Contents

[**Menulis Kode JavaScript Pertama** 1](#_Toc107414920)

[Comments 1](#_Toc107414921)

[Variable 1](#_Toc107414922)

[Tipe Data 4](#_Toc107414923)

[Undefined 4](#_Toc107414924)

[Numbers 5](#_Toc107414925)

[BigInt 6](#_Toc107414926)

[Strings 8](#_Toc107414927)

[Boolean 9](#_Toc107414928)

[Null 11](#_Toc107414929)

[Symbol 11](#_Toc107414930)

[Operator 12](#_Toc107414931)

[Assignment Operator 12](#_Toc107414932)

[Comparison Operator 13](#_Toc107414933)

[Logical Operator 15](#_Toc107414934)

[If/Else Statement 16](#_Toc107414935)

[Truthy & Falsy 20](#_Toc107414936)

[Switch Case Statement 20](#_Toc107414937)

[Loop 22](#_Toc107414938)

[For of loop 23](#_Toc107414939)

[While and do-while 24](#_Toc107414940)

[Infinite loops 25](#_Toc107414941)

[Rangkuman Materi 26](#_Toc107414942)

**Menulis Kode JavaScript Pertama**

Menulis kode dalam suatu bahasa pemrograman sederhananya adalah menuliskan instruksi-instruksi untuk dijalankan oleh komputer. Kode di bawah ini merupakan instruksi bagi terminal atau konsol untuk mencatat (*log*) kalimat “Hello, World!”.

1. console.log(“Hello World!”);

console.log adalah kode bawaan JavaScript untuk menampilkan pesan ke konsol, bisa berupa web konsol atau konsol dari *terminal/command prompt*.

[](https://www.dicoding.com/academies/256/tutorials/13863?from=15313)

Kode atau teks yang berada di dalam tanda kurung adalah pesan yang ingin ditampilkan. Pada contoh kode di atas, kita menggunakan tanda kutip (“”) untuk menandakan bahwa pesan yang ingin ditampilkan merupakan sebuah *string* atau teks. Kita akan membahas mengenai tipe data pada beberapa modul ke depan. Saat ini kita akan fokus untuk menampilkan teks terlebih dulu. Sekarang cobalah untuk menampilkan nama Anda sendiri ke konsol.

1

console.log("Hello World!");

2

​

3

// Tampilkan nama Anda ke layar pada kode di bawah:

 Input  Reset

 Jalankan

JavaScript membagi instruksi secara umum menjadi dua jenis, yaitu *expression* dan *statement*.

Expression adalah unit kode yang dapat dievaluasi menjadi suatu nilai. Untuk lebih mudahnya, perhatikan contoh kode berikut:

1. 5; // Baris kode ini merupakan expression karena interpreter akan membaca kode ini dan menghasilkan nilai 5.
2. 2 + 3; // Baris kode ini juga merupakan expression. Interpreter mengevaluasi kode dan juga akan menghasilkan nilai 5.

Sementara itu, statement adalah instruksi untuk melakukan sebuah aksi tertentu. Aksi ini bisa termasuk membuat variabel, melakukan perulangan, pengecekan kondisi, dll.

1. var yourName;
2. let yourAge;
3. const numberOfDays;  // declaration statement

Jika Anda masih bingung untuk membedakan expression dan statement, tidak perlu khawatir. Saat ini belum banyak kode yang bisa kita jadikan contoh. Namun, konsep ini akan banyak kita gunakan pada materi-materi yang akan datang. Yang perlu Anda ingat adalah:

* Expression merupakan kode yang menghasilkan nilai.
* Statement menunjukkan aksi yang dilakukan.

## Comments

Pada materi sebelumnya kita telah membahas bahwa kode yang kita tulis merupakan kumpulan instruksi yang harus dijalankan oleh komputer. Salah satu instruksi yang penting adalah memberi tahu komputer untuk mengabaikan perintah yang kita tulis. Instruksi yang ditulis dalam suatu program tetapi tidak dijalankan oleh komputer disebut “comments”.

Sebuah komentar akan dilewatkan oleh interpreter atau compiler, sehingga tidak akan memengaruhi alur program yang kita tulis. Komentar ini bisa digunakan sebagai dokumentasi atau penjelasan dari kode yang kita tulis.

Terdapat dua metode untuk memberikan komentar. Pertama, untuk memberikan komentar pada satu baris saja, kita bisa gunakan tanda dua garis miring (//) di awal baris.

// Ini merupakan komentar satu baris, kode di bawah tidak akan dijalankan

// console.log("Halo!");

Lalu, untuk memberikan komentar lebih dari satu baris kita bisa menggunakan tanda /\* sebagai pembuka komentar dan tanda \*/ untuk penutup komentar. Teks apa pun yang berada di antara tanda tersebut akan dijadikan komentar dan tidak akan dieksekusi.

/\* Ini merupakan komentar dengan lebih dari satu baris

Teks apapun yang berada disini akan dijadikan komentar.

Ketika menggunakan ini, jangan lupa untuk menutupnya.

\*/

## Variable

Ketika menulis sebuah program, kita memberi tahu komputer cara memproses informasi seperti mencetak teks ke layar atau melakukan operasi perhitungan. Untuk lebih mudah dalam penggunaan dan pemanggilan data, kita bisa memanfaatkan variabel. Variabel umumnya digunakan untuk menyimpan informasi atau nilai yang akan dikelola dalam sebuah program.

Pada JavaScript setidaknya ada tiga cara untuk mendeklarasikan sebuah variabel, yaitu menggunakan keyword var, let, dan const. Pada versi ECMAScript 2015 (ES6) dikenalkan deklarasi variabel dengan let dan const untuk menggantikan var yang dinilai kontroversial dan rawan menimbulkan bug.

Dalam kelas ini, kita akan banyak menggunakan keyword let dan const. Jika Anda penasaran kenapa var sudah tidak lagi disarankan, silakan simak diskusi [berikut](https://softwareengineering.stackexchange.com/questions/274342/is-there-any-reason-to-use-the-var-keyword-in-es6).

Sekarang bagaimana caranya membuat sebuah variabel dalam JavaScript? Cukup mudah. Ketikkan keyword let yang diikuti dengan nama variabelnya.

1. let lastName;

Kode untuk mendeklarasikan variabel seperti di atas juga dikenal dengan declaration statement.

