Chapter 2. JavaScript Foundation

# JavaScript Variable

## Deklarasi Variable

Cara mendeklarasikan sebuah variabel adalah dengan kata kunci let atau const dan diikuti dengan nama variabelnya, seperti ini:

let namaVariabel1 = "hello";

const namaVariabel2 = "world";

Dalam menamakan variabel, kalian pun perlu mengikuti beberapa ketentuan sebagai berikut:

1. Menggunakan konvensi camelCase, yaitu menggunakan huruf kecil pada permulaan kata, dan jika nama variabel terdiri dari 2 kata atau lebih, maka kata kedua dan berikutnya diawali dengan huruf besar.
2. Nama variabel boleh diawali dengan underscore(\_), huruf besar (jika diharuskan), dan $.
3. Nama variabel tidak diawali dengan angka, @, \*, #, !, %, (), -, +.
4. Tidak boleh menggunakan kata kunci JavaScript seperti boolean, break, else, extends, dan lain-lain.
5. Antara kata pertama dan kata kedua tidak diberi spasi atau dash(-);
6. Karena JavaScript bersifat case-sensitive, maka penamaan juga case-sensitive (jadi variabel hitung dengan variabel Hitung itu berbeda).

Contoh:

// Contoh penulisan nama variabel yang benar

let nama;

let tempatTinggal;

let \_nomorHandphone;

let $nomorAkunBank;

// Contoh penulisan nama variabel yang salah

let makanan pembuka; //tidak boleh ada spasi

let @username; //tidak boleh diawali dengan @

let delete; //delete itu merupakan kata kunci di JavaScript

## Tipe Data JavaScript

Sebuah variabel bisa kita isi dengan suatu data. Caranya adalah dengan menggunakan = setelah nama variabel, lalu diikuti dengan datanya.

Contohnya seperti ini:

let nama = "Stefanus";

const namaIbuKandung = "Clara";

const umur = 20;

const apakahSudahMenikah = false;

let jumlahAnak;

Pada contoh di atas, variabel nama bernilai Stefanus dan tipe datanya adalah string. Tipe data string memiliki ciri khas yaitu nilai data dibungkus dengan tanda ' atau ".

Variabel namaIbuKandung juga memiliki nilai yang bertipe data string Clara. Lalu apa yang membedakan kedua variabel tersebut?

Variabel yang dideklarasikan menggunakan kata kunci let dapat diubah nilainya. Misalnya kita ingin mengubah variabel nama tadi menjadi Albert, kita cukup memberinya data tersebut seperti ini:

// memberi nilai baru kepada variabel nama

nama = "Albert";

Sementara variabel yang dideklarasikan menggunakan kata kunci const tidak dapat diubah nilainya; sudah konstan. Jika kita tetap mencoba untuk memberinya nilai seperti di bawah ini, akan muncul error.

// memberi nilai baru kepada variabel namaIbuKandung tadi

namaIbuKandung = "Juliana";

// akan muncul error

Uncaught TypeError: Assignment to constant variable.

#### 📝Catatan:

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

Kembali ke pembahasan tipe data. Variabel umur memiliki nilai 20 yang mana termasuk tipe data number. Tipe data ini bisa berupa:

* integer (seperti 1, 2, dan 3)
* atau float (seperti 3.14, 2.718, dan 1.618)

Selanjutnya, variabel apakahSudahMenikah bernilai false dengan tipe data boolean.

Tipe boolean ini terdiri dari 2 nilai:

* true yang bernilai benar
* false yang bernilai salah

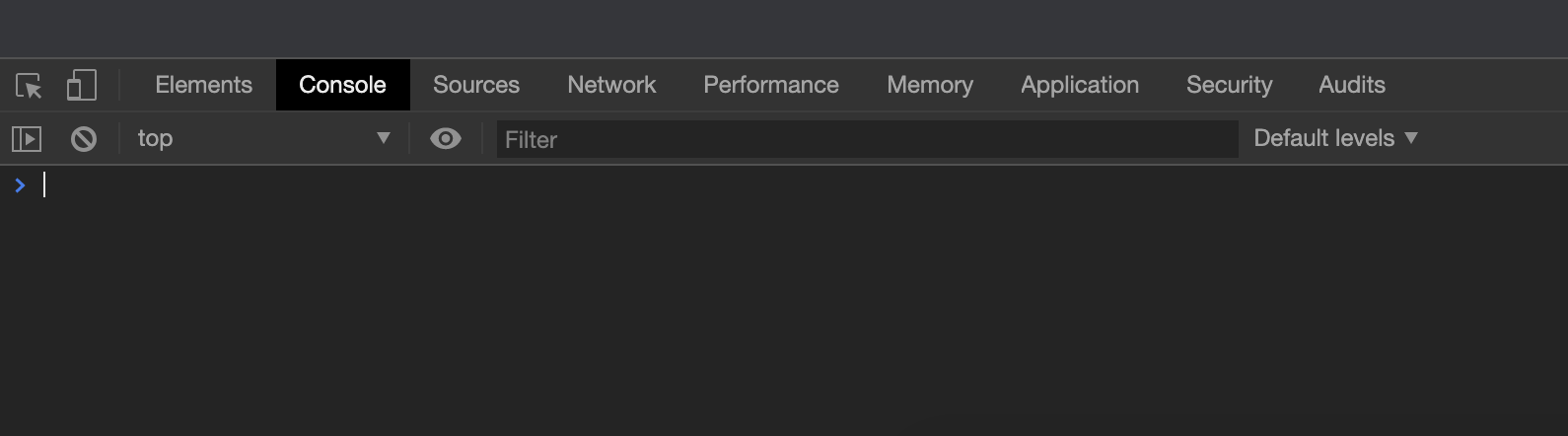
Terakhir, variabel jumlahAnak bernilai undefined, karena memang tidak diisi dengan nilai apapun. Jika variabel tidak kita beri sebuah nilai, secara default (standar) ia memiliki nilai undefined.

Untuk merangkum, berikut adalah 5 tipe data primitif yang ada dalam JavaScript:

* string - deretan karakter yang diapit oleh sepasang tanda kutip;
* number - bilangan bulat, pecahan, dan lain-lain;
* boolean - nilai benar dari sebuah pernyataan yang dituliskan sebagai true atau false;
* null - sebuah nilai yang berarti kosong atau menunjuk pada nilai yang tidak ada;
* undefined - berbeda dari null, undefined menandakan kondisi variabel yang belum diberi sebuah nilai. Jadi pernyataan "nilai variabel itu adalah undefined" sebenarnya kurang tepat, sebab variabelnya memang tidak mempunyai sebuah nilai;

## Menampilkan Data di Console

Browser pada umumnya mempunyai tab console yang bisa digunakan untuk menampilkan data console.log dari kode JavaScript. Untuk mengaktifkan tab console di Chrome, bisa dilakukan dengan pilih menu View -> Developer -> Developer Tools atau dengan tekan tombol F12 atau kombinasi tombol Ctrl-Shift-I.



Kenalan dulu yuk dengan console.log. Apa itu console.log dan apakah kegunaannya?

console.log adalah salah satu syntax JavaScript untuk menampilkan nilai suatu variabel atau hasil operasi di tab console.

Contoh:

let nama = "Budi";

let umur = 20;

let apakahSudahMenikah = false;

let jumlahAnak;

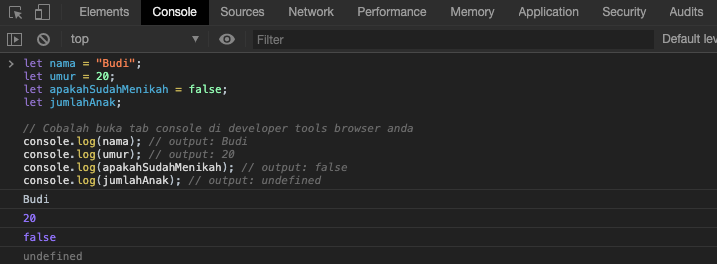
// Cobalah buka tab console di developer tools browser anda

console.log(nama); // output: Budi

console.log(umur); // output: 20

console.log(apakahSudahMenikah); // output: false

console.log(jumlahAnak); // output: undefined



Untuk apa kita menggunakan console.log?

Dengan console.log, output dari variabel atau operasi yang kita masukkan tidak akan kelihatan di halaman website dan hanya bisa tampil di tab console. Dengan begitu, output tersebut tidak akan mengganggu tampilan website kamu.

console.log juga sering digunakan developer ketika mau debug/investigasi kesalahan di kode yang ditulis.

# JavaScript Operators

## Arithmetic Operators

Operator aritmatika digunakan di operasi matematika yang melibatkan data dengan tipe number.

Operator-operator tersebut dapat dilihat pada tabel berikut ini:

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

Contoh penggunaan pada operasi matematika JavaScript:

### Penjumlahan

let bilangan1 = 10;

let bilangan2 = 3;

console.log(bilangan1 + bilangan2); // Output: 13

### Pengurangan

let bilangan1 = 10;

let bilangan2 = 3;

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

### Perkalian

let bilangan1 = 10;

let bilangan2 = 3;

console.log(bilangan1 \* bilangan2); // Output: 30

### Pembagian

let bilangan1 = 10;

let bilangan2 = 3;

console.log(bilangan1 / bilangan2); // Output: 3.3333333333333335

### Eksponen (Pangkat)

let bilangan1 = 10;

let bilangan2 = 3;

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

### Modulus

let bilangan1 = 10;

let bilangan2 = 3;

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

### Increment (Tambah 1)

let bilangan = 10;

Bilangan++;

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

Contoh operasi di atas apabila dituliskan tanpa operator increment akan sama seperti ini:

let bilangan = 10;

bilangan = bilangan + 1;

console.log(bilangan); // output: 11

### Decrement (Kurang 1)

let bilangan = 10;

bilangan--;

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

Contoh operasi di atas apabila dituliskan tanpa operator decrement akan sama seperti ini:

let bilangan = 10;

bilangan = bilangan - 1;

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

## Assignment Operator

Assignment operator digunakan untuk memberikan nilai pada variabel.

