatkan pembuatan banyak subclass untuk setiap kombinasi perilaku. Dekorator menghilangkan kebutuhan untuk subclass, yang menghasilkan basis kode yang lebih bersih dan lebih mudah dipahami.

4.3 Contoh

```
class Kopi {
  biaya() {
    return 5;
  }
}

class DekoratorSusu {
  constructor(kopi) {
    this.kopi = kopi;
  }

  biaya() {
    return this.kopi.biaya() + 2;
  }
}

class DekoratorGula {
  constructor(kopi) {
    this.kopi = kopi;
  }

  biaya() {
    return this.kopi.biaya() + 1;
  }
}

// Penggunaan
let kopi = new Kopi();
console.log("Biaya kopi biasa:", kopi.biaya());

let susuKopi = new DekoratorSusu(kopi);
console.log("Biaya kopi susu:", susuKopi.biaya());

let gulaSusuKopi = new DekoratorGula(susuKopi);
console.log("Biaya kopi susu gula:", gulaSusuKopi.biaya());
```

5. Fasad (Facade)

Pola desain Fasad adalah antarmuka yang disederhanakan yang dapat diakses oleh klien untuk menggunakan operasi-operasi level rendah lainnya yang disembunyikan di tempat lain dalam basis kode. Pola desain ini sering digunakan dalam sistem yang dibangun di sekitar arsitektur multi-lapisan. Fasad memungkinkan klien untuk melakukan tugas-tugas tertentu tanpa perlu memahami kompleksitas dari sistem yang mendasarinya.

5.1 Komponen Fasad

Fasad (Facade)

Fasad adalah antarmuka yang akan diakses oleh klien. Ini menyediakan antarmuka yang sederhana dan terpadu yang mendelegasikan permintaan klien ke objek yang sesuai dalam subsistem.

Subsisitem (Subsystem)

Subsistem terdiri dari semua komponen dan fungsionalitas beragam yang dibungkus oleh Fasad. Subsistem dan Fasad berinteraksi satu sama lain tetapi beroperasi secara independen.

5.2 Manfaat Fasad

Antarmuka yang Disederhanakan

Fasad menyediakan antarmuka yang sederhana dan mudah dipahami.

Organisasi Kode

Fasad membantu mengorganisasi kode dengan mengenkapsulasi fungsionalitas subsistem dan memberikan pemisahan yang jelas terhadap perhatian.

Pemeliharaan yang Lebih Mudah

Perubahan pada subsistem dapat diisolasi dalam Fasad, mengurangi dampak pada kode klien.

5.3 Contoh

```
// Subsistem Plumbing
class SubsistemPlumbing {
  constructor() {}

  nyalakanAir() {
    console.log("Plumbing: Air dinyalakan");
  }

  matikanAir() {
    console.log("Plumbing: Air dimatikan");
  }
}

// Subsistem Electrical
class SubsistemElectrical {
  constructor() {}

  nyalakanListrik() {
    console.log("Electrical: Listrik dinyalakan");
  }

  matikanListrik() {
    console.log("Electrical: Listrik dimatikan");
  }
}

// Fasad Rumah
class FasadRumah {
  constructor() {
    this.subsistemPlumbing = new SubsistemPlumbing();
    this.subsistemElectrical = new SubsistemElectrical();
  }

  masukRumah() {
    this.subsistemPlumbing.nyalakanAir();
    this.subsistemElectrical.nyalakanListrik();
    console.log("Anda telah memasuki rumah.");
  }

  keluarRumah() {
    this.subsistemPlumbing.matikanAir();
    this.subsistemElectrical.matikanListrik();
    console.log("Anda telah keluar dari rumah.");
  }
}

// Klien
const klien = () => {
  const rumah = new FasadRumah();

  // Masuk rumah
  rumah.masukRumah();

  // Keluar rumah
  rumah.keluarRumah();
};

// Jalankan klien
klien();
```

6. Pola Flyweight

Pola desain Flyweight bertujuan untuk meminimalkan penggunaan memori atau biaya komputasi dengan menyimpan nilai intrinsik (properti serupa) dari objek serupa dalam aplikasi, mengurangi jumlah kode duplikat. Ini sangat berguna ketika berurusan dengan sejumlah besar objek serupa dalam aplikasi.

6.1. Komponen Pola Flyweight

Pabrik Flyweight (FlyweightFactory)

Pabrik flyweight membuat objek flyweight. Ini berisi fungsi yang akan membuat flyweight jika flyweight tersebut belum ada, dan menyimpan flyweight yang baru dibuat untuk permintaan di masa depan.

Flyweight

Flyweight berisi data intrinsik yang akan dibagikan di seluruh aplikasi.

6.2. Manfaat Pola Flyweights

Efisiensi Memori

Dengan berbagi data intrinsik di antara beberapa objek, pola Flyweight secara signifikan mengurangi penggunaan memori, terutama ketika berurusan dengan sejumlah besar instans.

Peningkatan Kinerja

Dengan penggunaan memori yang lebih rendah, kinerja keseluruhan aplikasi meningkat. Penggunaan memori yang lebih rendah biasanya mengarah pada waktu eksekusi yang lebih cepat dan kinerja aplikasi yang lebih lancar.

Mempermudah Manajemen Status

Dengan memisahkan data intrinsik (nilai bersama) dan data ekstrinsik (nilai unik), Flyweights mempermudah manajemen status. Ini memungkinkan pemisahan perhatian yang lebih bersih dan pendekatan yang lebih terorganisir dalam penanganan status.

6.3. Contoh

```
// Objek Flyweight untuk Kamera
function Kamera(make, model, resolusi) {
  this.make = make;
  this.model = model;
  this.resolution = resolusi;
}

// Pabrik Flyweight untuk Kamera
var PabrikKameraFlyweight = (function () {
  var flyweights = {};

  return {
    get: function (make, model, resolusi) {
      if (!flyweights[make + model]) {
        flyweights[make + model] = new Kamera(make, model, resolusi);
      }
      return flyweights[make + model];
    },

    getCount: function () {
      var count = 0;
      for (var f in flyweights) count++;
      return count;
    },
  };
})();

// Koleksi Kamera
function KoleksiKamera() {
  var cameras = {};
  var count = 0;

  return {
    add: function (make, model, resolusi, serial) {
      cameras[serial] = {
        flyweight: PabrikKameraFlyweight.get(make, model, resolusi),
        serial: serial,
      };
      count++;
    },

    get: function (serial) {
      return cameras[serial];
    },

    getCount: function () {
      return count;
    },
  };
}

// Menjalankan contoh
function jalankan() {
  var cameras = new KoleksiKamera();

  cameras.add("Canon", "EOS R5", "45MP", "A1234");
  cameras.add("Nikon", "D850", "45.7MP", "B5678");
  cameras add ("Sony", "A7R III", "42.4MP", "C9101");
  cameras add ("Canon", "EOS R5", "45MP", "D1212"); // Menggunakan kembali flyweight yang ada

  console.log("Kamera: " + cameras.getCount());
  console.log("Flyweights: " + PabrikKameraFlyweight.getCount());
}

// Jalankan contoh
jalankan();
```