Selanjutnya, Anda bisa mengisi nilai variabel di atas menggunakan tanda sama dengan (=).

let lastName;

lastName = "Skywalker";

console.log(lastName);

/\* output

Skywalker

\*/

Anda juga bisa langsung menginisialisasi nilai variabel setelah mendeklarasikannya seperti berikut:

let lastName = "Skywalker";

console.log(lastName);

/\* output

Skywalker

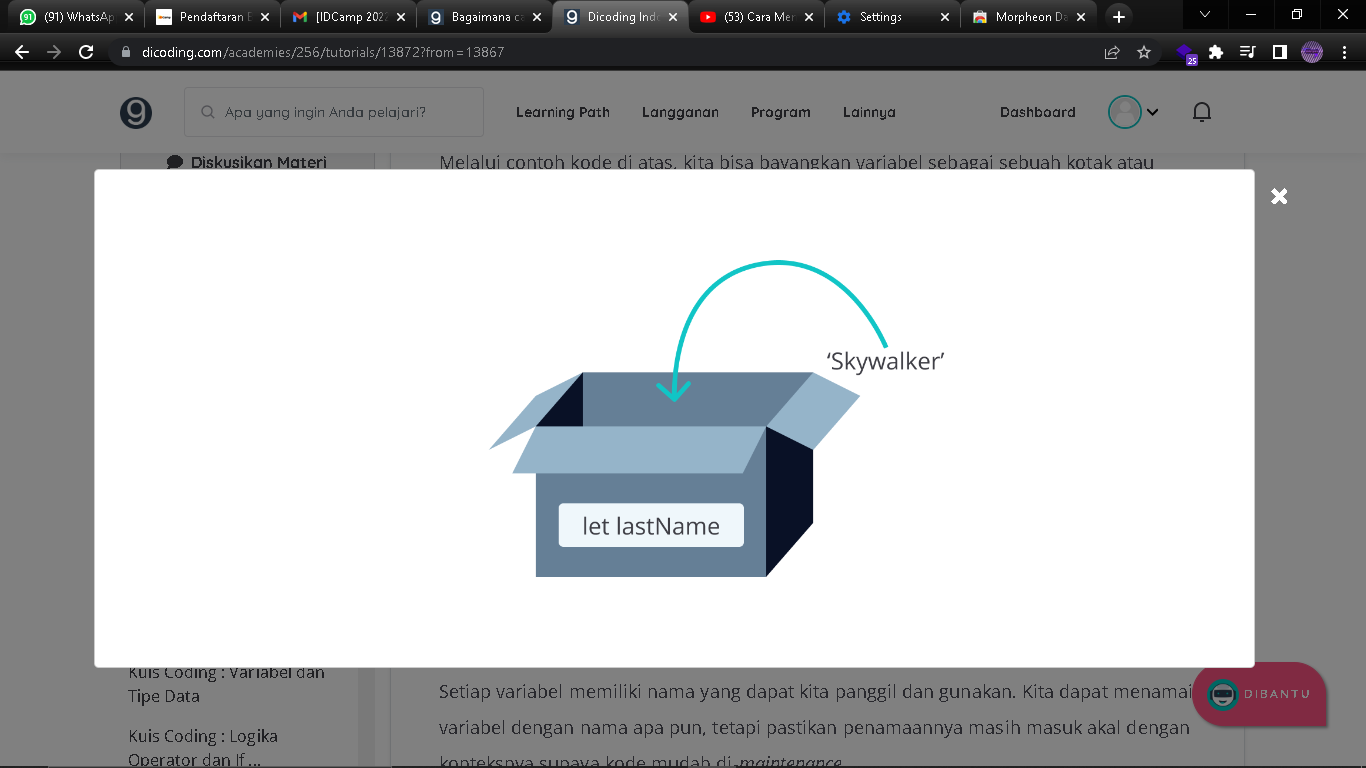
\*/

Kode untuk menginisialisasikan nilai ke dalam sebuah variabel dengan tanda sama dengan (=) ini disebut dengan assignment expression.

Tunggu sebentar. Kita kembali bertemu dengan istilah statement dan expression. Karena expression pasti menghasilkan nilai, sehingga mereka bisa muncul di mana pun dalam program JavaScript. Sementara itu, statement merujuk pada aksi. Sehingga, pada bagian kode tertentu yang membutuhkan nilai tidak bisa kita isi dengan sebuah statement. Contohnya seperti kode berikut:

1. let fullName = let lastName; // Error karena let lastName adalah sebuah statement untuk deklarasi variabel. Statement tidak bisa berada di posisi expression.
2. let fullName = (lastName = "Skywalker"); // (lastName = "Skywalker") merupakan expression, sehingga kode ini tidak error.
3. let fullName = "Luke" + "Skywalker"; // "Luke" + "Skywalker" juga merupakan expression, sehingga kode ini tidak error.

Melalui contoh kode di atas, kita bisa bayangkan variabel sebagai sebuah kotak atau wadah yang menyimpan nilai. Proses inisialisasi atau assignment berarti kita memasukkan nilai ke dalam kotak tersebut.