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

Contoh penggunaan di JavaScript:

### Assignment =

let nama = "Budi";

let umur = 20;

let apakahSudahMenikah = false;

let jumlahAnak;

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

console.log(umur); // Output: 20

console.log(apakahSudahMenikah); // Output: false

console.log(jumlahAnak); // Output: undefined

### Addition Assignment +=

let bilangan = 10;

bilangan += 5;

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

### Subtraction Assignment -=

let bilangan = 10;

bilangan -= 5;

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

### Multiplication Assignment \*=

let bilangan = 10;

bilangan \*= 5;

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

### Division Assignment /=

let bilangan = 10;

bilangan /= 5;

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

### Modulus Assignment %=

let bilangan = 10;

bilangan %= 5;

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

### Exponent Assignment \*\*=

let bilangan = 10;

bilangan \*\*= 5;

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

## String Operators

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

Ada 2 macam string operator, yaitu:

* +
* +=

Contoh penggunaan:

### operator +

let namaDepan = "Albert";

let namaBelakang = "Einstein";

console.log(namaDepan + " " + namaBelakang); // Output: Albert Einstein

### operator +=

let kata = "Halo ";

kata += "Dunia";

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

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

Contoh:

let kata = "Kambing ada ";

let bilangan = 5;

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

## Relational Operators

Operator perbandingan digunakan untuk membandingkan dua data atau nilai.

| **Operator** | **Deskripsi** |
| --- | --- |
| == | sama dengan (cek nilai) |
| === | sama dengan (cek nilai dan tipe data) |
| != | tidak sama dengan (cek nilai) |
| !== | tidak sama dengan (cek nilai dan tipe data) |
| > | lebih dari |
| < | kurang dari |
| >= | lebih dari atau sama dengan |
| <= | kurang dari atau sama dengan |
| ? : | ternary operator |

Contoh penggunaan di JavaScript:

### Operator == (Loose Equality)

let bilangan = 10;

console.log(bilangan == 10); // Output: true

console.log(bilangan == 8); // Output: false

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

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

#### 📝Catatan:

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

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

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

Kenapa kode di atas menghasilkan output true?

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

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

### Operator === (Strict Equality)

let bilangan = 10;

console.log(bilangan === 10); // Output: true --> nilai dan tipe data sama

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

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

### Operator !=

let bilangan = 10;

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

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

console.log(bilangan != 10); // Output: false --> nilai dan tipe data sama

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

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

### Operator !==

let bilangan = 10;

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

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

console.log(bilangan !== 10); // Output: false --> nilai dan tipe data sama

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

Kali ini nilai dan tipe data akan dicek. Operator ini hanya akan mengeluarkan output false apabila nilai dan tipe data dari kedua operand sama.

### Operator >(Lebih Dari)

let bilangan = 10;

console.log(bilangan > 10); // Output: false

console.log(bilangan > 11); // Output: false

console.log(bilangan > 8); // Output: true

### Operator < (Kurang Dari)

let bilangan = 10;

console.log(bilangan < 10); // Output: false

console.log(bilangan < 8); // Output: false

console.log(bilangan < 11); // Output: true

### Operator >=(Lebih Dari atau Sama Dengan)

let bilangan = 10;

console.log(bilangan >= 10); // Output: true

console.log(bilangan >= 8); // Output: true

console.log(bilangan >= 11); // Output: false

### Operator <= (Kurang Dari atau Sama Dengan)

let bilangan = 10;

console.log(bilangan <= 10); // Output: true

console.log(bilangan <= 11); // Output: true

console.log(bilangan <= 8); // Output: false

### Operator Ternary

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

Syntaxnya adalah:

variabel = (kondisi true) ? nilai1 : nilai2;

// Artinya apabila kondisi true, maka variabel akan diberi nilai1

// Apabila kondisi false, maka variabel akan diberi nilai2

Contoh:

let makanan = "daging";

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

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

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

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

## Operator Logika

Operator logika digunakan untuk menentukan logika antara dua kondisi atau nilai.

| **Operator** | **Deskripsi** |
| --- | --- |
| && | AND (Dan) |
| || | OR (Atau) |
| ! | NOT (Bukan) |

Syntax dari operator logika di JavaScript itu seperti ini:

### Operator && (AND)

kondisi1 && kondisi2

Operator ini hanya akan menghasilkan output true apabila kedua operand-nya bernilai true (kondisi1 dan kondisi2 bernilai true).

### Operator || (OR)

kondisi1 || kondisi2

Operator ini akan menghasilkan output true apabila salah satu dari operand-nya bernilai true (kondisi1 bernilai true atau kondisi2 bernilai true).

### Operator ! (NOT)

!kondisi

Operator ini akan bernilai true apabila operand-nya bernilai false (negation). Dan sebaliknya, operator akan mengembalikan false apabila operand-nya bernilai true.

### Tabel Perbandingan

Jika kedua kondisi bertipe boolean, maka hasil dari masing-masing operator logika akan seperti ini:

| **Operator** | **Kondisi 1** | **Kondisi 2** | **Hasil** |
| --- | --- | --- | --- |
| && | true | true | true |
| && | true | false | false |
| && | false | true | false |
| && | false | false | false |
| || | true | true | true |
| || | true | false | true |
| || | false | true | true |
| || | false | false | false |
| ! | true | - | false |
| ! | false | - | true |

Contoh penggunaan:

let bilangan1 = 6;

let bilangan2 = 4;

console.log(bilangan1 === 6 && bilangan2 === 4); // Output: true

console.log(bilangan1 === 6 && bilangan2 < 4); // Output: false

console.log(bilangan1 < 6 && bilangan2 === 4); // Output: false

console.log(bilangan1 < 6 && bilangan2 < 4); // Output: false

console.log(bilangan1 === 6 || bilangan2 === 4); // Output: true

console.log(bilangan1 === 6 || bilangan2 < 4); // Output: true

console.log(bilangan1 < 6 || bilangan2 === 4); // Output: true

console.log(bilangan1 < 6 || bilangan2 < 4); // Output: false

console.log(!true); // Output: false

console.log(!false); // Output: true

# JavaScript Function

Apa yang dimaksud dengan fungsi (function) di JavaScript?

Di konsep JavaScript, fungsi bisa diilustrasikan sebagai berikut:

Seandainya kita mempunyai sebuah mesin pembuat kue. Kita bisa memasukkan bahan baku seperti tepung terigu, telur, gula, air, dan bahan lainnya ke dalam mesin tersebut, kemudian mesin itu akan mengolah bahan-bahan tersebut yang akhirnya akan menghasilkan kue.

Nah, dari gambaran tersebut, mesin pembuat kue adalah fungsi (function). Tepung terigu, telur, gula, air, dan bahan-bahan lain untuk membuat kue adalah input (parameter) dari sebuah fungsi. Kemudian kue yang dihasilkan dari proses pengolahan mesin kue adalah hasil (return value).

## Function Declaration & Function Expression

Fungsi dapat *berdiri sendiri* atau *disimpan* di dalam sebuah variabel.

Cara mendeklarasikannya seperti ini:

// Fungsi yang berdiri sendiri

function namaFungsi(){

// Kode yang akan dijalankan

}

// Fungsi yang disimpan di dalam variabel

let namaVariabelFungsi = function(){

// Kode yang akan dijalankan

}

Fungsi dapat dipanggil cukup dengan menambahkan tanda kurung () setelah nama fungsi tersebut. Dalam dua contoh di atas, pemanggilan fungsinya adalah sebagai berikut:

namaFungsi();

namaVariabelFungsi();

Contoh:

// Fungsi yang berdiri sendiri

function sapa() {

return "Selamat Pagi!";

}

console.log(sapa()); // Output: Selamat Pagi!

// Fungsi disimpan di dalam variabel

let berkenalan = function() {

return "Hallo, nama saya John.";

};

console.log(berkenalan()); // Output: Hallo, nama saya John.

Pada contoh kedua di atas, return value-nya adalah "Hallo, nama saya John.". Bedakan return dengan console.log() ya.

console.log() hanya menampilkan informasi ke dalam tab console JavaScript, sedangkan return akan mengembalikan sebuah nilai ke tempat di mana fungsi itu dipanggil.

Kalau bingung coba lihat contoh kode berikut:

console.log(berkenalan() + " nama kamu siapa?")

Kalau kita lihat fungsi berkenalan() tadi, return valuenya adalah "Hallo, nama saya John.". Nilai dari return ini akan dikembalikan ke tempat pemanggilan fungsi tersebut, sehingga kode di atas saat dijalankan sebenarnya akan menjadi:

console.log("Hallo, nama saya John." + " nama kamu siapa?")

Lalu bagaimana jika kita tidak menulis perintah return?

Misalnya seperti contoh di bawah ini:

let umur = 21;

function ulangTahun() {

umur += 1;

}

console.log(ulangTahun()); // Output: undefined

Fungsi tersebut tidak memiliki perintah return. Di JavaScript, apabila tidak ada perintah return, secara default (standar) fungsi tersebut akan mengembalikan nilai undefined.

