JAVA SCRIPT

## DAFTAR ISI

[**DAFTAR ISI 2**](#_Toc148720717)

[**I. TIPE DATA DI JAVASCRIPT 3**](#_Toc148720718)

[**II. VARIABEL 5**](#_Toc148720719)

[**III. POP UP BOX 6**](#_Toc148720720)

[**IV. CONTROL FLOW 6**](#_Toc148720721)

[**V. FUNCTION 7**](#_Toc148720722)

[**VI. REFACTORING 8**](#_Toc148720723)

[**VII. SCOPE 8**](#_Toc148720724)

[**VIII. REKURSIF 8**](#_Toc148720725)

[**IX. ARRAY 9**](#_Toc148720726)

[**X. OBJECT 10**](#_Toc148720727)

[**XI. DOCUMENT OBJECT MODEL (DOM) 12**](#_Toc148720728)

[**XII. DOM TREE 12**](#_Toc148720729)

[**XIII. DOM SELECTION 13**](#_Toc148720730)

[**XIV. DOM MANIPULATION 14**](#_Toc148720731)

[**Manipulation Element 14**](#_Toc148720732)

[**Manipulation Node 15**](#_Toc148720733)

[**XV. DOM EVENTS 16**](#_Toc148720734)

[**XVI. DOM TRAVERSAL 17**](#_Toc148720735)

[**Prevent Default 18**](#_Toc148720736)

[**Event Bubling 18**](#_Toc148720737)

## TIPE DATA DI JAVASCRIPT

* Number

Tidak ada *Integer* di dalam bahasa pemrograman Javascript yang ada hanyalah floating point (number). Dimana number ini dianggap bilangan oktal sehingga jika kita menuliskan 0 di awalnya misalkan 022 sama dengan 18.

Dan juga apabila kita menuliskan 0xFF dengan tipe number maka akan dianggap bilangan Hexa.

Terdapat Angka Spesial di dalam tipe Number :

* Infinite
* -Infinite
* NaN (Not a Number)

\*jika sebuah bilangan dibagi dengan string maka jumlah string akan di proses menjadi angka.

|  |  |
| --- | --- |
| Nama Operator | Operator |
| Aritmatika | +, -, \*, /, % |
| Penugasan | =, +=, -=, \*=, /=, %= |
| Perbandingan | == ( mengecek nilainya )  != ( tidak sama dengan )  === ( mengecek tipe datanya )  > ( lebih besar dari )  < ( lebih kecil dari )  >= ( lebih besar dan sama dengan)  <= (lebih kecil dan sama dengan) |
| String | + |
| Kondisional | Kondisi ? true : false |
| TypeOf | typeof( operand )  \*untuk memeriksa tipe data dari sebuah nilai  Unary = memerlukan 1 operand.  Binary = memerlukan 2 operand.  Ternary = memerlukan 3 operand. |

Tipe tipe Operator :

* String ( “ … “ , ‘ … ‘ , ` … `)

Tipe data berupa text. Kita bisa melakukan escape character dengan menggunakan \” atau \’.

* Boolean ( True, False )
* Object
* Function
* Undefined
  + Undefined : adalah nilai variabel kosong yang belum diisikan.
  + Null : adalah nilai kosong.

## VARIABEL

Variabel pada javascript terdiri dari var, let, dan juga const. Sekarang kita dituntut untuk tidak lagi menggunakan var dan sebaiknya menggunakan let atau const.

Var : akan menghasilkan nilai *undefined* jika kita memanggil sebelum var-nya.

\*jika menggunakan var kita bisa mengakses variabel yang ada di dalam looping.

\*ini akan error jika block scope nya dimasukkan ke dalam function scope, karena javascript menganut function scope.

IIFE atau SIAF

IIFE (Immediately Invoked Function Expression) fungsi yang dipanggil langsung ketika dibuat.

SIAF (Self Invoking Anonymous Function) / *function tidak bernama* dibuat agar variabel functionnya tidak bisa diakses dari luar.

Ex : ( *function() {*

*for(var i = 0; i < 10; i++) {*

*console.log(i);  
}*

*} () );*

*\*maka tidak akan bisa mengakses i di dalam function. Sekarang dengan perkembangan js kita cukup dengan menggunakan let karena sama dengan function diatas.*

Let : akan menghasilkan nilai *error* jika kita memanggil sebelum let-nya.