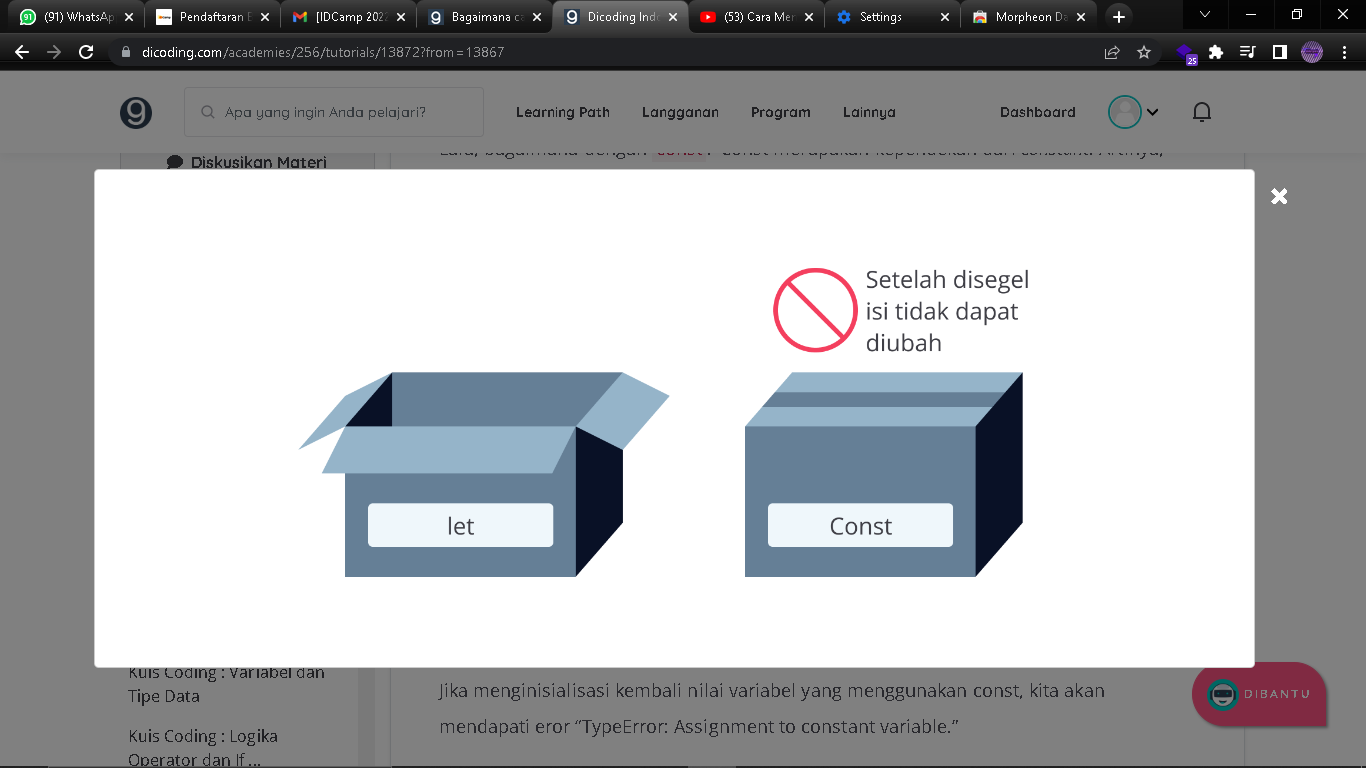
Variabel lastName di atas akan tersimpan di dalam memori komputer.

Setiap variabel memiliki nama yang dapat kita panggil dan gunakan. Kita dapat menamai variabel dengan nama apa pun, tetapi pastikan penamaannya masih masuk akal dengan konteksnya supaya kode mudah di-*maintenance*.

Sebaiknya hindari penamaan variabel dengan istilah umum seperti “data”. Gunakanlah penamaan variabel yang dapat mendeskripsikan nilai dari variabel itu sendiri. Berikut beberapa aturan dalam penamaan variabel yang perlu Anda ketahui:

* Harus dimulai dengan huruf atau underscore (\_).
* Dapat terdiri dari huruf, angka, dan underscore (\_) dengan berbagai kombinasi.
* Tidak boleh mengandung spasi (whitespace). Jika penamaan variabel lebih dari dua kata, tuliskan secara camelCase. Contoh firstName, lastName, catName, dll.
* Tidak boleh mengandung karakter spesial (! . , / \ + \* = dll.)

Lalu, bagaimana dengan const? Const merupakan kependekan dari *constant*. Artinya, kita akan mendeklarasikan sebuah variabel dengan const ketika ingin variabel bernilai konstan dan tidak bisa diubah setelah diinisialisasi nilainya. Bayangkan variabel bernilai const sebagai sebuah kotak yang ditutup dan disegel setelah diisi, sehingga nilainya tidak bisa diubah lagi.



Jika menginisialisasi kembali nilai variabel yang menggunakan const, kita akan mendapati eror “TypeError: Assignment to constant variable.”

const z = 100;

console.log(z);

z = 200;

console.log(z)

/\* TypeError: Assignment to constant variable. \*/

## Tipe Data

Pada materi sebelumnya kita telah belajar tentang variabel untuk menyimpan nilai. Nilai yang kita tetapkan pada variabel pasti memiliki tipe data. Tipe data merupakan pengklasifikasian data berdasarkan jenisnya. Pada JavaScript terdapat beberapa tipe data sebagai berikut:

### Undefined

Tipe data ini terbentuk ketika sebuah variabel tidak memiliki nilai. Artinya, ketika kita mendeklarasikan variabel tanpa menginisialisasikan nilainya, variabel tersebut menjadi undefined. Contoh:

let x;

console.log(typeof(x));