## Parameter & Argument

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

Parameter adalah *syarat* input yang harus dimasukkan ke dalam suatu fungsi dan dideklarasikan bersama dengan deklarasi fungsi. Sementara argument adalah *nilai* yang dimasukan ke dalam suatu fungsi, sesuai dengan persyaratan parameter, di mana argument dituliskan bersamaan dengan pemanggilan fungsi.

Bisa digambarkan seperti ini:

function operasiPerkalian(angka1, angka2){

return angka1 \* angka2;

}

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

Penjelasan kode di atas:

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

## Function Hoisting

Ada hal keren yang dimiliki oleh fungsi JavaScript, yaitu kita bisa memanggilnya terlebih dahulu, padahal deklarasi fungsi ada di baris setelahnya.

*"Apa kerennya? Perasaan biasa saja"* mungkin kalian akan berpikir seperti itu.

Contoh:

console.log(operasiPerkalian(5, 5)); // Output: 25

function operasiPerkalian(angka1, angka2) {

return angka1 \* angka2;

}

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

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

Contoh:

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

const operasiPerkalian = function(angka1, angka2) {

return angka1 \* angka2;

};

## Lingkup Global vs Lingkup Lokal

Sekarang kita akan belajar tentang lingkup dari sebuah variabel. Di JavaScript, ada istilah lingkup global (global scope) dan lingkup lokal (local scope), apakah perbedaannya? Mari kita pelajari bersama di bawah ini.

### Lingkup Global

Lingkup global adalah ketika sebuah variabel bisa diakses dari mana saja, baik di dalam maupun di luar dari suatu fungsi atau blok (grup) kode.

Contoh:

const olahraga = 'basketball';

function namaAtlet() {

let atlet = 'Lionel Messi';

const noMessi = 10;

console.log(olahraga); // Output: basketball

if (olahraga === 'basketball') {

atlet = 'Kobe Bryant';

const noKobe = 24;

console.log(olahraga); // Output: basketball

}

return atlet;

}

console.log(namaAtlet()); // Output: Kobe Bryant

console.log(olahraga); // Output: basketball

Pada contoh di atas, fungsi variabel olahraga bisa diakses di dalam fungsi namaAtlet dan bahkan di dalam blok if walaupun variabel olahraga tidak dideklarasikan di dalam fungsi/blok tersebut. Ini dikarenakan variabel olahraga yang dideklarasikan di luar fungsi namaAtlet memiliki lingkup global, jadi bisa diakses dari mana saja selama ada di dalam lingkup variabel tersebut.

### Lingkup Lokal

Sebaliknya, lingkup lokal adalah ketika sebuah variabel hanya bisa diakses di dalam sebuah fungsi atau blok kode. Semua variabel yang dideklarasikan di dalam sebuah fungsi/blok hanya bisa di dalam fungsi/blok tersebut saja.

Jadi, lingkup lokal bisa dibagi lagi menjadi dua tipe:

* lingkup fungsi (function scope)
* lingkup blok (block scope)

Mari kita gunakan contoh sebelumnya dan ubah perintah console.log menjadi:

const olahraga = 'basketball';

function namaAtlet() {

let atlet = 'Lionel Messi'; // lingkup fungsi

const noMessi = 10; // lingkup fungsi

if (olahraga === 'basketball') {

atlet = 'Kobe Bryant';

const noKobe = 24; // lingkup blok

console.log(noMessi); // Output: 10

}

console.log(noKobe); // Output: Uncaught ReferenceError: noKobe is not defined

return atlet;

}

namaAtlet();

Pada contoh di atas, perintah console.log(noKobe) akan mengembalikan nilai error karena kita mencoba mengakses variabel noKobe di luar lingkup di mana dia dideklarasikan (yaitu di dalam blok if saja). Sedangkan perintah console.log(noMessi) akan menghasilkan output 10 karena variabel noMessi dideklarasikan di luar blok if jadi semua kode di dalam blok if tersebut mempunyai akses ke variabel itu.

# JavaScript Array

Masih ingat tentang konsep variabel JavaScript? Kita bahas sedikit ya untuk mengingat kembali.

Variabel JavaScript bisa kita gambarkan sebagai wadah yang bisa menampung barang dengan berbagai macam tipe, tak peduli bentuknya, material pembuatnya, warnanya dan lain-lain. Sama seperti wadah, variabel juga bisa menampung berbagai macam tipe data. Namun, variabel yang kita pelajari kemarin hanya bisa menampung satu data saja, betul?

Bagaimana kalau kita memiliki lebih dari satu data dan ingin ditampung pada wadah, atau di kasus ini, variabel yang sama? Tenang, JavaScript memiliki solusinya. Kita bisa menggunakan array.

Array adalah tipe variabel yang dapat menampung berbagai jenis data dengan tipe yang bermacam-macam, dengan jumlah yang tidak terbatas. Array di JavaScript memiliki ciri khas yaitu data yang ditampung dibungkus dengan sepasang kurung siku [ ].

let namaArray = [ nilai1, nilai2, nilai3, ...];

Di bawah ini adalah perbandingan jika kita menyimpan *data di variabel satu-per-satu* dengan menyimpan *banyak data di satu variabel* array:

// Menyimpan data di variabel satu-per-satu

let murid1 = 'Andi';

let murid2 = 'Ratna';

let murid3 = 'Juwita';

// Menyimpan lebih dari satu data dalam satu array

let murid = ['Andi', 'Ratna', 'Juwita'];

Pada contoh di atas, data yang kita miliki hanya berjumlah 3 buah. Bayangkan jika kita memiliki 1000 data! Pasti capek jika harus membuat 1000 variabel dalam satu waktu 😵.

Lebih ringkas bukan menggunakan satu array untuk menampung banyak data dibandingkan harus membuat variabel untuk masing-masing data? Iya dong jelas 👍

Yuk mari pelajari lebih lanjut tentang array!

## Mendeklarasikan Array

Di topik sebelumnya, sebenarnya kalian sudah melihat contoh syntax mendeklarasikan variabel array.

Yap betul, untuk mendeklarasi sebuah array, caranya hampir sama dengan mendeklarasi objek. Perbedaannya adalah data array dibungkus dengan kurung siku [] atau menggunakan kata kunci new Array().

### Array Literal

Cara mendeklarasikan array dengan *array literal* adalah sebagai berikut:

let namaArray = [ element1, element2, element3 ];

Contoh:

let buah = [ 'Mangga', 'Apel', 'Jeruk'];

let hargaBuah = ["20000", 30000, 15000]; // Tipe data di array boleh berbeda

### Kata Kunci New

Cara mendeklarasikan array JavaScript dengan kata kunci new adalah sebagai berikut:

let namaArray = new Array(element1, element2, element3);

Contoh:

let buah = new Array( 'Mangga', 'Apel', 'Jeruk');

let hargaBuah = new Array("20000", 30000, 15000); // Tipe data di array boleh berbeda

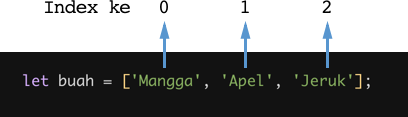
## Array Index

Kita sudah tau bahwa array bisa menampung lebih dari satu data, bahkan tidak terhingga. Nah, sekarang muncul pertanyaan, bagaimana cara mengakses data-data tersebut?

Sebelum menjawab pertanyaan tersebut, ada baiknya kita telusuri dulu anatomi dari array.

Setiap data di array memiliki nomor index. Nomor index berguna untuk mengakses data suatu array di posisi tertentu. Nomor index di array selalu dimulai dari angka nol 0.

Gambar di bawah ini adalah ilustrasi nomor index di array:



Kemudian, setiap array pasti memiliki jumlah data yang ditampungnya, atau disebut dengan Array Length.

Contoh:

let buah = ['Mangga', 'Apel', 'Jeruk'];

let olahraga = ['basket', 'sepakbola', 'badminton', 'berenang'];

let murid = [];

console.log(buah.length); // Output: 3

console.log(olahraga.length); // Output: 4

console.log(murid.length); // Output: 0

Jadi, mohon diingat sekali lagi 🙏🏻:

* Nomor index dari suatu array selalu dimulai dari angka nol 0.

## Mengakses Data/Element di Dalam Array

Di topik ini kita akan mempelajari bagaimana caranya untuk dapat mengakses salah satu data/element di dalam sebuah array.

Disinilah kita akan menggunakan nomor index dan array length yang telah kita pelajari dari topik sebelum ini.

### Mengakses Element Tunggal

Syntax yang digunakan untuk mengakses data di dalam array seperti ini:

namaArray[nomorIndex]

Contohnya seperti ini:

let namaBuah = ["mangga", "apel", "jeruk", "durian"];

console.log(namaBuah[0]); // Output: mangga

console.log(namaBuah[1]); // Output: apel

console.log(namaBuah[2]); // Output: jeruk

console.log(namaBuah[3]); // Output: durian

### Mengakses Element Terakhir di Array

Coba kalian perhatikan contoh sebelumnya. durian adalah element terakhir dari array namaBuah. Untuk mengakses element itu, kita bisa hitung berdasarkan nomor index-nya dan kebetulan array namaBuah hanya memiliki 4 element jadi hitungnya masih gampang.