7. Proxy

Pola desain Proxy adalah pola desain struktural yang memungkinkan Anda untuk menyediakan pengganti atau pelindung untuk objek lain. Objek proxy ini dapat mengendalikan akses ke objek asli. Di Javascript, objek `proxy` sudah ada dalam bahasa ini dan memudahkan implementasi pola desain Proxy.

7.1. Komponen Pola Proxy

Proksi (Proxy)

Proksi berisi antarmuka yang mirip dengan objek asli, menyimpan referensi yang memungkinkan proksi mengakses objek asli, dan menangani permintaan serta meneruskannya ke objek asli.

Subyek Asli (RealSubject)

Ini adalah objek sebenarnya yang diwakili oleh Proksi.

7.2. Manfaat dari Proksi

Akses yang Terkendali

Proksi memungkinkan Anda mengendalikan akses ke objek asli, memungkinkan Anda mengimplementasikan logika pengendalian akses seperti izin, pembatasan, atau validasi sebelum mengizinkan akses ke objek yang mendasarinya.

Penambahan Perilaku (Behavior Augmentation)

Proksi dapat menambahkan perilaku atau fungsionalitas tambahan sebelum atau setelah pemanggilan metode atau akses ke properti objek asli. Ini berguna untuk mengimplementasikan perhatian silang seperti logging, caching, atau penanganan kesalahan.

Penggunaan Cache

Proksi dapat mengimplementasikan mekanisme caching untuk menyimpan hasil operasi yang mahal, meningkatkan kinerja dan efisiensi.

Inisialisasi Malas (Lazy Initialization)

Proksi memungkinkan inisialisasi malas, di mana Anda dapat menunda pembuatan objek aktual hingga dibutuhkan. Ini dapat meningkatkan kinerja dengan mengurangi penggunaan sumber daya awal.

7.3. Contoh

```
// Objek asli yang mewakili rekening bank
const rekeningBank = {
  saldo: 1000,

  setor(amount) {
    this.saldo += amount;
    console.log(`Setor ${amount}. Saldo baru: ${this.saldo}`);
  },

  tarik(amount) {
    if (amount <= this.saldo) {
      this.saldo -= amount;
      console.log(`Tarik ${amount}. Saldo baru: ${this.saldo}`);
    } else {
      console.log("Dana tidak mencukupi.");
    }
  },
};

// Membuat proksi untuk rekening bank
const proksiRekeningBank = new Proxy(rekeningBank, {
  // Menangkap akses properti
  get(target, property) {
    if (property === "saldo") {
      // Tambahkan perilaku kustom sebelum mengakses 'saldo'
      console.log("Saldo diakses.");
    }
    return target[property];
  },

  // Menangkap pemanggilan metode
  apply(target, thisArg, args) {
    // Tambahkan perilaku kustom sebelum memanggil metode
    console.log(`Metode "${args[0]}" dipanggil.`);
    return target.apply(thisArg, args);
  },
});

// Mengakses proksi
console.log(proksiRekeningBank.saldo); // Ini akan memicu perilaku kustom

proksiRekeningBank.setor(500); // Ini akan memicu perilaku kustom untuk pemanggilan metode
```

Pola Desain Perilaku

Berfokus pada bagaimana objek berkomunikasi satu sama lain dan menetapkan tanggung jawab kepada mereka.

1. Rantai Tanggung Jawab (Chain of Responsibility)

Rantai Tanggung Jawab adalah pola desain perilaku, dan tujuannya adalah mengambil permintaan dan meneruskannya melalui serangkaian penangan. Ketika permintaan melewati rantai, setiap penangan akan memutuskan apakah akan memproses permintaan atau meneruskannya ke penangan berikutnya dalam rantai. Pola ini memungkinkan beberapa penangan untuk menangani permintaan tanpa pengirim perlu tahu penangan mana yang akan memprosesnya.

1.1. Komponen Rantai Tanggung Jawab

Permintaan (Request)

Permintaan hanyalah objek yang dikirim oleh klien yang perlu diproses. Permintaan melewati rantai penangan hingga diproses atau mencapai akhir rantai.

Interface/Kelas Penangan Abstrak (Abstract Handler)

Ini adalah antarmuka/kelas dasar yang mendefinisikan metode yang akan digunakan untuk menangani permintaan. Antarmuka penangan berisi logika untuk urutan rantai dan bagaimana permintaan dilewati.

Penangan Konkret (Concrete Handlers)

Ini adalah metode/kelas yang mengimplementasikan penangan abstrak. Setiap penangan konkret berisi logika untuk menangani jenis permintaan tertentu. Jika penangan konkret dapat memproses permintaan, maka akan melakukannya; jika tidak, maka akan meneruskan permintaan ke penangan berikutnya.

1.2. Manfaat Rantai Tanggung Jawab

Kemudahan Penggunaan

Pengirim tidak perlu mengetahui metode khusus untuk memproses permintaan, pengirim hanya perlu meneruskannya ke rantai. Ini memudahkan pengirim untuk memproses permintaan.

Fleksibilitas dan Perluasan

Penangan baru dapat ditambahkan ke rantai atau dihapus dari rantai tanpa mengubah kode pengirim. Ini memungkinkan modifikasi dinamis urutan pemrosesan.

Mempromosikan Segregasi yang Bertanggung Jawab

Penangan bertanggung jawab untuk memproses jenis permintaan tertentu berdasarkan logika yang ditentukan. Ini mempromosikan pemisahan tanggung jawab yang jelas, sehingga lebih mudah mengelola dan memelihara setiap penangan secara independen.

Pemrosesan Permintaan Berurutan

Pola ini memastikan bahwa setiap permintaan diproses secara berurutan melalui rantai penangan. Setiap penangan dapat memilih untuk memproses permintaan atau meneruskannya ke penangan berikutnya. Ini dapat sangat berguna dalam skenario di mana permintaan perlu diproses dalam urutan tertentu.