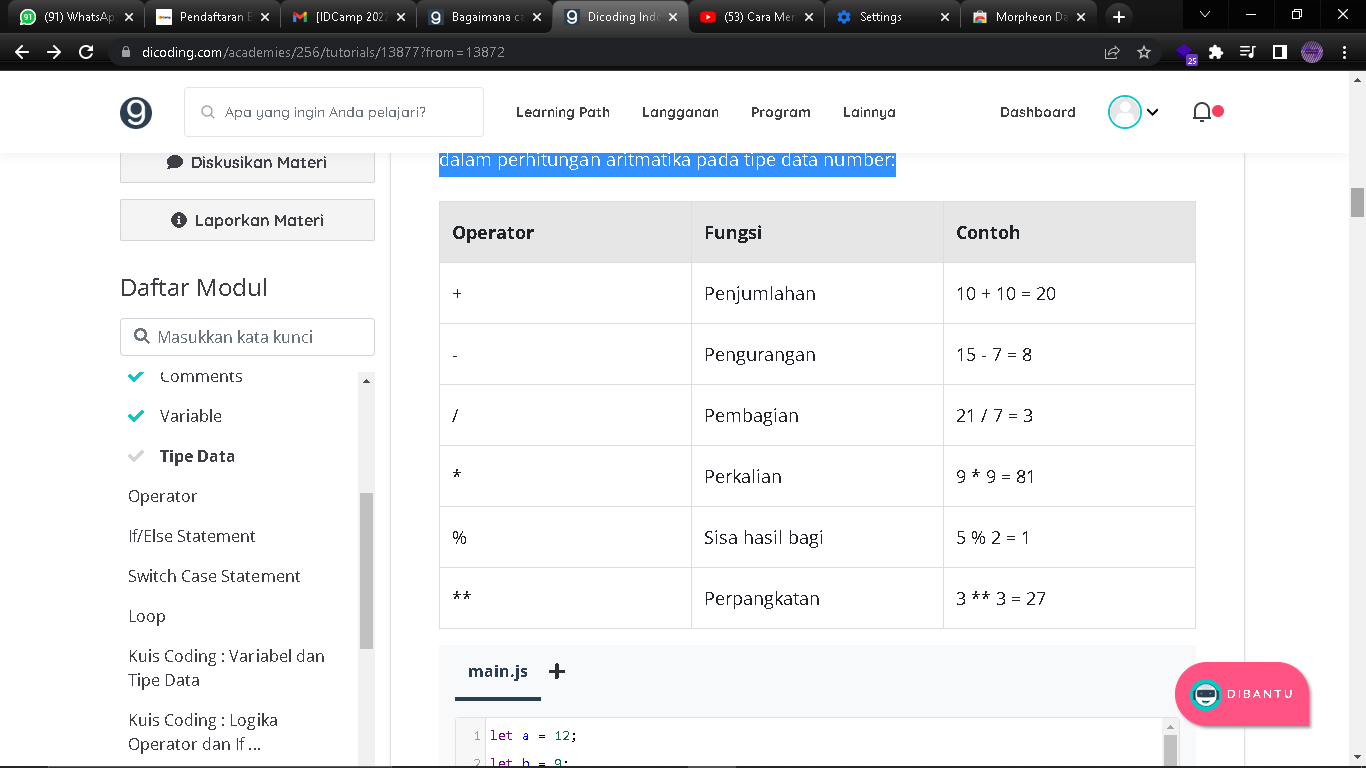
/\* output: undefined \*/

Fungsi typeof() digunakan untuk memastikan tipe data pada variabel dengan mengembalikan tipe data tersebut dalam bentuk teks.

### Numbers

Nilai dari tipe data number adalah angka. Variabel bertipe data number dituliskan seperti angka pada umumnya:

1. let x = 10;



let a = 12;

let b = 9;

console.log(a + b)

console.log(a - b)

console.log(a \* b)

console.log(a / b)

console.log(a % b)

/\* output:

21

3

108

1.3333333333333333

3

\*/

Pada operator aritmatika juga terdapat operator *increment* (++) dan *decrement* (--). Operator *increment* dan *decrement* digunakan untuk menambahkan atau mengurangi nilai 1 pada nilai variabel yang ada sekarang.

Operator ini dapat dituliskan sebelum atau sesudah variabel, tetapi hal tersebut bukan berarti sama. Berikut ketentuannya:

* Jika dituliskan setelah variabel (x++), *expression* akan menghasilkan nilai variabel sebelum ditingkatkan nilainya.
* Jika dituliskan sebelum variabel (++x), *expression* akan menghasilkan nilai variabel setelah ditingkatkan nilainya.

Untuk lebih jelasnya, berikut adalah contoh kode penerapan operator tersebut, perhatikan hasil yang didapat.

/\* Increment dan Decrement \*/

let postfix = 5;

console.log(postfix++);

/\* output: 5 \*/

console.log(postfix);

/\* output: 6 \*/

let prefix = 5;

console.log(++prefix);

/\* output: 6 \*/

### BigInt

Pada JavaScript, tipe data “Number” hanya mencakup nilai dari -(253 - 1) hingga (253 - 1). Untuk kebutuhan umum, sebenarnya nilai tersebut sudah sangat cukup. Namun, akan ada kebutuhan tertentu di mana kita membutuhkan cakupan nilai yang lebih besar, seperti untuk kriptografi atau menentukan waktu hingga presisi microsecond.

Untuk nilai di luar Number, kita bisa menggunakan tipe BigInt. Untuk membedakan tipe BigInt dan Number, tambahkan karakter n di akhir angka. Contohnya adalah seperti kode di bawah ini. Bandingkan dengan yang bertipe Number.

const bigNumber = 1234567890123456789012345678901234567890n;

const myInt = 1234567890123456789012345678901234567890;

console.log(bigNumber);

console.log(myInt);

/\* output

1234567890123456789012345678901234567890n

1.2345678901234568e+39 //(Ada huruf e+39 jika tidak menggunakan bigInt)

\*/

Meskipun digunakan untuk menyimpan angka dengan nilai besar, namun BigInt tetap bisa digunakan untuk nilai yang lebih kecil.

console.log(5n + 2n);

console.log(5n - 2n);

console.log(5n \* 2n);

console.log(5n / 2n);

console.log(5n % 2n);

/\* output

7n

3n

10n

2n; Bukan 2.5n

1n

\*/

### Strings

Tipe data selanjutnya adalah string yang merupakan sebuah teks. Untuk menetapkan nilai sebagai string pada variabel gunakan tanda petik satu (‘) atau petik dua (“) di antara teksnya. Contohnya:

let greet = "Hello";

console.log(typeof(greet))

/\* output: string \*/

Tidak ada perbedaan antara menggunakan petik satu atau petik dua. Anda dapat menggunakan tanda petik secara bergantian, khususnya jika Anda memiliki teks yang mengandung tanda petik.

const question = '"What do you think of JavaScript?" I asked';

console.log(question)

/\* output: "What do you think of JavaScript?" I asked \*/

Lalu bagaimana jika teks memiliki kedua tanda petik seperti ini?

const answer = '"I think it's awesome!" he answered confidently';

console.log(answer);

output:

/home/glot/main.js:1 const answer = '"I think it's awesome!" he answered confidently'; ^ SyntaxError: Unexpected identifier

Tentunya kode di atas akan menghasilkan eror. Solusinya, gunakan backslash(\) untuk mengurangi ambiguitas dalam tanda petik. Mekanisme ini juga dikenal dengan nama escape string. Sehingga kode di atas akan menjadi seperti berikut:

1. const answer = '"I think it\'s awesome!" he answered confidently';

Backslash sebelum tanda petik akan memberitahukan interpreter bahwa itu hanyalah tanda petik dan tidak boleh ditafsirkan sebagai pembatas string. Selain tanda petik, backslash juga berguna untuk mengabaikan simbol lain yang menimbulkan ambigu di dalam string, contohnya seperti backslash itu sendiri.

1. console.log("Windows path: C:\\Program Files\\MyProject");

Pada String, kita juga dapat menggunakan operator plus (+). Operator tersebut berfungsi untuk menggabungkan dua teks yang terpisah menjadi satu buah teks. Contohnya seperti ini:

let greet = "Hello";

let moreGreet = greet + greet;

console.log(moreGreet);

/\* output: HelloHello \*/

Ingat, string concatenation seperti di atas akan menggabungkan string apa adanya, sehingga jika Anda ingin menggabungkan dua kata atau lebih perlu menambahkan spasi sendiri.