Coba kalian perhatikan contoh berikut:

let countries = ["Afghanistan","Albania","Algeria","Andorra","Angola","Anguilla","Antigua &amp;Barbuda","Argentina","Armenia","Aruba","Australia","Austria","Azerbaijan","Bahamas","Bahrain","Bangladesh","Barbados","Belarus","Belgium","Belize","Benin","Bermuda","Bhutan","Bolivia","Bosnia &amp; Herzegovina","Botswana","Brazil","British Virgin Islands","Brunei","Bulgaria","Burkina Faso","Burundi","Cambodia","Cameroon","Cape Verde","Cayman Islands","Chad","Chile","China","Colombia","Congo","Cook Islands","Costa Rica","Cote D Ivoire","Croatia","Cruise Ship","Cuba","Cyprus","Czech Republic","Denmark","Djibouti","Dominica","Dominican Republic","Ecuador","Egypt","El Salvador","Equatorial Guinea","Estonia","Ethiopia","Falkland Islands","Faroe Islands","Fiji","Finland","France","French Polynesia","French West Indies","Gabon","Gambia","Georgia","Germany","Ghana","Gibraltar","Greece","Greenland","Grenada","Guam","Guatemala","Guernsey","Guinea","Guinea Bissau","Guyana","Haiti","Honduras","Hong Kong","Hungary","Iceland","India","Indonesia","Iran","Iraq","Ireland","Isle of Man","Israel","Italy","Jamaica","Japan","Jersey","Jordan","Kazakhstan","Kenya","Kuwait","Kyrgyz Republic","Laos","Latvia","Lebanon","Lesotho","Liberia","Libya","Liechtenstein","Lithuania","Luxembourg","Macau","Macedonia","Madagascar","Malawi","Malaysia","Maldives","Mali","Malta","Mauritania","Mauritius","Mexico","Moldova","Monaco","Mongolia","Montenegro","Montserrat","Morocco","Mozambique","Namibia","Nepal","Netherlands","Netherlands Antilles","New Caledonia","New Zealand","Nicaragua","Niger","Nigeria","Norway","Oman","Pakistan","Palestine","Panama","Papua New Guinea","Paraguay","Peru","Philippines","Poland","Portugal","Puerto Rico","Qatar","Reunion","Romania","Russia","Rwanda","Saint Pierre &amp; Miquelon","Samoa","San Marino","Satellite","Saudi Arabia","Senegal","Serbia","Seychelles","Sierra Leone","Singapore","Slovakia","Slovenia","South Africa","South Korea","Spain","Sri Lanka","St Kitts &amp; Nevis","St Lucia","St Vincent","St. Lucia","Sudan","Suriname","Swaziland","Sweden","Switzerland","Syria","Taiwan","Tajikistan","Tanzania","Thailand","Timor L'Este","Togo","Tonga","Trinidad &amp; Tobago","Tunisia","Turkey","Turkmenistan","Turks &amp; Caicos","Uganda","Ukraine","United Arab Emirates","United Kingdom","Uruguay","Uzbekistan","Venezuela","Vietnam","Virgin Islands (US)","Yemen","Zambia","Zimbabwe"];

Tentunya akan lebih sulit untuk mengakses element terakhir dari array countries apabila kita harus menghitung nomor indexnya satu-per-satu dimulai dari element pertama.

Akan tetapi ada cara singkat untuk melakukan semuanya itu!

Di topik terakhir, kita sudah mempelajari tentang array length di mana syntax array.length itu mengembalikan jumlah data dari sebuah array. Di topik yang sama, kita juga diingatkan kalau nomor index dari sebuah array dimulai dari angka nol. Jadi untuk mengakses element terakhir dari sebuah array, kita bisa menggunakan formula sebagai berikut:

nomor index element terakhir array = jumlah data array - 1

Jadi untuk mengakses element terakhir dari array countries di atas bisa menggunakan cara singkat berikut:

// menggunakan formula di atas

let indexElementTerakhir = countries.length - 1;

console.log(countries[indexElementTerakhir]); // Output: Zimbabwe

// atau

console.log(countries[countries.length - 1]); // Output: Zimbabwe

### Mengakses Seluruh Element Array

Namun, jika kita ingin mengakses seluruh data yang berada di dalam suatu array, maka kita cukup *panggil* nama dari array tersebut.

Contoh:

let olahraga = ["Berenang", "Sepak Bola", "Bola Basket"];

console.log(olahraga); // Output: ["Berenang", "Sepak Bola", "Bola Basket"]

### Mengubah Data/Element pada Array

Data/element dari suatu array bisa kita ubah dengan syntax seperti ini:

namaArray[nomorIndex] = nilaiBaru;

Contohnya:

let namaBuah = ["Mangga", "Apel", "Jeruk"];

namaBuah[1] = "Semangka";

console.log(namaBuah); // Output: ["Mangga", "Semangka", "Jeruk"]

Apakah kalian masih ingat di beberapa topik sebelumnya, kita sempat membahas tentang penggunaan let dan const di mana variabel yang dideklarasikan dengan menggunakan const tidak bisa diubah datanya terkecuali variabel itu adalah tipe objek atau array.

Jadi untuk array JavaScript, pengubahan nilai element seperti berikut tidak akan memunculkan error walaupun variabel array-nya dideklarasikan dengan const:

const namaArray = [nilaiElement1, nilaiElement2];

// mengubah data di element pertama

namaArray[0] = nilaiElementBaru;

// menambah element baru ke array

namaArray[2] = nilaiElement3;

## Array Method

Di JavaScript, hampir semua tipe data yang bukan primitive (primitive contohnya string, number, boolean, null, undefined) adalah objek. Jadi karena array bukan tipe data primitive, array termasuk objek dan bisa mempunyai properti dengan nilai fungsi yang kita kenal dengan method.

Di topik ini, kita akan bahas beberapa array method yang umum digunakan.

### Mengubah Element Array Menjadi String dengan toString()

Method toString() akan menggabungkan semua element array dan mengubahnya ke tipe data string dengan penghubung tanda koma , di antara element array.

Contoh:

let array = [1, 2, 3, "halo", false, true];

console.log(array.toString()); // Output: 1,2,3,halo,false,true

Bisa dilihat dari contoh di atas, di dalam array terdapat data dengan tipe number, string, bahkan boolean, namun hasil dari method toString() mengubah mereka semua menjadi string.

### Menggabungkan Element Array dengan join()

Method join() menggabungkan semua element array, kemudian mengubahnya menjadi string. Konsepnya hampir sama dengan method toString(), namun perbedaannya kita bisa menambahkan karakter tertentu sebagai penghubung antara element array.

array.join(penghubung);

Penjelasan syntax di atas:

* Parameter penghubung adalah optional parameter, yaitu parameter yang boleh diisi, boleh juga dikosongkan. penghubung menentukan karakter apa yang mau ditambahkan sebagai penghubung antara element array. Secara default, apabila dikosongkan, array.join() akan menggunakan tanda koma , sebagai penghubung element array.

Contoh:

let array = [1, 2, 3, "halo", false, true];

console.log(array.join()); // Output: 1,2,3,halo,false,true

console.log(array.join(" ")); // Output: 1 2 3 halo false true

console.log(array.join("#")); // Output: 1#2#3#halo#false#true

console.log(array.join("...")); // Output: 1...2...3...halo...false...true

### Menambah dan Mengeluarkan (Menghapus) Element Array dengan pop(), push(), shift(), dan unshift()

pop(), mengeluarkan element terakhir dari sebuah array.  
Contoh:  
let array = [1, 2, 3, "halo", false, true];

array.pop();

1. console.log(array); // Output: [1, 2, 3, "halo", false]

push(), menambah element di bagian akhir dari sebuah array.  
Contoh:  
let array = [1, 2, 3, "halo", false, true];

array.push("Selamat Pagi");

1. console.log(array); // Output: [1, 2, 3, "halo", false, true, "Selamat Pagi"]

shift(), mengeluarkan element pertama dari sebuah array.  
Contoh:  
let array = [1, 2, 3, "halo", false, true];

array.shift();

1. console.log(array); // Output: [2, 3, "halo", false, true]

unshift(), menambah element di bagian awal array.  
Contoh:  
let array = [1, 2, 3, "halo", false, true];

array.unshift("Selamat Pagi");

1. console.log(array); // Output: ["Selamat Pagi", 1, 2, 3, "halo", false, true]

### Menambah, Menghapus, dan Mengganti Element Array dengan splice()

Digunakan untuk menambah, menghapus, dan mengganti element di sebuah array.

array.splice(index, jumlah, item1, ..., itemX);

Penjelasan syntax di atas:

* Parameter index adalah required parameter, yaitu parameter yang wajib diisi ketika memanggil array method splice. index menjelaskan posisi di mana element harus ditambah atau dihapus.
* Parameter jumlah adalah optional parameter, yaitu parameter yang boleh diisi, boleh juga dikosongkan. jumlah menentukan jumlah element yang mau dihapus di array.
* Parameter item1, ..., itemX adalah optional parameter. item1, ..., itemX merupakan element baru yang mau ditambah ke dalam array.

Contoh:

let buah = ["Pisang", "Jeruk", "Apel", "Mangga"];

buah.splice(2, 0, "Lemon", "Kiwi");

console.log(buah); // Output: ["Pisang", "Jeruk", "Lemon", "Kiwi", "Apel", "Mangga"]

Penjelasan contoh di atas:

* Parameter pertama bernilai 2, mendefinisikan posisi di mana element baru harus ditambahkan, yaitu di index ke 2 atau element ke 3 (ingat, index di array dimulai dari angka 0).
* Parameter kedua bernilai 0, mendefinisikan jumlah element yang harus dihapus, yaitu berjumlah 0 (tidak ada yang dihapus).
* Parameter ketiga, keempat, dan seterusnya di method splice() mendefinisikan element baru yang ingin ditambahkan ke dalam array. Jika parameter ketiga dan seterusnya tidak dituliskan, maka tidak ada element yang ditambahkan. Pada contoh di atas, element ketiga dan keempat adalah "Lemon" dan "Kiwi", maka kedua element tersebut ditambahkan pada array buah.