1.3. Contoh

```
// Handler interface
class CoffeeHandler {
  constructor() {
    this.nextHandler = null;
  }

  setNext(handler) {
    this.nextHandler = handler;
  }

  processRequest(request) {
    throw new Error(
      "Metode `processRequest` harus diimplementasikan oleh subclass (kelas turunan)."
    );
  }
}

// Konkrit handler untuk memesan kopi

class OrderCoffeeHandler extends CoffeeHandler {
  processRequest(request) {
    if (request === "Coffee") {
      return "Pesanan untuk segelas kopi telah ditempatkan.";
    } else if (this.nextHandler) {
      return this.nextHandler.processRequest(request);
    } else {
      return "Maaf, kami tidak menyajikan " + request + ".";
    }
  }
}

// Konkrit handler untuk menyiapkan kopi

class PrepareCoffeeHandler extends CoffeeHandler {
  processRequest(request) {
    if (request === "PrepareCoffee") {
      return "Kopi sedang disiapkan.";
    } else if (this.nextHandler) {
      return this.nextHandler.processRequest(request);
    } else {
      return "Tidak bisa menyiapkan " + request + ".";
    }
  }
}

// Kode klien
const orderHandler = new OrderCoffeeHandler();
const prepareHandler = new PrepareCoffeeHandler();

// Menyiapkan rantai (chain)
orderHandler.setNext(prepareHandler);

// Memesan kopi
console.log(orderHandler.processRequest("Coffee")); // Hasil: Pesanan ditempatkan untuk segelas kopi.

// Menyiapkan kopi
console.log(orderHandler.processRequest("PrepareCoffee")); // Hasil: Kopi sedang disiapkan.

// Coba pesan yang lain
console.log(orderHandler.processRequest("Tea")); // Hasil: Maaf, kami tidak menyajikan Teh.
```

2. Command

Pola desain perintah adalah pola desain perilaku yang memungkinkan Anda untuk mengkapsulasi permintaan sebagai objek. Objek tersebut akan berisi semua informasi yang diperlukan untuk eksekusi permintaan. Pola ini memungkinkan untuk memarameterisasi dan mengantri permintaan serta memberikan kemampuan untuk membatalkan operasi.

2.1. Komponen-Komponen Pola Komando

Invoker

Objek yang meminta eksekusi perintah. Invoker memiliki referensi ke perintah dan dapat mengeksekusi perintah dengan memanggil metodenya `execute`. Invoker tidak perlu mengetahui detail bagaimana perintah tersebut dieksekusi. Ia hanya memicu perintah.

Command

Ini adalah antarmuka atau kelas abstrak yang mendeklarasikan metode `execute`. Ini mendefinisikan metode umum yang harus diimplementasikan oleh kelas perintah konkret.

Receiver

Ini adalah objek yang melakukan pekerjaan sebenarnya ketika metode `execute` dari perintah dipanggil. Penerima tahu bagaimana melaksanakan tindakan yang terkait dengan perintah tertentu.

2.2. Manfaat Pola Komando

Fleksibilitas dan Kemudahan Perluasan

Pola ini memungkinkan penambahan perintah baru dengan mudah tanpa perlu memodifikasi invoker atau penerima.

Operasi Pembatalan dan Pengulangan

Pola perintah memfasilitasi implementasi operasi pembatalan dan pengulangan. Setiap objek perintah dapat melacak status sebelumnya, memungkinkan pembatalan tindakan yang dieksekusi.

Parameterisasi dan Penyusunan Antrian

Perintah dapat diparameterisasi dengan argumen, memungkinkan penyesuaian tindakan saat runtime. Selain itu, pola ini memungkinkan penyusunan antrian dan penjadwalan permintaan, memberikan kendali atas urutan eksekusi.

Riwayat dan Pencatatan

Memungkinkan untuk mempertahankan riwayat perintah yang dieksekusi, yang dapat berguna untuk audit, pencatatan, atau pelacakan tindakan pengguna.

2.3. Contoh

```
class Command {
  constructor(receiver) {
    this.receiver = receiver;
  }

  execute() {
    throw new Error("Metode execute() harus diimplementasikan");
  }
}

class ConcreteCommand extends Command {
  constructor(receiver, parameter) {
    super(receiver);
    this.parameter = parameter;
  }

  execute() {
    this.receiver.action(this.parameter);
  }
}

class Receiver {
  action(parameter) {
    console.log(
      `Receiver sedang menjalankan aksi dengan parameter: ${parameter}`
    );
  }
}

class Invoker {
  constructor() {
    this.commands = [];
  }

  addCommand(command) {
    this.commands.push(command);
  }

  executeCommands() {
    this.commands.forEach((command) => command.execute());
    this.commands = []; // Menghapus perintah yang sudah dieksekusi
  }
}

// Penggunaan
const receiver = new Receiver();
const command1 = new ConcreteCommand(receiver, "Parameter Perintah 1");
const command2 = new ConcreteCommand(receiver, "Parameter Perintah 2");

const invoker = new Invoker();
invoker.addCommand(command1);
invoker.addCommand(command2);

invoker.executeCommands();
```

3. Interpolator

Pola desain interperator digunakan untuk menentukan tata bahasa (grammar) untuk suatu bahasa dan menyediakan sebuah penerjemah (interpreter) untuk mengartikan kalimat-kalimat dalam bahasa tersebut. Biasanya digunakan untuk membuat penerjemah bahasa atau pemroses bahasa, tetapi Anda juga dapat menggunakannya dalam aplikasi Anda. Bayangkan Anda memiliki aplikasi desktop yang kompleks, Anda dapat merancang bahasa skrip dasar yang memungkinkan pengguna akhir untuk memanipulasi aplikasi Anda melalui instruksi sederhana.

3.1. Komponen-komponen dari Interpolator

Konteks

Sebuah keadaan global atau konteks yang digunakan oleh Interpolator untuk menginterpretasikan ekspresi-ekspresi. Konteks ini sering berisi informasi yang relevan selama interpretasi ekspresi.

Ekspresi-ekspresi Abstrak

Mendefinisikan antarmuka untuk semua jenis ekspresi dalam tata bahasa (grammar) bahasa tersebut. Ekspresi-ekspresi ini biasanya diwakili sebagai kelas abstrak atau antarmuka.

Ekspresi-e Ekspresi Terminal

Mewakili simbol-simbol terminal dalam tata bahasa (grammar). Ini adalah daun-daun dari pohon ekspresi. Ekspresi terminal mengimplementasikan antarmuka yang didefinisikan oleh ekspresi abstrak.