Selain concatenation, string pada JavaScript juga mendukung string interpolation. Sederhananya, kita bisa memasukkan variabel ke dalam sebuah string template. Contohnya adalah seperti berikut:

const myName = "Luke";

console.log(`Hello, my name is ${myName}.`);

/\* output: Hello, my name is Luke. \*/

Perhatikan bahwa untuk mendefinisikan string template, Anda perlu menggunakan backticks (`), biasanya terletak di keyboard di bawah tombol Esc . Di dalam string letakkan variabel yang ingin dimasukkan ke dalam placeholder ${myName}.

### Boolean

Boolean hanya memiliki dua nilai, yaitu **true** atau **false**. Tipe data ini menjadi kunci utama dalam penentuan logika. Kita akan banyak menggunakannya nanti dalam materi if/else statement. Untuk menetapkan nilai boolean pada variabel, gunakan keyword true atau false seperti di bawah ini.

let x = true;

let y = false;

console.log(typeof(x))

console.log(typeof(y))

/\* output:

boolean

boolean

\*/

Kita juga bisa menggunakan operator komparasi seperti lebih dari (>) atau kurang dari (<). Contohnya:

const a = 10;

const b = 12;

let isGreater = a > b;

let isLess = a < b;

console.log(isGreater);

console.log(isLess);

/\* output:

false

true

\*/

### Null

Tipe berikutnya adalah null. Serupa dengan undefined, namun null perlu diinisialisasikan pada variabel. Null biasa digunakan sebagai nilai sementara pada variabel, tapi sebenarnya nilai tersebut “tidak ada”.

Terkadang kita perlu membuat sebuah variabel, namun kita belum memerlukan nilai apa-apa dan tidak ingin terikat oleh tipe data apa pun. Nah, daripada kita tidak menetapkan nilai apa pun (variabel akan undefined) sebaiknya kita beri nilai null pada variabel tersebut dan ubah nanti ketika kita membutuhkannya.

Untuk menetapkan null pada variabel, kita dapat gunakan keyword null ketika variabel tersebut diinisialisasi.

let someLaterData = null;

console.log(someLaterData);

/\* output:

null

\*/

### Symbol

Symbol adalah tipe data baru yang dikenalkan pada ES6. Tipe data Symbol digunakan untuk menunjukkan identifier yang unik. Ketika membuat Symbol, kita bisa memberikan deskripsi atau nama symbol seperti ini:

const id = Symbol("id");

console.log(id);

/\* output

Symbol(id)

\*/

Symbol disebut sebagai identifier yang unik karena meskipun kita membuat dua variabel symbol dengan nama atau deskripsi yang sama, kedua nilainya tetap dianggap berbeda. Contohnya lihat kode berikut:

const id1 = Symbol("id");

const id2 = Symbol("id");

console.log(id1);

console.log(id2);

console.log(id1 == id2);

/\* output

Symbol(id)

Symbol(id)

false\*/

Symbol ini umumnya digunakan sebagai nama property dari Object. Object sendiri merupakan tipe data kompleks untuk menyimpan berbagai struktur data. Kita akan segera bertemu dan mempelajari tentang object pada modul Data Structure.

## Operator

Pada materi ini kita akan mempelajari tentang operator yang terdapat pada JavaScript. Operator dalam bahasa pemrograman sendiri adalah simbol yang memberi tahu interpreter untuk melakukan operasi seperti matematika, relasional, atau logika untuk memberikan hasil tertentu.

### Assignment Operator

Dari contoh kode yang kita gunakan sebelumnya, sebenarnya kita sudah menggunakan assignment operator. Operator ini digunakan untuk memberikan nilai pada variabel.

Pada dasarnya operator ini adalah tanda sama dengan (=), di mana tanda ini digunakan untuk menginisialisasi nilai pada variabel. Tempatkan variabel yang ingin diberi nilai di sebelah kiri, sementara nilainya di sebelah kanan. Di antara keduanya terdapat operator assignment.

1. x = y;

Expression di atas berarti kita menginisialisasikan nilai y pada variabel x, sehingga nilai x sekarang memiliki nilai yang sama dengan y.

Ada beberapa assignment operator tambahan lain dalam menginisialisasikan nilai pada variabel. Kita bisa menyebutnya sebagai shortcut dalam menentukan nilai. Contohnya:

let x = 10;

let y = 5

x += y;

console.log(x);

/\* output

15

\*/

Pada contoh kode di atas, terdapat expression x += y; Apa artinya? Assignment operator tersebut digunakan sebagai shortcut dari x = x + y. Cara ini juga dapat digunakan pada operator aritmatika lain seperti, perkalian, pengurangan, pembagian, dan lainnya.

1. let x = 10;
2. let y = 5;
4. x += y; // artinya -> x = x + y;
5. x -= y; // artinya -> x = x - y;
6. x \*= y; // artinya -> x = x \* y;
7. x /= y; // artinya -> x = x / y;
8. x %= y; // artinya -> x = x % y;

### Comparison Operator

Sekarang kita sudah mengetahui bagaimana cara menyimpan nilai pada sebuah variabel. Nah, selanjutnya kita akan belajar mengenai operator komparasi sebagai logika dasar dalam membandingkan nilai pada JavaScript.

Terdapat serangkaian karakter khusus yang disebut dengan operator pembanding/komparasi yang dapat mengevaluasi dan membandingkan dua nilai. Berikut daftar operator dan fungsinya:

| **Operator** | **Fungsi** |
| --- | --- |
| == | Membandingkan kedua nilai apakah sama (tidak identik). |
| != | Membandingkan kedua nilai apakah tidak sama (tidak identik). |
| === | Membandingkan kedua nilai apakah identik. |
| !== | Membandingkan kedua nilai apakah tidak identik. |
| > | Membandingkan dua nilai apakah nilai pertama lebih dari nilai kedua. |
| >= | Membandingkan dua nilai apakah nilai pertama lebih atau sama dengan nilai kedua. |
| < | Membandingkan dua nilai apakah nilai pertama kurang dari nilai kedua. |
| <= | Membandingkan dua nilai apakah nilai pertama kurang atau sama dengan nilai kedua. |

Ketika kita melakukan perbandingan antara dua nilai, JavaScript akan mengevaluasi kedua nilai tersebut dan mengembalikan boolean dengan nilai hasil perbandingan tersebut, baik false atau true. Berikut contohnya:

let a = 10;

let b = 12;

console.log(a < b);

console.log(a > b);

/\* output

true

false

\*/

#### Perbedaan antara “Sama” dan “Identik”

Dalam operator komparasi di JavaScript, hal yang menjadi sedikit “tricky” adalah membedakan antara “sama” (==) dan “identik” (===).