### Menggabungkan Lebih Dari Satu Array dengan concat()

Method concat() digunakan untuk menggabungkan 2 array atau lebih.

array1.concat(array2, array3, ..., arrayX)

Contoh:

let buah = ["pisang", "apel", "jeruk"];

let sayur = ["tomat", "bayam", "wortel"];

let biji = ["kedelai", "kacang tanah", "kacang polong"];

let makanan = buah.concat(sayur, biji);

console.log(makanan); // Output: ["pisang", "apel", "jeruk", "tomat", "bayam", "wortel", "kedelai", "kacang tanah", "kacang polong"]

### Mengambil Element Array dengan slice()

Method slice() digunakan untuk mengambil beberapa element pada array, dan menaruhnya pada array baru.

array.slice(mulai, akhir)

Penjelasan syntax di atas:

* Parameter mulai adalah optional parameter. mulai menentukan permulaan index dari element yang akan diambil.
* Parameter akhir adalah optional parameter. akhir menentukan batas akhir index element yang akan diambil (tanpa diikutsertakan). Apabila dikosongkan, maka semua element dimulai dari index mulai sampai ke element terakhir dari array akan diambil.

Contoh:

let cemilan = ["kripik singkong", "kripik kentang", "krupuk ikan", "permen", "coklat", "kue"];

let cemilanGurih = cemilan.slice(0, 3);

let cemilanManis = cemilan.slice(3);

console.log(cemilanGurih); // Output: ["kripik singkong", "kripik kentang", "krupuk ikan"]

console.log(cemilanManis); // Output: ["permen", "coklat", "kue"]

# JavaScript Object

Di topik sebelumnya, kita sudah belajar variabel yang menampung satu data saja. Namun di JavaScript sebenarnya variabel itu tidak dibatasi hanya untuk menampung satu data saja. Ada yang namanya variabel tipe objek atau array yang bisa menampung banyak data dari berbagai macam tipe data di JavaScript.

Di JavaScript, objek bisa digambarkan sebagai sesuatu yang memiliki properti dan nilai.

Sebagai contoh, manusia. Manusia memiliki nama, pekerjaan, umur, dan lain sebagainya. Nah, kemudian kita tahu bahwa nama orang ini adalah John, pekerjaannya adalah programmer, umurnya 24 tahun, dan lain-lain.

Dalam hal ini, *manusia* adalah objek. Kemudian *nama*, *pekerjaan*, *umur* adalah properti objek. Terakhir, *John* adalah nilai dari properti nama, *programmer* adalah nilai dari properti pekerjaan, dan *24 tahun* adalah nilai dari properti umur.

Yap, sesimpel itu konsep dari sebuah objek JavaScript. Oh, jangan lupa! Objek JavaScript dapat menampung properti dan nilai *sebanyak apapun* dengan *tipe data yang beragam*.

Yuk kita pelajari lebih lanjut tentang objek JavaScript!

## Mendeklarasikan Objek

### Menggunakan Object Literal

Cara mendeklarasikan objek dengan *object literal* adalah sebagai berikut:

let namaObjek = {

namaProperti1: nilai1,

namaProperti2: nilai2

};

Contoh:

let buah = { nama: 'mangga', warnaKulit: 'hijau', hargaPerBuah: 5000 };

let orang = {

nama: 'john',

umur: 24,

pekerjaan: 'programmer'

};

Perhatikan contoh kode di atas! Objek berciri khas datanya dibungkus dengan kurung kurawal {}, dan antar data diberi koma ,.

### Menggunakan Kata Kunci new

Cara mendeklarasikan objek JavaScript dengan kata kunci new adalah sebagai berikut:

let namaObjek = new Object();

namaObjek.namaProperti1 = nilai1;

namaObjek.namaProperti2 = nilai2;

Contoh:

let orang = new Object();

orang.nama = 'john';

orang.umur = 24;

orang.pekerjaan = 'programmer';

## Mengakses Properti Objek

Jika kita ingin menggunakan nilai yang terdapat di dalam properti suatu objek, maka kita harus mengakses properti objek tersebut.

Caranya bagaimana?

Ada 2 cara yaitu:

### Dot Notation

let objek = {

namaProperti: nilaiProperti

};

// Dot Notation

objek.namaProperti // Output: nilaiProperti

Contoh:

let orang = {

nama: 'john',

umur: 24,

pekerjaan: 'programmer'

};

console.log(orang.nama); // Output: "john"

console.log(orang.umur); // Output: 24

console.log(orang.pekerjaan); // Output: "programmer"

### Bracket Notation

let objek = {

namaProperti: nilaiProperti

};

// Bracket Notation

objek["namaProperti"] // Output: nilaiProperti

// bisa juga menggunakan single quote

objek['namaProperti'] // Output: nilaiProperti

Contoh:

let orang = {

nama: 'john',

umur: 24,

pekerjaan: 'programmer'

};

console.log(orang["nama"]); // Output: "john"

console.log(orang['umur']); // Output: 24

console.log(orang["pekerjaan"]); // Output: "programmer"

#### 📝Catatan:

#### Perlu kita perhatikan yang berikut ini ketika kita mau mengakses properti objek secara dynamic (dinamis). Yang dimaksud dengan mengakses properti objek secara dinamis adalah ketika kita menggunakan nama dari suatu variabel sebagai nama properti saat mengakses properti objek tersebut.

let namaVariabel = namaProperti;

let objek = {

namaProperti: nilaiProperti

};

// kita bisa mengakses properti objek secara dinamis

// dengan cara berikut

objek[namaVariabel] // Output: nilaiProperti

Contoh:

const smartphone = {

kamera: "10MP",

memori: "128GB"

};

const fitur = "kamera";

Apabila kita ingin mengakses nilai "10MP" dari objek smartphone dengan menggunakan variabel fitur. Cara yang benar adalah:

console.log(smartphone[fitur]); // Output: 10MP

// karena variabel fitur mempunya nilai kamera

// maka, baris di atas sama hasilnya dengan

console.log(smartphone["kamera"]); // Output: 10MP

console.log(smartphone.kamera); // Output: 10MP

Kita tidak bisa mengakses properti objek secara dinamis dengan menggunakan dot notation. Jadi:

console.log(smartphone.fitur); // Output: undefined

Contoh di atas akan menghasilkan output undefined dikarenakan objek smartphone tidak mempunyai properti dengan nama fitur.

## Menambah Properti Baru pada Objek

Cara menambah properti baru ke dalam suatu objek mirip dengan cara mendeklarasikannya.

### Dot Notation

let objek = { namaProperti1: nilaiProperti1 };

objek.namaProperti2 = nilaiProperti2;

console.log(objek); // output: { namaProperti1: nilaiProperti1, namaProperti2: nilaiProperti2 }

Contoh:

let orang = {

nama: 'john',

umur: 24,

pekerjaan: 'programmer'

};

orang.kebangsaan = 'warga negara indonesia';

console.log(orang);

/\*

Output:

{

nama: 'john',

umur: 24,

pekerjaan: 'programmer',

kebangsaan: 'warga negara indonesia'

}

\*/

### Bracket Notation

let objek = { namaProperti1: nilaiProperti1 };

objek['namaProperti2'] = nilaiProperti2;

console.log(objek); // output: { namaProperti1: nilaiProperti1, namaProperti2: nilaiProperti2 }

Contoh:

let orang = {

nama: 'john',

umur: 24,

pekerjaan: 'programmer'

};

orang['kebangsaan'] = 'warga negara indonesia';

console.log(orang);

/\*

Output:

{

nama: 'john',

umur: 24,

pekerjaan: 'programmer',

kebangsaan: 'warga negara indonesia'

}

\*/

Apakah kalian masih ingat di beberapa topik sebelumnya, kita sempat membahas tentang penggunaan let dan const di mana variabel yang dideklarasikan dengan menggunakan const tidak bisa diubah datanya terkecuali variabel itu adalah tipe objek atau array.

Jadi untuk objek JavaScript, pengubahan nilai properti seperti berikut tidak akan memunculkan error walaupun variabel objek-nya dideklarasikan dengan const:

const objek = { namaProperti1: nilaiProperti1 };

// mengubah data properti

objek.namaProperti1 = nilaiProperti2; // OK

// menambah properti baru ke objek

objek.namaProperti3 = nilaiProperti3; // OK

#### 📝Catatan:

#### Variabel tipe objek dan array yang dideklarasi dengan menggunakan const bisa diubah atau ditambah nilai properti/element-nya. Tetapi variabel-nya sendiri tetap tidak bisa diubah nilainya.

## Object Method

Seperti yang kita bahas di topik sebelumnya, objek itu memiliki properti dan nilai. Selama ini kalian baru belajar untuk menambahkan nilai yang berupa string atau number, misalnya seperti ini

const user = {

nama: "Stefan",

umur: 21

}

Sebenarnya tipe data dari nilai sebuah properti itu tidak terbatas kepada string dan number saja. Kita juga bisa memberi nilai berupa sebuah fungsi.

*"Ha? Berarti ada fungsi di dalam objek gitu?"*

Benar banget! Fungsi yang terhubung pada sebuah objek kita sebut dengan istilah method.