Ekspresi-e Ekspresi Non-Terminal

Mewakili simbol-simbol non-terminal dalam tata bahasa (grammar). Ekspresi non-terminal menggunakan ekspresi-ekspresi terminal dan/atau ekspresi-e non-terminal lainnya untuk mendefinisikan ekspresi-e kompleks dengan menggabungkan atau menggabungkan mereka.

Pohon Ekspresi (Expression Tree)

Mewakili struktur hirarkis ekspresi bahasa tersebut. Pohon ekspresi dibangun dengan menggabungkan ekspresi terminal dan ekspresi non-terminal berdasarkan aturan tata bahasa bahasa tersebut.

Penerjemah (Interpreter)

Mendefinisikan antarmuka atau kelas yang menginterpretasikan pohon sintaks abstrak yang dibuat oleh pohon ekspresi. Biasanya mencakup metode `interpret` yang mengambil konteks dan menginterpretasikan ekspresi berdasarkan konteks yang diberikan.

klien

Membangun pohon sintaks abstrak menggunakan ekspresi-e terminal dan non-terminal berdasarkan tata bahasa bahasa. Klien kemudian menggunakan penerjemah untuk menginterpretasikan ekspresi.

3.2. Manfaat Interpreters

Kemudahan Interpretasi Tata Bahasa (Grammar)

Pola ini menyederhanakan interpretasi aturan tata bahasa yang kompleks dengan memecahnya menjadi ekspresi-e yang lebih kecil dan lebih mudah dikelola. Setiap jenis ekspresi menangani interpretasinya sendiri, menjadikan proses interpretasi secara keseluruhan lebih mudah dikelola.

Penanganan Kesalahan yang Lebih Baik

Karena setiap jenis ekspresi menangani interpretasinya sendiri, penanganan kesalahan dapat disesuaikan dengan ekspresi tertentu. Ini memungkinkan pesan kesalahan yang lebih tepat dan bermakna saat memparsing atau menginterpretasikan masukan.

Fleksibilitas dan Pemuaian (Extensibility)

Pola interpeter menyediakan cara fleksibel untuk mendefinisikan dan memperluas aturan tata bahasa dan ekspresi bahasa tanpa mengubah logika inti interpreter. Anda dapat dengan mudah menambahkan ekspresi-e baru dengan membuat kelas-kelas ekspresi terminal dan non-terminal yang baru.

Integrasi dengan Pola Desain Lainnya

Pola Interpolator dapat digabungkan dengan pola desain lain, seperti Composite, untuk membangun hierarki ekspresi yang kompleks. Ini memungkinkan pembuatan interpreter yang kuat dan kaya fitur.

3.3. Contoh

```
// Ekspresi Abstrak
class Ekspresi {
  tafsirkan(konteks) {
    // Akan digantikan oleh subclass
  }
}

// Ekspresi Terminal: EkspresiAngka
class EkspresiAngka extends Ekspresi {
  constructor(angka) {
    super();
    this.angka = angka;
  }

  tafsirkan(konteks) {
    return this.angka;
  }
}

// Ekspresi Terminal: EkspresiVariabel
class EkspresiVariabel extends Ekspresi {
  constructor(variabel) {
    super();
    this.variabel = variabel;
  }

  tafsirkan(konteks) {
    return konteks[this.variabel] || 0;
  }
}

// Ekspresi Non-terminal: EkspresiTambah
class EkspresiTambah extends Ekspresi {
  constructor(ekspresi1, ekspresi2) {
    super();
    this.ekspresi1 = ekspresi1;
    this.ekspresi2 = ekspresi2;
  }

  tafsirkan(konteks) {
    return (
      this.ekspresi1.tafsirkan(konteks) + this.ekspresi2.tafsirkan(konteks)
    );
  }
}

// Ekspresi Non-terminal: EkspresiKurang
class EkspresiKurang extends Ekspresi {
  constructor(ekspresi1, ekspresi2) {
    super();
    this.ekspresi1 = ekspresi1;
    this.ekspresi2 = ekspresi2;
  }

  tafsirkan(konteks) {
    return (
      this.ekspresi1.tafsirkan(konteks) - this.ekspresi2.tafsirkan(konteks)
    );
  }
}

// Kode Klien
const konteks = { a: 10, b: 5, c: 2 };

const ekspresi = new EkspresiKurang(
  new EkspresiTambah(new EkspresiVariabel("a"), new EkspresiVariabel("b")),
  new EkspresiVariabel("c")
);
```

```
const hasil = ekspresi.tafsirkan(konteks);
console.log("Hasil:", hasil); // Output: Hasil: 13
```

4. Iterator

Pola Iterator memungkinkan klien untuk secara efektif mengulangi koleksi objek

4.1. Komponen dari Iterator

Iterator

Mengimplementasikan antarmuka atau kelas Iterator dengan metode seperti `first()` , `next()` . Iterator melacak posisi saat ini saat melintasi koleksi.

Item

Ini adalah objek individu dari koleksi yang akan dijelajahi oleh Iterator.

4.2. Manfaat dari iterator

Kompatibilitas dengan Struktur Data yang Berbeda

Pola Iterator memungkinkan logika iterasi yang sama diterapkan pada struktur data yang berbeda.

Dukungan untuk Iterasi Bersamaan

Iterator dapat mendukung iterasi bersamaan atas koleksi yang sama tanpa saling mengganggu, ini memungkinkan klien untuk mengulangi koleksi dengan cara yang berbeda secara bersamaan.

Pemuatan Malas (Lazy Loading)

Iterator dapat dirancang untuk memuat elemen sesuai permintaan, yang bermanfaat untuk koleksi besar di mana memuat semua elemen sekaligus mungkin tidak praktis atau memakan banyak sumber daya. Elemen dapat diambil sesuai kebutuhan, mengoptimalkan penggunaan memori.

Antarmuka yang Disederhanakan

Pola Iterator menyediakan antarmuka yang bersih dan konsisten untuk mengakses elemen dalam koleksi tanpa mengekspos struktur internal koleksi. Ini menyederhanakan penggunaan dan pemahaman koleksi.