Kita sudah mengetahui bahwa setiap nilai pasti memiliki tipe data baik itu number, string atau boolean. Contohnya sebuah string “10” dan number 10 merupakan hal yang serupa, tetapi keduanya tidak benar-benar sama.

Hal inilah yang membedakan antara sama dan identik pada JavaScript. Jika kita ingin membandingkan hanya dari kesamaan nilainya kita bisa gunakan == tapi jika kita ingin membandingkan dengan memperhatikan tipe datanya kita gunakan ===.

Contohnya seperti berikut:

const aString = '10';

const aNumber = 10

console.log(aString == aNumber) // true, karena nilainya sama-sama 10

console.log(aString === aNumber) // false, karena walaupun nilainya sama, tetapi tipe datanya berbeda

/\* output

true

false

\*/

### Logical Operator

Terdapat beberapa operator lain yang dapat kita gunakan untuk menetapkan logika yang lebih kompleks, yakni dengan logical operators. Dengan logical operator, kita dapat menggunakan kombinasi dari dua nilai boolean atau bahkan lebih dalam menetapkan logika.

Pada JavaScript terdapat tiga buah karakter khusus yang berfungsi sebagai logical operator. Berikut macam-macam logical operator dan fungsinya:

| **Operator** | **Deskripsi** |
| --- | --- |
| && | Operator dan (and). Logika akan menghasilkan nilai true apabila semua kondisi terpenuhi (bernilai true). |
| || | Operator atau (or). Logika akan menghasilkan nilai true apabila ada salah satu kondisi terpenuhi (bernilai true). |
| ! | Operator tidak (not). Digunakan untuk membalikkan suatu kondisi. |

Berikut contoh penerapannya pada JavaScript:

let a = 10;

let b = 12;

/\* AND operator \*/

console.log(a < 15 && b > 10); // (true && true) -> true

console.log(a > 15 && b > 10); // (false && true) -> false

/\* OR operator \*/

console.log(a < 15 || b > 10); // (true || true) -> true

console.log(a > 15 || b > 10); // (false || true) -> true

/\* NOT operator \*/

console.log(!(a < 15)); // !(true) -> false

console.log(!(a < 15 && b > 10)); // !(true && true) -> !(true) -> false

/\* output

true

false

true

true

false

false

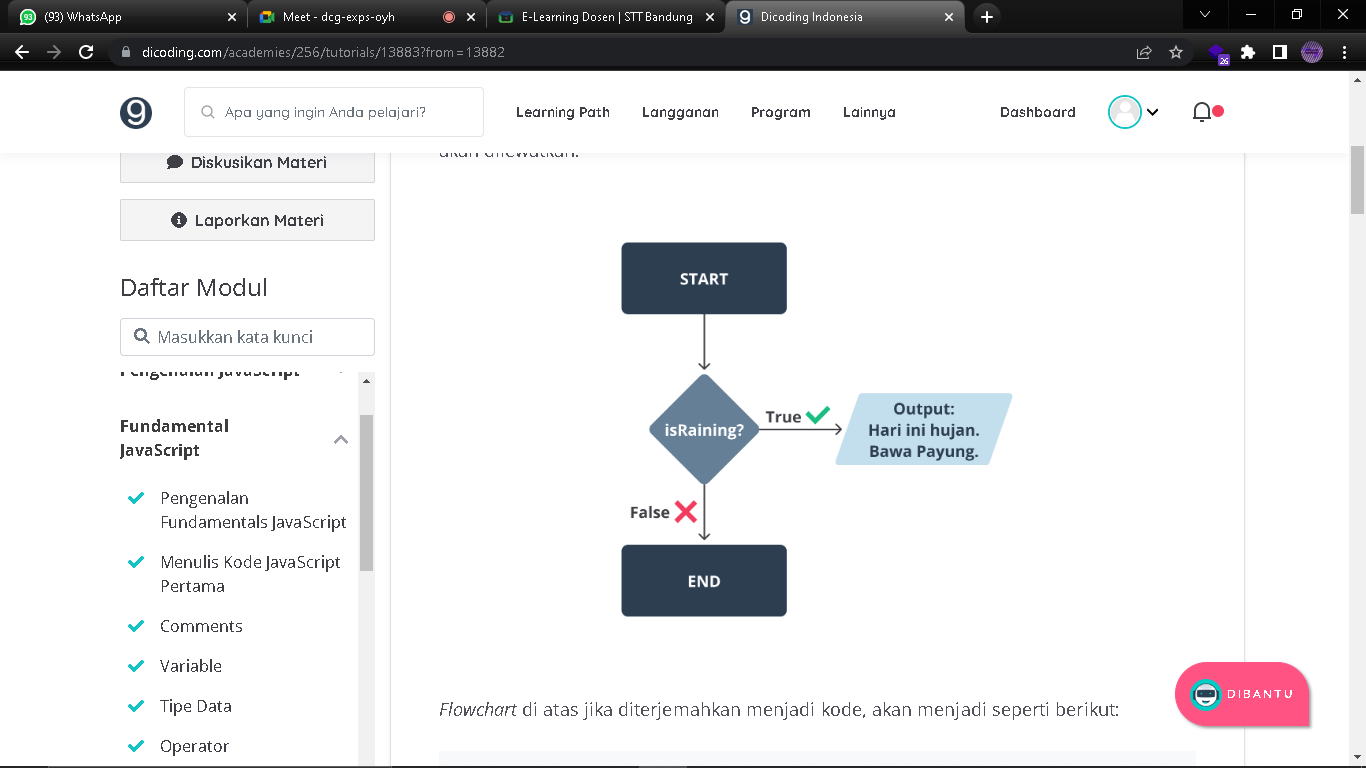
\*/

## If/Else Statement

Setiap hari kita melakukan perhitungan dan perbandingan guna membuat keputusan, apa pun itu. Contohnya, apakah perlu mencuci kendaraan ketika cuaca sedang cerah? Apa saja transportasi online yang bisa dipesan ketika hujan untuk sampai di tempat tujuan?

Ketika mengembangkan sebuah program, kita akan bertemu dengan alur bercabang tergantung pada kondisi yang terjadi. Untuk mengakomodasi dan mengecek sebuah kondisi dalam JavaScript, kita menggunakan kata kunci **if**.