Jadi, objek sebenarnya bisa menampung fungsi di dalam propertinya, seperti ini contohnya:

let objek = {

namaProperti1: nilaiProperti1,

namaProperti2: function() {...} // nilai dari namaProperti2 adalah method dari variabel objek

};

Contoh:

let kalkulator = {

namaOperasi: 'penjumlahan',

jumlah: function(angka1, angka2) {

return angka1 + angka2;

}

};

*"Lalu bagaimana cara mengakses method yang berada di dalam objek? Apakah sama dengan cara mengakses properti objek biasa?"*

Hampir mirip, hanya tinggal tambahkan tanda kurung () dan isi parameter di dalam tanda kurung tersebut jika method tersebut membutuhkan parameter.

Contoh:

let kalkulator = {

namaOperasi: 'penjumlahan',

jumlah: function(angka1, angka2) {

return angka1 + angka2;

}

};

console.log(kalkulator.jumlah(2, 3)); // Output: 5

#### 📝Catatan:

#### Karena method merupakan properti dari suatu objek, maka untuk menambahkan method baru ke dalam objek caranya sama seperti menambahkan properti baru ke dalam objek.

Kalian sekarang mungkin berpikir "*Kenapa tidak buat fungsi sendiri saja di luar objeknya?*"

Menambahkan fungsi sebagai method dari sebuah objek itu memilki beberapa kelebihan:

#### **Menghindari konflik penamaan fungsi**

Misalkan kita mempunyai dua objek, kucing dan anjing. Lalu kita ingin membuat dua buah fungsi suara yang mengembalikan suara dari kedua hewan tersebut. Tentu kita tidak bisa melakukannya seperti di bawah ini. Sebab saat kita mendefinisikan fungsi yang namanya sama dengan fungsi sebelumnya, fungsi baru itu akan menimpa fungsi yang lama.

const kucing = {

// isi properti dan nilai objek kucing

};

function suara() { return "Meong"; }

const anjing = {

// isi properti dan nilai objek anjing

};

//fungsi ini akan menimpa fungsi suara yang pertama

function suara() { return "Gukguk"; }

console.log(suara()); // Output: Gukguk

Dengan membuat fungsi suara sebagai method dari masing-masing objek, kita tetap dapat menghindari konflik tersebut. Sebab fungsi suara sekarang akan terhubung dengan objek yang berbeda. Jadi pada saat pemanggilannya, kita juga mengawalinya dengan nama objek di mana method tersebut didefinisikan.

const kucing = {

// isi properti dan nilai objek kucing,

suara: function() { return "Meong"; }

};

const anjing = {

// isi properti dan nilai objek anjing

suara: function() { return "Gukguk"; }

};

//panggil method suara dalam masing-masing objek

console.log(kucing.suara()); //Output: Meong

console.log(anjing.suara()); // Output: Gukguk

#### **Definisi fungsi yang terpusat**

Kita ambil contoh objek di awal tadi

const user = {

nama: "Stefan",

umur: 21

};

Katakanlah kita ingin menggunakan kalimat perkenalan "Nama saya Stefan. Saya berumur 21 tahun"di beberapa fungsi lainnya. Cara manual tentu dengan menulisnya satu per satu seperti ini

function fungsiA() {

//isi dari fungsiA

console.log(`Nama saya ${user.nama}. Saya berumur ${user.umur} tahun`);

}

function fungsiB() {

//isi dari fungsiB

console.log(`Nama saya ${user.nama}. Saya berumur ${user.umur} tahun`);

}

function fungsiC() {

//isi dari fungsiC

console.log(`Nama saya ${user.nama}. Saya berumur ${user.umur} tahun`);

}

Tapi misalkan sekarang kita ingin menambahkan properti baru makananFavorit: "Pizza" ke objek user. Tentu kita juga ingin mengubah kalimat perkenalannya menjadi "Nama saya Stefan. Saya berumur 21 tahun. Saya suka Pizza". Tapi karena kita membuat kalimatnya secara manual pada fungsiA, fungsiB, dan fungsiC, kita harus mengubah secara manual juga di ketiga fungsi tersebut.

"*Repot banget dong kalau begitu*"

Nah, dengan membuat sebuah method di objek user yang mengembalikan kalimat perkenalan tadi, kita cukup memanggil method tersebut di tempat kita ingin menggunakan kalimat tersebut saja.

// buat method perkenalanDiri yang mengembalikan kalimat perkenalan

const user = {

nama: "Stefan",

umur: 21,

makananFavorit: "Pizza",

perkenalanDiri: () => {

return `Nama saya ${this.nama}. Saya berumur ${this.umur}. Saya suka ${this.makananFavorit}`;

}

};

// panggil method tersebut di dalam fungsi yang kita inginkan

function fungsiA() {

// isi dari fungsiA

console.log(user.perkenalanDiri());

}

function fungsiB() {

// isi dari fungsiB

console.log(user.perkenalanDiri());

}

function fungsiC() {

// isi dari fungsiC

console.log(user.perkenalanDiri());

}

Sekarang, jika ada perubahan pada kalimat perkenalan, kita hanya perlu mengubahnya di satu tempat saja: di method perkenalanDiri() pada objek user.

# JavaScript Conditional

Kalau diterjemahkan ke dalam bahasa Indonesia, conditional artinya adalah persyaratan. Lalu, kalau dalam konsep JavaScript, conditional itu kira-kira seperti apa?

Ketika kita menulis sebuah aplikasi, seringkali akan ada beberapa operasi yang akan dijalankan dengan persyaratan tertentu. Persyaratan itu sendiri nanti kita yang tentukan.

Sebagai contoh dalam kehidupan sehari-hari, jika kita lapar, maka kita makan, dan jika kita tidak lapar, maka kita tidak makan. Dari contoh tersebut, persyaratan/kondisinya adalah *jika kita lapar* atau *jika kita tidak lapar*. Sementara, keputusan yang dilaksanakan adalah *makan* atau *tidak makan*.

Dari contoh tersebut, ada kata kunci yang bisa kita ambil, yaitu jika dan maka. Jika menyatakan sebuah kondisi, dan maka menyatakan keputusan yang akan dilaksanakan.

Sama seperti di JavaScript, kita juga bisa menyatakan sebuah kondisi dan menyatakan keputusan (program) yang akan dijalankan.

Di JavaScript ada dua cara menulis perintah conditional, yaitu:

* Menggunakan if, else if dan else.
* Menggunakan switch dan case.

Mari kita bahas satu-persatu.

## if, else if dan else

Syntax dalam menggunakan if, else if dan else adalah sebagai berikut.

### if

Digunakan apabila hanya ada 1 kondisi dan 1 keputusan yang dijalankan.

if (kondisi1) {

// masukkan kode yang akan dijalankan di sini apabila kondisi1 tercapai

}

### if...else

Digunakan apabila ada 1 kondisi dan 2 keputusan yang dijalankan.

if (kondisi1) {

// masukkan kode yang akan dijalankan di sini apabila kondisi1 tercapai

} else {

// masukkan kode yang akan dijalankan di sini apabila kondisi1 TIDAK tercapai

}

### if...else if...else

Digunakan apabila ada beberapa kondisi dan beberapa keputusan yang dijalankan.

if (kondisi1) {

// masukkan kode yang akan dijalankan di sini apabila kondisi1 tercapai

} else if (kondisi2) {

// masukkan kode yang akan dijalankan di sini apabila kondisi1 tidak tercapai dan kondisi2 tercapai

} else if (kondisi3) {

// masukkan kode yang akan dijalankan di sini apabila kondisi1 dan kondisi2 tidak tercapai, dan kondisi3 tercapai

} ... {

} else {

// masukkan kode yang akan dijalankan di sini apabila semua kondisi di atas TIDAK tercapai

}

Contoh Penggunaan:

Coba bayangkan sebuah aplikasi yang bisa menampilkan kalimat:

* "SANGAT MEMUASKAN" jika nilai di atas 80
* "MEMUASKAN" jika nilai berada di antara 60 hingga 80, dan
* "JANGAN MENYERAH, COBA LAGI!" jika nilai di bawah 60

Ternyata, Andi mendapatkan nilai 95. Maka, Andi seharusnya mendapatkan kalimat "SANGAT MEMUASKAN" bukan?

Bagaimana cara membuat aplikasinya? Yaitu seperti berikut ini:

let nilaiAndi = 95;

if (nilaiAndi > 80) {

console.log("SANGAT MEMUASKAN");

} else if (nilaiAndi >= 60 && nilaiAndi <= 80) {

console.log("MEMUASKAN");

} else {

console.log("JANGAN MENYERAH, COBA LAGI!");

}

// Output: SANGAT MEMUASKAN

Penjelasan kode di atas:

* if (nilaiAndi > 80) { console.log('SANGAT MEMUASKAN'); }artinya jika variabel nilaiAndi bernilai di atas 80, maka aplikasi akan menampilkan kalimat "SANGAT MEMUASKAN".
* else if (nilaiAndi >= 60 && nilaiAndi <= 80) { console.log('MEMUASKAN'); }, artinya jika variabel nilaiAndi berada di atas atau sama dengan 60 dan nilaiAndi berada di bawah atau sama dengan 80 (yang bisa diartikan sebagai jika nilaiAndi berada di antara 60 hingga 80), maka aplikasi akan menampilkan kalimat "MEMUASKAN".
* else { console.log('JANGAN MENYERAH, COBA LAGI!');}, artinya selain kedua kondisi yang telah disebutkan di if dan else if, maka aplikasi akan menampilkan kalimat "JANGAN MENYERAH, COBA LAGI!".

## switch dan case

Jika kita memiliki banyak pilihan kondisi, menggunakan perintah if, else if dan else akan kurang efisien karena kode yang ditulis akan menjadi sangat panjang sekali.