4.3. Contoh

```
// Kelas Mobil yang Mewakili Sebuah Mobil
class Car {
  constructor(make, model) {
    this.make = make;
    this.model = model;
  }

  getInfo() {
    return `${this.make} ${this.model}`;
  }
}

// Antarmuka Iterator
class Iterator {
  constructor(collection) {
    this.collection = collection;
    this.index = 0;
  }

  next() {
    return this.collection[this.index++];
  }

  hasNext() {
    return this.index < this.collection.length;
  }
}

// Koleksi Mobil
class CarCollection {
  constructor() {
    this.cars = [];
  }

  addCar(car) {
    this.cars.push(car);
  }

  createIterator() {
    return new Iterator(this.cars);
  }
}

// Penggunaan
const carCollection = new CarCollection();
carCollection.addCar(new Car("Toyota", "Corolla"));
carCollection.addCar(new Car("Honda", "Civic"));
carCollection.addCar(new Car("Ford", "Mustang"));

const iterator = carCollection.createIterator();

while (iterator.hasNext()) {
  const car = iterator.next();
  console.log(car.getInfo());
}
```

5. Mediator

Pola Mediator berperan sebagai perantara antara sekelompok objek dengan mengkapsulasi cara objek-objek ini berinteraksi. Mediator memfasilitasi komunikasi antara komponen-komponen yang berbeda dalam suatu sistem tanpa mereka perlu merujuk langsung satu sama lain.

5.1. Komponen dari Mediator

Mediator

Mediator mengelola kontrol pusat atas operasi. Ini berisi antarmuka untuk berkomunikasi dengan objek-objek Kolaborator dan memiliki referensi ke setiap objek Kolaborator.

Kolaborator

Kolaborator adalah objek-objek yang dimediasi, setiap kolaborator memiliki referensi ke mediator.

5.2. Manfaat dari Mediator

Kontrol Terpusat

Pusatkan komunikasi dalam Mediator memungkinkan pengendalian dan koordinasi yang lebih baik antara komponen-komponen. Mediator dapat mengelola distribusi pesan, mengutamakan pesan, dan menerapkan logika khusus berdasarkan kebutuhan sistem.

Pemudahan Komunikasi

Mediator menyederhanakan logika komunikasi, karena komponen mengirim pesan ke Mediator daripada berurusan dengan kompleksitas komunikasi langsung. Hal ini menyederhanakan interaksi antara komponen dan memungkinkan pemeliharaan dan pembaruan yang lebih mudah.

Dapat Digunakan Kembali Mediator

Mediator dapat digunakan kembali di berbagai komponen dan skenario, memungkinkan satu titik kontrol untuk bagian-bagian yang berbeda dari aplikasi. Dapat digunakannya ini mempromosikan konsistensi dan membantu dalam mengelola aliran komunikasi dengan efisien.

Mendorong Pemeliharaan

Pola Mediator mendorong pemeliharaan dengan mengurangi kompleksitas komunikasi antar komponen yang langsung. Seiring pertumbuhan sistem, perubahan dan pembaruan dapat dilakukan dalam Mediator tanpa memengaruhi komponen-komponen individu, membuat pemeliharaan lebih mudah dan kurang rentan terhadap kesalahan.

```
var Participant = function (name) {
  this.name = name;
  this.chatroom = null;
};

Participant.prototype = {
  send: function (message, to) {
    this.chatroom.send(message, this, to);
  },
  receive: function (message, from) {
    console.log(from.name + " to " + this.name + ": " + message);
  },
};

var Chatroom = function () {
  var participants = {};

  return {
    register: function (participant) {
      participants[participant.name] = participant;
      participant.chatroom = this;
    },

    send: function (message, from, to) {
      if (to) {
        // Pesan Tunggal
        to.receive(message, from);
      } else {
        // Pesan Penyiaran
        for (key in participants) {
          if (participants[key] !== from) {
            participants[key].receive(message, from);
          }
        }
      }
    },
  };
};

function run() {
  var yoko = new Participant("Yoko");
  var john = new Participant("John");
  var paul = new Participant("Paul");
  var ringo = new Participant("Ringo");

  var chatroom = new Chatroom();
  chatroom.register(yoko);
  chatroom.register(john);
  chatroom.register(paul);
  chatroom.register(ringo);

  yoko.kirim("Semua yang Anda butuhkan adalah cinta.");
  yoko.kirim("Aku mencintaimu John.");
  john.kirim("Hei, tidak perlu disiarkan", yoko);
  paul.kirim("Ha, aku mendengarnya!");
  ringo.kirim("Paul, apa pendapatmu?", paul);
}
```

6. Memento

Pola desain Memento memungkinkan status objek disimpan dan dipulihkan pada waktu yang akan datang tanpa mengekspos struktur internalnya. Memento adalah objek terpisah yang menyimpan status objek asli.

6.1. Komponen dari Memento

Originator

Ini adalah objek yang disimpan. Itu membuat objek Memento untuk menyimpan status internalnya.

Memento

Memento bertanggung jawab untuk menyimpan status dari Originator. Ini memiliki metode untuk mengambil dan mengatur status Originator tetapi tidak mengekspos struktur internal Originator.

Caretaker

Caretaker bertanggung jawab untuk melacak dan mengelola Memento. Ini tidak mengubah atau memeriksa isi dari Memento.

6.2. Manfaat dari Memento

Pemeliharaan dan Pemulihan Status

Memento memungkinkan status internal objek disimpan dan dipulihkan pada waktu yang akan datang.

Operasi Undo/Redo

Memento memudahkan implementasi fungsi undo dan redo dengan mudah. Karena Memento menyimpan status objek pada berbagai titik waktu, Anda dapat mendukung pembatalan perubahan yang dibuat pada objek atau mengulang perubahan yang dibuat pada objek.

Kinerja yang Ditingkatkan

Menyimpan status objek dalam Memento memungkinkan operasi penyimpanan dan pengambilan yang lebih efisien dibandingkan dengan pendekatan lain.

Desain yang Fleksibel

Ini menyediakan cara fleksibel untuk mengelola sejarah status objek. Caretaker dapat memutuskan status mana yang akan dijaga dan kapan akan mengembalikannya, memungkinkan pendekatan yang dapat disesuaikan berdasarkan kebutuhan aplikasi.