Statement if akan menguji suatu kondisi. Jika kondisi bernilai true, maka blok kode di dalamnya akan dijalankan. Sebaliknya, jika bernilai false, maka proses yang ditentukan akan dilewatkan.



Flowchart di atas jika diterjemahkan menjadi kode, akan menjadi seperti berikut:

const isRaining = true; // bisa juga jika diisi dengan nilai string atau number, namun tidak dapat diisi null atau undefined

console.log("Persiapan sebelum berangkat kegiatan.");

if (isRaining) {

console.log("Hari ini hujan. Bawa payung.");

}

console.log("Berangkat kegiatan.");

/\* output:

Persiapan sebelum berangkat kegiatan.

Hari ini hujan. Bawa payung.

Berangkat kegiatan.

\*/

Jika Anda mengubah nilai isRaining menjadi false, maka kode di dalam blok kode if akan dilewatkan. Sehingga program Anda tidak akan mengingatkan untuk membawa payung.

Lalu bagaimana jika Anda ingin melakukan operasi lain ketika kondisi bernilai false? Jawabannya adalah statement else. Pada contoh kode berikut kita akan melakukan pengecekan kondisi menggunakan operator perbandingan.

let x = 50;

if(x > 70) {

console.log(x);

} else {

console.log("Nilai kurang dari 70");

}

/\* output

Nilai kurang dari 70

\*/

Terdapat variabel x dengan nilai 50, kemudian kita bertanya, “Hai JavaScript! Apakah x lebih dari 70?” Jika kondisi tersebut benar, maka kita dapat memerintahkan JavaScript untuk menampilkan nilainya. Jika salah, kita perintahkan JavaScript untuk menampilkan teks “Nilai kurang dari 70”.

Kita juga bisa mengecek beberapa kondisi sekaligus dengan menggabungkan else dan if. Contohnya adalah seperti program berikut:

let language = "French";

let greeting = "Selamat Pagi"

if(language === "English") {

greeting = "Good Morning!";

} else if(language === "French") {

greeting = "Bonjour!"

} else if(language === "Japanese") {

greeting = "Ohayou Gozaimasu!"

}

console.log(greeting);

/\* output

Bonjour!

\*/

Pengecekan kondisi akan dilakukan dari statement if paling awal. Sehingga, ketika nilai language adalah “French”, maka pengecekan untuk language === “Japanese” tidak akan dilakukan.

Selain if statement di atas, JavaScript juga mendukung ternary operator atau conditional expressions. Dengan ini, kita bisa menuliskan if-else statement hanya dalam satu baris.

// condition ? true expression : false expression

const isMember = false;

const discount = isMember ? 0.1 : 0;

console.log(`Anda mendapatkan diskon sebesar ${discount \* 100}%`)

/\* output

Anda mendapatkan diskon sebesar 0%

\*/

Ternary operator membutuhkan tiga operand. Sebelum tanda tanya (?) berisi kondisi yang ingin kita evaluasi. Kemudian diikuti dengan expression apabila nilai kondisinya benar sebelum tanda titik dua. Terakhir adalah expression yang dieksekusi ketika kondisinya salah. Karena merupakan conditional expression, maka operand kedua dan ketiga harus mengembalikan nilai.

### Truthy & Falsy

Di dalam if statement kita perlu memasukkan expression yang akan dievaluasi. Umumnya, expression tersebut mengembalikan nilai boolean untuk menentukan kondisi true atau false. Lalu bagaimana jika kita menuliskan expression yang tidak mengembalikan nilai boolean? Jawabannya bisa.

Setiap nilai pada JavaScript pada dasarnya juga mewarisi sifat boolean. Nilai ini dikenal dengan truthy atau falsy. Nilai truthy berarti nilai yang ketika dievaluasi akan menghasilkan nilai true, begitu pula falsy bernilai false. Jadi manakah yang termasuk truthy dan falsy? Selain nilai boolean false, tipe data atau nilai yang dianggap falsy, antara lain:

* Number 0
* BigInt 0n
* String kosong seperti “” atau ‘’
* null
* undefined
* NaN, atau Not a Number

Selain contoh di atas maka nilainya adalah truthy dan ketika dievaluasi ke dalam if statement akan bernilai true. Berikut ini contohnya dalam kode:

let name = "";

if (name) {

console.log(`Halo, ${name}`);

} else {

console.log("Nama masih kosong");

}

/\* output:

Nama masih kosong

\*/

## Switch Case Statement

Sebelumnya kita telah mempelajari bagaimana percabangan logika menggunakan if statement. Selain if, JavaScript juga mendukung switch statement untuk melakukan pengecekan banyak kondisi dengan lebih mudah dan ringkas.

1. switch (expression) {
2. case value1:
3. // do something
4. break;
5. case value2:
6. // do something
7. break;
8. ...
9. ...
10. default:
11. // do something else
12. }

let language = "French";

let greeting = null;

switch (language) {

case "English":

greeting = "Good Morning!";

break;

case "French":

greeting = "Bonjour!";

break;

case "Japanese":

greeting = "Ohayou Gozaimasu!";

break;

default:

greeting = "Selamat Pagi!";

}

console.log(greeting);

/\* output

Bonjour!

\*/

## Loop

Ketika menulis program komputer, akan ada situasi di mana kita perlu melakukan hal yang sama berkali-kali. Misalnya kita ingin menampilkan semua nama pengguna yang terdaftar di aplikasi atau sesederhana menampilkan angka 1 sampai 10. Tentunya tidak praktis jika kita menulis kode seperti berikut:

1. console.log(1);
2. console.log(2);
3. console.log(3);
4. console.log(4);
5. console.log(5);
6. console.log(6);
7. console.log(7);
8. console.log(8);
9. console.log(9);
10. console.log(10);

Bagaimana jika kita perlu menampilkan angka 1 sampai 100? Sudah terbayang repotnya, bukan? Maka dari itu kita perlu mempelajari teknik yang dapat mengatasi permasalahan tersebut, teknik ini disebut dengan looping. JavaScript memiliki banyak opsi untuk melakukan looping atau perulangan kode, antara lain:

Dari beberapa cara melakukan proses loop pada JavaScript, “for” merupakan salah satu cara yang banyak digunakan. Struktur dasar dari for tampak seperti berikut:

1. for(inisialisasi variabel; test kondisi; perubahan nilai variabel) {
2. // do something
3. }