*"Alternatifnya apa dong?"*

Ada cara yang lebih efisien yaitu dengan menggunakan perintah switch dan case.

Syntaxnya seperti ini:

switch (pernyataan) {

case kondisi1:

// keputusan yang dijalankan ketika kondisi1 tercapai

break;

case kondisi2:

// keputusan yang dijalankan ketika kondisi2 tercapai

break;

case kondisi3:

// keputusan yang dijalankan ketika kondisi3 tercapai

break;

...

default:

// keputusan yang dijalankan ketika semua kondisi tidak tercapai

}

Contoh Penggunaan:

Seandainya kita disuruh menulis sebuah aplikasi yang akan menampilkan jenis seragam apa yang harus dikenakan murid sesuai dengan harinya. Contohnya seperti di bawah ini:

let seragamSekolah;

let hari = "senin";

switch (hari) {

case "senin":

seragamSekolah = "kemeja putih bawahan merah";

break;

case "selasa":

seragamSekolah = "kemeja hijau bawahan hitam";

break;

case "rabu":

seragamSekolah = "kemeja putih bawahan putih";

break;

case "kamis":

seragamSekolah = "kemeja batik merah bawahan hitam";

break;

case "jumat":

seragamSekolah = "baju pramuka";

break;

default:

seragamSekolah = "baju bebas";

}

console.log(seragamSekolah); // Output: kemeja putih bawahan merah

Penjelasan kode di atas:

* switch digunakan untuk mendeklarasikan pernyataan yang menjadi patokan kondisi. Pada contoh di atas, kita akan melihat variabel hari digunakan sebagai patokan kondisi untuk merubah nilai dari variabel seragamSekolah sesuai dengan nilai hari-nya (kondisi).
* case digunakan untuk menulis masing-masing kondisi dari hasil pernyataan yang akan diperiksa. Pada contoh di atas, jika variabel hari bernilai "senin", maka variabel seragamSekolah bernilai "kemeja putih bawahan merah", dan seterusnya.
* break digunakan untuk menghentikan program berjalan ketika sudah menemukan kondisi yang tercapai. Pada contoh di atas, program akan berhenti pada case bernilai "senin".
* default adalah keputusan yang akan dijalankan apabila hasil pernyataan di switch tidak ada yang sesuai dengan masing-masing kondisi di case (tidak ada kondisi yang tercapai). Fungsinya sama seperti else di topik sebelumnya . Pada contoh di atas, apabila nilai variabelhari yang dimasukkan misalnya "sabtu", maka seragamSekolah akan bernilai "baju bebas".

Apabila kita mempunyai dua atau lebih kondisi yang berbeda, tetapi keputusan yang akan dijalankan sama, kita bisa menggabungkan beberapa perintah case tanpa menggunakan break dan hanya menggunakan break di kondisi terakhir.

Misalnya pada contoh kita di atas, murid diharuskan mengenakan seragam "kemeja putih bawahan merah" untuk hari "senin" sampai "kamis", maka kita bisa mengubah kode di atas menjadi:

let seragamSekolah;

let hari = "senin";

switch (hari) {

case "senin":

case "selasa":

case "rabu":

case "kamis":

seragamSekolah = "kemeja putih bawahan merah";

break;

case "jumat":

seragamSekolah = "baju pramuka";

break;

default:

seragamSekolah = "baju bebas";

}

console.log(seragamSekolah); // Output: kemeja putih bawahan merah

# JavaScript Loop

Loop dalam Bahasa Indonesia artinya putaran. Dalam JavaScript, maksudnya adalah perulangan. Jadi, loop adalah sekumpulan kode yang akan dijalankan berulang kali sampai batas yang ditentukan.

Sebagai contoh, coba tampilkan angka 1 sampai dengan 10 di console. Solusi seperti apakah yang kalian pikirkan? Apakah seperti ini?

console.log(1);

console.log(2);

console.log(3);

console.log(4);

console.log(5);

console.log(6);

console.log(7);

console.log(8);

console.log(9);

console.log(10);

/\*

Output:

1

2

3

4

5

6

7

8

9

10

\*/

Coba bayangkan jika kalian disuruh untuk menampilkan angka 1 sampai 1000 pada console dengan cara di atas, mungkin jari kalian sudah pegal-pegal.

Nah untuk menghindari itu, kita bisa menggunakan JavaScript Loop.

Ada 3 jenis loop di JavaScript, yaitu:

* for
* while
* do...while

Kita akan bahas satu-persatu di topik selanjutnya.

## for loop

Syntax yang digunakan ketika menggunakan for loop adalah sebagai berikut:

for (pernyataan1; pernyataan2; pernyataan3) {

// kode yang akan dijalankan ketika pernyataan2 benar (true)

}

Penjelasan kode di atas:

* pernyataan1 digunakan untuk menentukan nilai awal berjalannya loop.
* pernyataan2 digunakan untuk mendefinisikan kondisi berjalannya sebuah loop. Apabila nilai kondisinya false, maka loop akan berakhir.
* pernyataan3 digunakan untuk menambah atau mengurangi nilai awal pada pernyataan1 setiap kali loop dijalankan.

Contoh Penggunaan:

Kita akan menampilkan angka 1 sampai dengan 10 di console. Caranya adalah:

for (let i = 1; i <= 10; i++) {

console.log(i);

}

/\*

Output:

1

2

3

4

5

6

7

8

9

10

\*/

Penjelasan contoh di atas:

* let i = 1 adalah pernyataan1. Kode tersebut berarti kita mendeklarasikan sebuah variabel baru bernama i dengan nilai awal 1. Jadi, pada contoh di atas, loop akan dimulai dari angka 1.
* i <= 10 adalah pernyataan2. Kode tersebut menentukan kondisi apakah loop bisa dijalankan. Loop akan terus berjalan apabila nilai variabel i masih di bawah atau sama dengan 10. Pada contoh di atas, perulangan yang dimaksud adalah menampilkan nilai variabel i pada console.
* i++ adalah pernyataan3. Kode tersebut berarti, setiap melakukan satu perulangan, maka tambahkan nilai i dengan angka 1.

#### 📝Catatan:

#### Masih ingat kan fungsi operator ++? Cek topik Operator Aritmatika.

## while loop

Syntax dalam menggunakan while loop adalah sebagai berikut:

while (kondisi) {

// kode yang akan dijalankan ketika kondisi benar (true)

}

Penjelasan kode di atas:

* while (kondisi) berarti ketika kondisi yang ditentukan benar (true), maka jalankan semua kode yang ada di dalam {}.

Contoh Penggunaan:

Kita akan menampilkan angka 1 sampai dengan 10 di console dengan while loop, caranya adalah:

let i = 1;

while (i <= 10) {

console.log(i);

i++;

}

/\*

Output:

1

2

3

4

5

6

7

8

9

10

\*/

Penjelasan kode di atas:

* let i = 1; berarti kita mendeklarasikan variabel i dengan nilai awal 1.
* while (i <= 10) berarti ketika variabel i bernilai kurang dari atau sama dengan 10, maka jalankan semua kode di dalam {}. Pada contoh di atas, program akan menampilkan nilai variabel i pada console dan kemudian menambahkan nilai variabel i dengan angka 1. Setelah itu, program akan mengevaluasi apakah kondisi i <= 10 masih true. Bila iya, maka kode akan terus dijalankan sampai kondisi false.

# 

## do while Loop

Syntax dalam penggunaan do...while loop adalah:

do {

// kode yang akan dijalankan ketika kondisi benar (true)

} while (kondisi);

Contoh Penggunaan:

Kita akan menampilkan angka 1 sampai dengan 10 di console dengan do...while loop, caranya adalah:

let i = 1;

do {

console.log(i);

i++;

} while (i <= 10);

/\*

Output:

1

2

3

4

5

6

7

8

9

10

\*/

Penjelasan kode di atas:

* let i = 1; berarti kita mendeklarasikan variabel i dengan nilai awal 1.
* do { console.log(i); i++; } menentukan kode yang akan dijalankan ketika kondisi benar (true). Pada contoh di atas, program akan menampilkan nilai variabel i pada console dan kemudian menambahkan nilai variabel i dengan angka 1.
* while (i <= 10) menentukan kondisi dari kode blok do {...}, yaitu apabila variabel i bernilai kurang dari atau sama dengan 10, maka jalankan semua kode di dalam do {...}.

# 

## Array built-in method

Perulangan sangat diperlukan untuk memudahkan dalam mengakses setiap elemen pada *array*. Kamu bisa menggunakan *looping* (for, while, dan do...while) untuk dapat mengakses masing-masing elemen. Contohnya seperti berikut ini:

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

for (let i = 0; i < arr.length; i++) {

console.log(arr[i]);

}

/\* Output

1

2

3

4

5

\*/

Namun *Array* memiliki *built-in method* atau *method* bawaan sendiri yang dapat kita gunakan untuk melakukan perulangan. Berikut kelebihan *built-in method* yang akan dipelajari :

1. Kode lebih mudah dibaca.
2. Mengurangi kesalahan ketika menggunakan banyak tanda operator seperti pada perulangan konvensional.
3. Menjaga variabel yang dimiliki tetap pada blok kodenya.

Pada topik selanjutnya, kamu akan mempelajari beberapa *built-in method* untuk melakukan perulangan pada *array* sehingga memudahkan kamu dalam melakukan perulangan karena masing-masing *method*-nya memiliki fungsi yang berbeda-beda.

#### 📝 Catatan:

#### Beberapa *method* yang akan dipelajari menggunakan argumen berupa *callback function*. Mudahnya, kamu akan memberikan sebuah *function* sebagai argumen pada *method* tersebut.