6.3. Contoh

```
// Kelas Komputer (Originator)
class Computer {
  constructor() {
    this.os = "";
    this.version = "";
  }

  setOS(os, version) {
    this.os = os;
    this.version = version;
  }

  getState() {
    return {
      os: this.os,
      version: this.version,
    };
  }

  restoreState(state) {
    this.os = state.os;
    this.version = state.version;
  }
}

// Caretaker
class Caretaker {
  constructor() {
    this.mementos = {};
    this.nextKey = 1;
  }

  add(memento) {
    const key = this.nextKey++;
    this.mementos[key] = memento;
    return key;
  }

  get(key) {
    return this.mementos[key];
  }
}

function run() {
  const computer = new Computer();
  const caretaker = new Caretaker();

  // Simpan status
  const originalState = computer.getState();
  const key = caretaker.add(originalState);

  // Merusak status
  computer.setOS("Windows", "11");

  // Pulihkan status asli
  const restoredState = caretaker.get(key);
  computer.restoreState(restoredState);

  console.log(computer.getState()); // Output: { os: '', version: '' }
}

run();
```

7. Observer

Pola observer memungkinkan banyak objek untuk berlangganan acara yang disiarkan oleh objek lain.

7.1. Komponen Observer

Subjek

Subjek adalah objek yang mengimplementasikan antarmuka yang memungkinkan objek pengamat berlangganan dan mengirim pemberitahuan kepada pengamat saat statusnya berubah.

Pengamat

Pengamat berlangganan subjek dan biasanya memiliki fungsi yang dipanggil saat diberitahu oleh Subjek.

7.2. Manfaat Pengamat

Pemudahan Penanganan Acara

Pola Observer dapat menyederhanakan mekanisme penanganan acara, karena acara dapat diperlakukan sebagai pemberitahuan kepada pengamat tentang perubahan status.

Mengurangi Kode yang Sama

Alih-alih menggandakan kode yang sama untuk merespons perubahan status di beberapa tempat, pola Observer memungkinkan adanya tempat terpusat untuk mengelola respons ini, mempromosikan kode yang lebih bersih dan mudah dikelola.

Dukungan untuk Komunikasi Penyiaran

Pola Observer memfasilitasi model komunikasi "satu-ke-banyak", di mana satu acara memicu tindakan dalam beberapa pengamat. Ini berguna dalam skenario di mana beberapa komponen perlu bereaksi terhadap perubahan status.

Modularitas dan Dapat Digunakan Kembali

Pengamat dapat digunakan kembali di berbagai subjek, mempromosikan modularitas dan dapat digunakan kembali kode. Ini memungkinkan kode yang lebih fleksibel dan mudah dikelola.

7.3. Contoh

```
function Click() {
  this.handlers = []; // observers
}

Click.prototype = {
  subscribe: function (fn) {
    this.handlers.push(fn);
  },

  unsubscribe: function (fn) {
    this.handlers = this.handlers.filter(function (item) {
      if (item !== fn) {
        return item;
      }
    });
  },

  fire: function (o, thisObj) {
    var scope = thisObj || window;
    this.handlers.forEach(function (item) {
      item.call(scope, o);
    });
  },
};

function run() {
  var clickHandler = function (item) {
    console.log("fired: " + item);
  };

  var click = new Click();

  click.subscribe(clickHandler);
  click.fire("event #1");
  click.unsubscribe(clickHandler);
  click.fire("event #2");
  click.subscribe(clickHandler);
  click.fire("event #3");
}
```

8. State

Pola State adalah pola desain perilaku yang memungkinkan Anda memiliki objek dasar dan memberikannya fungsionalitas tambahan berdasarkan statusnya. Pola ini sangat berguna ketika sebuah objek perlu mengubah perilakunya berdasarkan status internalnya.

8.1. Komponen dari State

State

Ini adalah objek yang mengkapsulasi nilai-nilai Status dan perilaku yang terkait dengan Status tersebut.

Context

Ini adalah objek yang menjaga referensi ke objek State yang menentukan Status saat ini. Ini juga mencakup antarmuka yang memungkinkan objek State lain untuk mengubah Status saat ini menjadi Status yang berbeda.

8.2. Manfaat dari State

Kode yang Modular dan Terorganisir

Setiap State dikapsulkan dalam kelasnya sendiri, membuat kode menjadi modular dan mudah dikelola.

Tidak Perlu Pernyataan Switch yang Panjang

Pernyataan switch juga dapat digunakan untuk mengubah perilaku objek, tetapi masalah dengan metode ini adalah pernyataan switch bisa menjadi sangat panjang saat proyek Anda berkembang. Pola State memperbaiki masalah ini.

Mempromosikan Dapat Digunakan Kembali

Status dapat digunakan kembali di berbagai konteks, ini mengurangi duplikasi kode.

Membuat Pengujian Lebih Sederhana

Menguji kelas status individual secara terisolasi lebih mudah dan efektif daripada menguji objek monolitik dengan logika kondisional yang kompleks.

8.3. Contoh

```
class Car {
  constructor() {
    this.state = new ParkState();
  }

  setState(state) {
    this.state = state;
    console.log(`Mengganti status menjadi: ${state.constructor.name}`);
  }

  park() {
    this.state.park(this);
  }

  drive() {
    this.state.drive(this);
  }

  reverse() {
    this.state.reverse(this);
  }
}
class ParkState {
  park(car) {
    console.log("Mobil sudah berada dalam Posisi Parkir");
  }

  drive(car) {
    console.log("Beralih ke Posisi Drive");
    car.setState(new DriveState());
  }

  reverse(car) {
    console.log("Beralih ke Posisi Reverse");
    car.setState(new ReverseState());
  }
}

class DriveState {
  park(car) {
    console.log("Beralih ke Posisi Parkir");
    car.setState(new ParkState());
  }

  drive(car) {
    console.log("Mobil sudah berada dalam Posisi Drive");
  }

  reverse(car) {
    console.log("Beralih ke Posisi Reverse");
    car.setState(new ReverseState());
  }
}

class ReverseState {
  park(car) {
    console.log("Beralih ke Posisi Parkir");
    car.setState(new ParkState());
  }

  drive(car) {
    console.log("Beralih ke Posisi Drive");
    car.setState(new DriveState());
  }

  reverse(car) {
    console.log("Mobil sudah berada dalam Posisi Reverse");
  }
```

```
}

// Contoh penggunaan
const car = new Car();

car.drive();
car.reverse();
car.drive();
car.park();
car.drive(); // Mencoba untuk mengemudi saat diparkir
```

9. Strategy

Pola Strategy pada dasarnya adalah pola desain yang memungkinkan Anda memiliki sekelompok algoritma (strategi) yang dapat dipertukarkan.

9.1. Komponen dari Strategy

Strategi

Ini adalah algoritma yang mengimplementasikan antarmuka Strategi.

Konteks

Ini adalah objek yang menjaga referensi ke Strategi saat ini. Ini mendefinisikan antarmuka yang memungkinkan klien untuk mengubah Strategi saat ini menjadi Strategi yang berbeda atau mengambil perhitungan dari Strategi saat ini yang dirujuk.