Berikut ini contoh penerapannya secara nyata:

for(let i = 0; i < 5; i++) {

console.log(i);

}

/\* output

0

1

2

3

4

Lebih ringkas, bukan? Mungkin kode tersebut sulit dipahami, jadi mari kita bahas sedikit demi sedikit. Terdapat tiga bagian utama dalam sintaks for di atas:

* Pertama, variabel i sebagai *index* iterasi. Pada variabel ini kita menginisialisasi nilai awal dari perulangan.
* Kedua, operasi perbandingan. Pada bagian ini, JavaScript akan melakukan pengecekan kondisi apakah perulangan masih perlu dilakukan. Jika bernilai true, maka kode di dalam blok for akan dijalankan.
* Ketiga, increment/decrement. Di sini kita melakukan penambahan atau pengurangan variabel iterasi. Jadi, pada contoh di atas variabel i akan ditambah dengan 1 di setiap akhir perulangan. Perubahan nilai ini penting karena jika kita mengubah nilainya, proses perulangan dapat terus berjalan selama kondisinya terpenuhi.

JIka diartikan, maka kode di atas bisa dimaknai dengan “Jika i kurang dari 5, maka tampilkan nilai i.”

### For of loop

Cara lain dalam melakukan looping adalah menggunakan for..of. For of mulai hadir pada ECMAScript 2015 (ES6). Cara ini jauh lebih sederhana dan modern dibanding for loop biasa. Sintaks dasar dari for of loop adalah seperti ini:

1. for(arrayItem of myArray) {
2. // do something
3. }

Yup, for of tidak membutuhkan banyak statement untuk melakukan looping pada array. Penjelasan tentang array akan lebih detail dibahas pada modul berikutnya. Sebagai permulaan, kita bisa menganggap array sebagai kumpulan nilai yang disimpan dalam satu variabel.

Dengan for..of nilai tiap array akan diinisialisasi pada variabel baru yang kita tentukan pada tiap proses looping-nya. Jumlah proses looping-nya pun akan menyesuaikan dengan ukuran dari array. Sederhananya seperti kita melakukan perintah “Hei JavaScript! Lakukan perulangan pada myArray, akses tiap nilainya, dan simpan pada variabel arrayItem”. Pada proses looping kita gunakan variabel arrayItem untuk mengakses tiap nilai dari item myArray.

Agak sulit memang menjelaskan dengan kata-kata, mari kita terjemahkan dalam kode secara langsung.

let myArray = ["Luke", "Han", "Chewbacca", "Leia"];

for(const arrayItem of myArray) {

console.log(arrayItem)

}

/\* output

Luke

Han

Chewbacca

Leia

\*/

### While and do-while

Metode lain untuk melakukan looping adalah dengan statement while. Sama seperti for, instruksi while mengevaluasi ekspresi boolean dan menjalankan kode di dalam blok while ketika bernilai true.

Untuk menampilkan angka 1 sampai 100 dengan while kita bisa menulis kode seperti berikut:

let i = 1;

while (i <= 100) {

console.log(i);

i++;

}

Bisa dilihat pada kode di atas bahwa looping dengan while tidak memiliki ketergantungan dengan variabel iterasi seperti pada for loop. Karena itu, meskipun while dapat melakukan perulangan yang sama dengan for, while lebih cocok digunakan pada kasus di mana kita tidak tahu pasti berapa banyak perulangan yang diperlukan.

Bentuk lain dari while adalah perulangan do-while.

let i = 1;

do {

console.log(i);

i++;

} while (i <= 100);

Kondisi pada while akan dievaluasi sebelum blok kode di dalamnya dijalankan, sedangkan do-while akan mengevaluasi boolean expression setelah blok kodenya berjalan. Ini artinya kode di dalam do-while akan dijalankan setidaknya satu kali.

### Infinite loops

Ketika menerapkan perulangan pada program, ada satu kondisi yang perlu kita hindari yaitu infinite loop. Infinite loop atau endless loop adalah kondisi di mana program kita stucked di dalam perulangan. Ia akan berjalan terus hingga menyebabkan crash pada aplikasi dan komputer kecuali ada intervensi secara eksternal, seperti mematikan aplikasi.

Kode berikut ini adalah contoh di mana kondisi infinite loop dapat terjadi:

While

1. let i = 1;
3. while (i <= 5) {
4. console.log(i);
5. }

for

1. for(let i = 1; i <= 5; i=1) {
2. console.log(i);
3. }

Dapatkah Anda menemukan apa yang salah dari kode di atas sehingga terjadi infinite loop?

Jawabannya adalah karena variabel i selalu bernilai 1. Alhasil, kondisi i <= 5 akan selalu bernilai true yang mengakibatkan aplikasi akan terus mencetak 1 ke konsol sehingga mengalami crash.

## Rangkuman Materi

Pada modul ini kita telah mempelajari logika dan sintaksis dasar JavaScript sebelum membuat aplikasi yang lebih kompleks ke depannya.

Beberapa hal yang telah kita bahas pada modul ini, antara lain:

* Gunakan comments untuk memberitahu interpreter supaya mengabaikan kode atau teks yang kita tulis. Ini akan berguna untuk membuat dokumentasi atau penjelasan atas kode yang kita tulis.
* Kita dapat menyimpan suatu nilai ke dalam variabel. ES6 mengenalkan dua cara baru untuk mendefinisikan variabel, yaitu let dan const. Gunakan const untuk menyimpan nilai yang tidak akan berubah setelah diinisialisasi. Gunakan let apabila nilai di dalam variabel bisa berubah atau diinisialisasi ulang.
* Terdapat tujuh (7) tipe data primitif yang mendefinisikan suatu nilai dalam JavaScript. Ketujuh nilai tersebut antara lain: Undefined, Number, BigInt, String, Boolean, Null, dan Symbol.
* JavaScript memiliki beragam operator yang memberi tahu interpreter untuk melakukan operasi matematika, relasional, atau logika untuk memberikan hasil akhir.
* Pengambilan keputusan adalah hal yang penting dalam pemrograman. Kita bisa memanfaatkan if-else statement atau switch-case untuk memilih satu opsi berdasarkan kondisi yang diberikan.
* Pemrograman juga membantu kita untuk melakukan pekerjaan yang berulang. Dengan kode for atau while, kita bisa melakukan perulangan terhadap suatu kode sebanyak ratusan bahkan ribuan kali hanya dengan beberapa baris saja.