### Array.forEach()

Dalam mengakses elemen *array*, biasanya kita melakukan perulangan dengan menggunakan for, while dan do-while. Namun selain menggunakan perulangan tersebut, JavaScript juga menyediakan banyak metode untuk mengakses elemen *array*, diantaranya adalah dengan menggunakan .forEach().

.forEach() digunakan untuk melakukan iterasi dalam mengakses elemen *array* dengan 3 parameter *callback* sebagai berikut:

1. *value* yaitu isi dari tiap elemen yang diproses.
2. *index* (*optional*) yaitu nomor *index* tiap elemen.
3. *array* (*optional*) yaitu *array* itu sendiri.

Berbeda dengan melakukan perulangan menggunakan for, while dan do-while. Melakukan perulangan dengan .forEach() mempunyai struktur kode seperti di bawah ini:

array.forEach(function (value, index, array){

// kode program

})

Untuk lebih jelasnya, contoh penggunaan .forEach() dapat dilihat seperti berikut ini:

let colors = ['blue', 'red', 'yellow', 'green'];

colors.forEach(function (data) {

console.log(data)

});

/\* Output:

blue

red

yellow

green

\*/

Dari kode di atas dapat kita lihat bahwa variabel data merepresentasikan elemen dari *array* colors.

Kita juga dapat menggunakan parameter index untuk mendapatkan/mengakses *index* dari setiap elemen array:

let colors = ['blue', 'red', 'yellow', 'green'];

colors.forEach(function (data, idx) {

if(idx % 2 === 0) { // melakukan pengecekan jika index elemen adalah bilangan genap

console.log(data, idx)

} else {

console.log('not allowed', idx)

}

});

/\* Output:

blue 0

not allowed 1

yellow 2

not allowed 3

\*/

Dari contoh ke-2, terdapat variabel idx sebagai *callback* yang mewakili *index* setiap elemen *array*.

Sama halnya dengan menggunakan for, while, dan do-while, kita dapat melakukan berbagai macam *action* saat iterasi dalam elemen *array* yang sedang berjalan. Penggunaan .forEach() dan perulangan lainnya untuk memberikan pilihan bagi programmer dalam kemudahan menulis serta membaca kode.

### Array.map()

Selain .forEach(), ada juga metode perulangan pada *array* yang lain, yaitu .map(). Berbeda dengan .forEach(), .map() digunakan untuk melakukan perulangan sambil mengecek/melakukan operasi terhadap setiap elemen *array*, lalu memberikan nilai balik berupa *array* baru tanpa mengubah nilai pada *array* yang lama.

*Callback function* pada .map() memiliki 3 parameter dasar yang dapat digunakan sesuai kebutuhan:

1. *value* yaitu isi dari tiap elemen yang diproses.
2. *index* (*optional*) yaitu nomor *index* tiap elemen.
3. *array* (*optional*) yaitu *array* itu sendiri.

Struktur kode dari .map():

arr.map(function (value, index, array) {

// kode program

});

Mungkin kamu masih bingung kapan harus menggunakan .forEach() dan kapan harus menggunakan .map(). Mari kita lihat kasus di bawah ini:

// Diberikan data input dan output berikut

// input = [100, 50, 60, 10]

// expected output = [10, 5, 6, 1]

jika kita lihat, *expected output* menghasilkan sebuah array baru. Perbandingan nilai array *input* dan *output* adalah 10. Sehingga dapat kita buat fungsi menggunakan .map() sebagai berikut:

let input = [100, 50, 60, 10];

let output = input.map(function (item) {

return item/10;

});

console.log(output);

// Output [10, 5, 6, 1]

console.log(input);

// Output [100, 50, 60, 10] --> Nilai array pertama tidak berubah

Dari kode di atas, dapat kita lihat bahwa output merupakan array baru hasil modifikasi input menggunakan .map(). Parameter item pada fungsi di atas merepresentasikan *value* (*callback function*), yaitu elemen dari input yang kemudian setiap nilainya dibagi dengan 10 lalu di return sebagai nilai elemen pada *array* output.

Kasus di atas juga dapat dipecahkan menggunakan .forEach() dengan cara sebagai berikut:

let input = [100, 50, 60, 10];

let output = [];

input.forEach(function (item){

output.push(item/10);

});

console.log(output);

// Output [10, 5, 6, 1]

Pada kode di atas, output merupakan *array* baru. Tiap elemen di dalamnya dihasilkan dari elemen input yang dibagi dengan 10, kemudian di push() ke dalam variabel output.

Bedanya penggunaan .map() dengan .forEach() adalah:

1. Menggunakan .map() dalam menghasilkan *array* baru terlihat lebih simpel dan tidak kompleks.
2. Terdapat return yang mengembalikan nilai baru untuk disimpan ke dalam variabel baru.

### Array.filter()

Sebagai *front-end developer*, terkadang kita memiliki data berbentuk *array*, namun hanya sebagian dari elemennya saja yang diperlukan. Elemen tersebut diambil berdasarkan kondisi tertentu sesuai dengan kebutuhan tampilan UI (*User Interface*). Hal ini dapat kamu lakukan menggunakan *method* .filter() pada variabel *array* yang kamu miliki. .filter() berfungsi untuk membuat sebuah *array* baru dengan memperhatikan kondisi tertentu pada setiap elemen dari *array* yang sudah ada.

*Callback function* pada .filter() memiliki 3 parameter yang dapat digunakan sesuai kebutuhan:

1. *value* yaitu isi dari tiap elemen yang diproses.
2. *index* (*optional*) yaitu nomor *index* tiap elemen.
3. *array* (*optional*) yaitu *array* itu sendiri.

Struktur kode dari .filter():

arr.filter(function (value, index, array){

// kode program

});

Penggunaan .filter() sangat cocok ketika kita ingin mendapatkan beberapa elemen dari variabel *array* berdasarkan kondisi yang diinginkan.

Misalnya kita memiliki variabel *array* angka dengan elemen angka 1 sampai 5. Dari data yang sudah ada, kita hanya ingin mendapatkan elemen yang nilainya lebih besar dari angka 2.

Dengan menggunakan .filter(), kita dapat melakukan hal tersebut seperti ini:

let arrAngka = [1, 2, 3, 4, 5];

let filterArr = arrAngka.filter(function (item){

return item > 2;

});

console.log(filterArr);

// Output: [3, 4, 5]

Pada kode di atas, tiap elemen akan diiterasi dan diwakilkan oleh variabel item. Jika benar isi dari variabel item lebih besar dari angka 2, maka item akan ditampilkan.

return item > 2;

Setelah proses iterasi selesai, data yang didapatkan akan disimpan pada variabel filterArr.

Contoh lain dalam menggunakan .filter()dapat kita lihat pada kode di bawah ini:

const fruits = ['Jambu', 'Jeruk', 'Apel', 'Mangga', 'Strawberry', 'Jeruk', 'Apel'];

const apples = fruits.filter(function (item){

return item === 'Apel'; // hanya menampilkan elemen array fruits 'Apel'

});

console.log(apples);

// Output: ['Apel', 'Apel']

Dari kode di atas, dapat kita lihat bahwa kita mendapatkan daftar apples dari fruits yang telah di .filter(). Sehingga .filter() memudahkan kita untuk mendapatkan data dengan kondisi tertentu dari sekumpulan data yang kita punya.

### Array.find()

Pencarian merupakan salah satu algoritma dasar pada pemrograman. Dimulai dari melakukan iterasi pada sekumpulan data dan kemudian mengembalikan nilai jika sesuai dengan kondisi yang dicari. *Array* pada JavaScript juga memiliki *built-in method* .find() yang dapat digunakan untuk melakukan pencarian.

.find() memberikan nilai balik berupa nilai tunggal (*single value*) dari elemen pertama yang ditemukan jika memenuhi suatu kondisi pada *return statement*.

*Callback function* pada .find() memiliki 3 parameter yang dapat digunakan sesuai kebutuhan:

1. *value* yaitu isi dari tiap elemen yang diproses
2. *index* (*optional*) yaitu nomor *index* tiap elemen
3. *array* (*optional*) yaitu *array* itu sendiri

Struktur kode dari .find():

arr.find(function (value, index, array){

// kode program

});

Mungkin terlihat hampir sama seperti .filter(). Namun .filter() memberikan hasil dalam bentuk *array*, sedangkan .find() hanya memberikan hasil *single value*.

Contoh lebih detail mengenai perbedaan .filter() dan .find() dapat kita lihat pada kode di bawah ini:

let people = [

{

name: 'Jane',

age: 20

},

{

name: 'Doe',

age: 15

}

];

let janeFind = people.find(function (item){

return item.name === 'Jane'

});

let janeFilter = people.filter(function (item){

return item.name === 'Jane'

});

console.log(janeFind);

// Output: {name: 'Jane', age: 20} --> bentuk object

console.log(janeFilter);

// Output: [{ name: 'Jane', age: 20 }] --> bentuk array of object

Contoh lainnya:

let number=[1, 2, 3, 4, 5, 6, 7];

let moreThanThree = number.find(function (data){

return data > 3

});

let lebihDariTiga = number.filter(function (data){

return data > 3

});

console.log(moreThanThree)

// Output: 4

console.log(lebihDariTiga)

// Output: [4,5,6,7]

Sehingga dari kedua contoh di atas, kita dapat mengambil kesimpulan bahwa .find() baik digunakan jika kita hanya membutuhkan *single value* dari elemen array yang kita cari sesuai dengan kondisi yang kita butuhkan.