9.2. Manfaat dari Strategy

Algoritma yang Dapat Dipertukarkan Secara Dinamis

Strategi dapat ditukar saat runtime, memungkinkan pemilihan dinamis algoritma berdasarkan kondisi atau persyaratan yang berbeda. Hal ini sangat berguna ketika algoritma yang tepat dapat bervariasi berdasarkan masukan pengguna, pengaturan konfigurasi, atau faktor lainnya.

Fleksibilitas dan Pemeliharaan

Strategi dapat diubah atau diperluas tanpa mengubah konteks yang menggunakannya. Ini membuat sistem lebih fleksibel dan lebih mudah dikelola karena perubahan dalam satu strategi tidak memengaruhi yang lain.

Membuat Pengujian Lebih Sederhana

Menguji strategi secara terisolasi lebih mudah karena setiap strategi adalah kelas terpisah. Ini memungkinkan pengujian yang terfokus dan memastikan bahwa perubahan dalam satu strategi tidak secara tidak sengaja memengaruhi yang lain.

Dapat Digunakan Kembali

Strategi dapat digunakan kembali dalam berbagai konteks atau aplikasi, mempromosikan penggunaan kode dan mengurangi duplikasi.

9.3. Contoh

```
class RegularCustomerStrategy {
  calculatePrice(bookPrice) {
    // Pelanggan reguler mendapatkan diskon tetap 10%
    return bookPrice * 0.9;
  }
}

class VIPCustomerStrategy {
  calculatePrice(bookPrice) {
    // Pelanggan VIP mendapatkan diskon tetap 20%
    return bookPrice * 0.8;
  }
}

class TokoBuku {
  constructor(pricingStrategy) {
    this.pricingStrategy = pricingStrategy;
  }

  aturStrategiHarga(pricingStrategy) {
    this.pricingStrategy = pricingStrategy;
  }

  hitungHarga(hargaBuku) {
    return this.pricingStrategy.hitungHarga(hargaBuku);
  }
}

// Usage
const regularCustomerStrategy = new RegularCustomerStrategy();
const vipCustomerStrategy = new VIPCustomerStrategy();

const bookstore = new BookStore(regularCustomerStrategy);
console.log("Harga pelanggan reguler:", tokoBuku.hitungHarga(50)); // Output: 45 (diskon 10%)
tokoBuku.aturStrategiHarga(new VIPCustomerStrategy());
console.log("Harga pelanggan VIP:", tokoBuku.hitungHarga(50)); // Output: 40 (diskon 20%)
```

10. Template Method

Pola Template Method adalah pola desain perilaku yang mendefinisikan kerangka program dari sebuah algoritma dalam sebuah metode, namun memungkinkan subclass mengganti langkah-langkah spesifik dari algoritma tanpa mengubah strukturnya.

10.1. Komponen dari Template Method

Kelas Abstrak

Kelas abstrak adalah template untuk algoritma. Ini mendefinisikan antarmuka yang dapat diakses oleh klien untuk memanggil metodenya. Ini juga berisi semua fungsi yang dapat digantikan oleh subclass.

Kelas Konkrit

Mengimplementasikan langkah-langkah sebagaimana yang didefinisikan dalam Kelas Abstrak dan dapat membuat perubahan pada langkah-langkah tersebut.

10.2. Manfaat dari Template Method

Penggunaan Kembali Kode

Pola ini mempromosikan penggunaan kembali kode dengan mendefinisikan kerangka algoritma dalam kelas dasar. Subclass dapat menggunakan struktur ini dan hanya perlu memberikan implementasi untuk langkah-langkah tertentu.

Pemeliharaan yang Mudah

Membuat perubahan pada algoritma menjadi lebih sederhana karena modifikasi hanya perlu dilakukan di satu tempat, yaitu metode template dalam kelas abstrak, daripada di beberapa subclass. Ini mengurangi peluang kesalahan dan memudahkan pemeliharaan.

Ekstensi dan Variasi

Pola ini memungkinkan ekstensi dan variasi algoritma dengan mudah. Subclass dapat mengganti langkah-langkah tertentu untuk memberikan implementasi kustom, efektif memperluas atau mengubah perilaku algoritma tanpa mengubah struktur inti.

Aliran Kontrol

Metode template mendefinisikan aliran kontrol algoritma, membuatnya lebih mudah dikelola dan memahami urutan operasi dalam algoritma.

10.3. Contoh

```
class Camera {
  // Metode template yang mendefinisikan langkah-langkah umum untuk mengambil foto
  capturePhoto() {
    this.turnOn();
    this.initialize();
    this.setExposure();
    this.capture();
    this.turnOff();
  }

  // Langkah-langkah umum untuk menyalakan kamera
  turnOn() {
    console.log("Menyalakan kamera");
  }

  // Metode abstrak untuk menginisialisasi kamera (akan di-override oleh subclass)
  initialize() {
    throw new Error(
      "Metode abstrak: initialize() harus diimplementasikan oleh subclass"
    );
  }

  // Metode abstrak untuk mengatur eksposur (akan di-override oleh subclass)
  setExposure() {
    throw new Error(
      "Metode abstrak: setExposure() harus diimplementasikan oleh subclass"
    );
  }

  // Langkah-langkah umum untuk mengambil foto
  capture() {
    console.log("Mengambil foto");
  }

  // Langkah-langkah umum untuk mematikan kamera
  turnOff() {
    console.log("Mematikan kamera");
  }
}

class KameraDSLR extends Camera {
  initialize() {
    console.log("Menginisialisasi kamera DSLR");
  }

  setExposure() {
    console.log("Mengatur eksposur untuk kamera DSLR");
  }
}

class KameraMirrorless extends Camera {
  initialize() {
    console.log("Menginisialisasi kamera mirrorless");
  }

  setExposure() {
    console.log("Mengatur eksposur untuk kamera mirrorless");
  }
}

// Penggunaan
const kameraDSLR = new KameraDSLR();
console.log("Mengambil foto dengan kamera DSLR:");
kameraDSLR.capturePhoto();
console.log("");

const kameraMirrorless = new KameraMirrorless();
```

```
console.log("Mengambil foto dengan kamera mirrorless:");
kameraMirrorless.capturePhoto();
```

11. Visitor

Pola desain Visitor adalah pola desain perilaku yang memungkinkan Anda memisahkan algoritma atau operasi dari objek yang dioperasikan.

11.1. Komponen dari Visitor

ObjectStructure

Menjaga koleksi Elemen yang dapat diiterasi.

Elemen

Elemen berisi metode accept yang menerima objek visitor.

Visitor

Mengimplementasikan metode visit di mana argumen metode adalah elemen yang sedang dikunjungi. Inilah cara perubahan pada elemen dilakukan.

11.2. Manfaat dari Visitor

Prinsip Terbuka/Tertutup

Pola ini sejalan dengan Prinsip Terbuka/Tertutup, yang menyatakan bahwa entitas perangkat lunak (kelas, modul, fungsi) harus terbuka untuk ekstensi tetapi tertutup untuk modifikasi. Anda dapat memperkenalkan operasi baru (visitor baru) tanpa mengubah struktur objek atau elemen yang ada.

Dapat Diperluas

Anda dapat memperkenalkan perilaku atau operasi baru dengan menambahkan implementasi visitor baru tanpa mengubah elemen atau struktur objek yang ada. Ini membuat sistem lebih dapat diperluas, memungkinkan penambahan fitur atau perilaku baru dengan mudah.

Perilaku yang Terpusat

Pola Visitor memusatkan kode terkait perilaku dalam kelas-kelas visitor. Setiap visitor mengkapsulasi perilaku tertentu, yang dapat digunakan kembali di berbagai elemen, mempromosikan penggunaan kode kembali dan modularitas.

Konsistensi dalam Operasi

Dengan pola Visitor, Anda dapat memastikan bahwa operasi tertentu (metode visitor) diterapkan secara konsisten di berbagai elemen, karena metode accept setiap elemen memanggil metode visitor yang sesuai untuk jenis elemen tersebut.

11.3. Contoh

```
class GymMember {
  constructor(name, subscriptionType, fitnessScore) {
    this.name = name;
    this.subscriptionType = subscriptionType;
    this.fitnessScore = fitnessScore;
  }

  accept(visitor) {
    visitor.visit(this);
  }

  getName() {
    return this.name;
  }

  getSubscriptionType() {
    return this.subscriptionType;
  }

  getFitnessScore() {
    return this.fitnessScore;
  }

  setFitnessScore(score) {
    this.fitnessScore = score;
  }
}

class FitnessEvaluation {
  visit(member) {
    member.setFitnessScore(member.getFitnessScore() + 10);
  }
}

class DiskonKeanggotaan {
  visit(member) {
    if (member.getSubscriptionType() === "Premium") {
      console.log(
        `${member.getName()}: Skor Kebugaran - ${member.getFitnessScore()}, Jenis Keanggotaan - ${member.getSub
      );
    } else {
      console.log(
        `${member.getName()}: Skor Kebugaran - ${member.getFitnessScore()}, Jenis Keanggotaan - ${member.getSub
      );
    }
  }
}

function jalankan() {
  const anggotaGym = [
    new AnggotaGym("Alice", "Basic", 80),
    new AnggotaGym("Bob", "Premium", 90),
    new AnggotaGym("Eve", "Basic", 85),
  ];

  const evaluasiKebugaran = new EvaluasiKebugaran();
  const diskonKeanggotaan = new DiskonKeanggotaan();

  for (let i = 0; i < anggotaGym.length; i++) {
    const anggota = anggotaGym[i];

    anggota.accept(evaluasiKebugaran);
    anggota.accept(diskonKeanggotaan);
  }
}

run();
```

Referensi

Ballard, P. (2018). JavaScript in 24 Hours, Sams Teach Yourself. Sams Publishing.

Crockford, D. (2008). JavaScript: The Good Parts. O'Reilly Media.

Duckett, J. (2011). HTML & CSS: Design and Build Websites. Wiley.

Duckett, J. (2014). JavaScript and JQuery: Interactive Front-End Web Development. Wiley.

Flanagan, D. (2011). JavaScript: The Definitive Guide. O'Reilly Media.

Freeman, E., & Robson, E. (2014). Head First JavaScript Programming: A Brain-Friendly Guide. O'Reilly Media.

Frisbie, M. (2019). Professional JavaScript for Web Developers. Wrox.

Haverbeke, M. (2019). Eloquent JavaScript: A Modern Introduction to Programming. No Starch Press.

Herman, D. (2012). Effective JavaScript: 68 Specific Ways to Harness the Power of JavaScript. Addison-Wesley Professional.

McPeak, J., & Wilton, P. (2015). Beginning JavaScript. Wiley.

Morgan, N. (2014). JavaScript for Kids: A Playful Introduction to Programming. No Starch Press.

Murphy C, Clark R, Studholme O, et al. (2014). Beginning HTML5 and CSS3: The Web Evolved. Apress.

Nixon, R. (2021). Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5. O'Reilly Media.

Powell, T., & Schneider, F. (2012). JavaScript: The Complete Reference. McGraw-Hill Education.

Resig, J. (2007). Pro JavaScript Techniques. Apress.

Resig, J., & Bibeault, B. (2016). Secrets of the JavaScript Ninja. Manning Publications.

Robbins, J. N. (2014). Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics. O'Reilly Media.

Daftar Sumber Belajar

Artikel untuk Belajar JavaScript

Berikut adalah beberapa artikel yang cocok untuk pemula yang ingin mempelajari JavaScript:

1. Pengantar JavaScript oleh Hostinger Link
2. Tutorial JavaScript oleh Geeks for Geeks Link
3. Dasar-Dasar JavaScript oleh Mozilla Developer Link
4. Pengantar JavaScript oleh Microverse Link
5. Tutorial JavaScript oleh Stackify Link

Buku-buku untuk Belajar JavaScript

1. Anda Tidak Tahu JS - Kyle Simpson Link
2. Cara Lebih Pintar Belajar JavaScript - Mark Myers Link
3. Eloquent JavaScript - Marijn Haverbeke Link
4. JavaScript: Bagian-Bagian yang Bagus - Douglas Crockford Link
5. JavaScript Profesional untuk Pengembang Web Link
6. Head First JavaScript Link
7. Rahasia Ninja JavaScript Link
8. Prinsip-Prinsip JavaScript Berbasis Objek Link

Sumber YouTube untuk Belajar JavaScript

1. Traversy Media - Link
2. The Net Ninja - Link
3. freeCodeCamp - Link
4. CodeWithHarry - Link
5. Academind - Link
6. Penguasaan JavaScript - Link
7. SuperSimpleDev - Link
8. Dave Gray - Link
9. Programmer Cerdas - Link
10. Akshay Saini - Link
11. Hitesh Choudhary - Link
12. thenewboston - Link

Maintainer

Suman Kunwar

Contributors