Const : nilai di dalam variabelnya akan tetap maka gunakan const. Jika diubah atau di isi ulang maka akan error.

Jika di dalam object kita masih bisa merubah assigment di dalam objectnya, namun jika diganti dengan object baru maka akan error. Begitu pula dengan array, tapi kita bisa menambahkan array baru di dalamnya, kita bisa menggantinya dengan syarat tidak menganti semuanya.

Deklarasi = mendaftarkan variabel kedalam lingkup yang sesuai. ( var, let, const ).

Inisialisasi = menyediakan memori untuk variabel. Ex: var x (x adalah inisialisasi).

Assignment = untuk menetapkan nilai untuk variabel.

## POP UP BOX

alert(“Hello World”); 🡨 Membuat Alert

prompt(“Siapa nama anda ?”); 🡨 Mengembalikan nilai inputan ( string )

confirm(“Are you sure?”); 🡨 Mengembalikan nilai true or false.

## CONTROL FLOW

*Perulangan* (Looping), adalah dimana kita bisa mengeksekusi statement secara berulang.

Perulangan ada for, while, do while, dan forEach.

*Pengkondisian*, adalah sebuah kadaan dimana kita bisa memilih statement yang berbeda berdasarkan kondisi yang diberikan.

Pengkondisian ada if, if else, if else if else, switch.

## FUNCTION

Function di dalam Javascript adalah kunci utama yang membuat js menjadi powerfull. Dengan menggunakan function kita bisa :

* Program yang ada di dalam program lainnya, yang dapat dipanggil kembali.
* Fundamental building bloks javascript.
* Kumpulan statement untuk melakukan tugas.
* Mengurangi perulangan code.

Kategori Function :

* Built-in Function ( function bawaan javascript )
* User-Defined Function ( function yang dibuat oleh user )
  + Function Declaration

*Adalah function yang memiliki nama. Contoh :*

*function hitung (a,b) { return a + b; }*

*console.log(hitung(1,2)); // 3*

* + Function Expression

*Adalah function yang tidak memiliki nama namun harus didefinisikan terlebih dahulu / sebagai argument. Contoh :*

*let jumlah = function(a, b) { return a + b; }*

*console.log(jumlah(1,2)); // 3*

Function Expression :

- harus didefinisikan terlebih dahulu.

- sebagai closure

- sebagai argument untuk function lain.

\*parameter variabel di dalam kurung function, argument nilai yang dikirimkan ke parameter.

\*jika parameter > banyak dari argument maka (undefined), tapi jika argument > parameter maka (diabaikan).

return arguments = mengembalikan array dari argument yang dimasukkan.

*\*****return mengembalikan nilai & merupakan cara memberhentikan function.***

## REFACTORING

Adalah mengubah code menjadi lebih baik tanpa merubah functional code, yang dimana ini bertujuan untuk meringkas code dan juga menghindari perulangan code.

## SCOPE

Scope adalah ruang lingkup dari sebuah variabel. Terdapat 2 Scope yang ada di dalam Javascript, yaitu :

* Block Scope (sama seperti kita menggunakan let dan const)
* Function Scope (sama seperti kita menggunakan var)

***“use strict”;* digunakan untuk membersihkan variabel local yang menjadi variabel global.**

*Misalkan :* a = 20;

Maka secara otomatis js akan menuliskan var a = 20; dan itu menjadi variabel global.

## REKURSIF

Sebuah function yang memanggil dirinya sendiri. Di dalam konsep rekursif terdapat *‘base case’*  yang dimana artinya kondisi akhir function yang mengembalikan nilai.

Contoh : akan memanggil function sampai bernilai 0 maka akan berhenti / hampir sama seperti looping.

*function* cetakAngka(*n*) {

*// BaseCase*

    if(n === 0) {

        return;

    }

    console.log(n);

    cetakAngka(n-1);

}

cetakAngka(10);

## ARRAY

Array adalah Tipe data yang bisa menampung lebih dari 1 nilai. Array mendeskripsikan suatu nilai (key, value).

* Bisa Menampung > dari 1 nilai.
* Variabel yang sakti
* Key index dimulai dari 0.
* Array berupa tipe-nya Object

|  |  |  |
| --- | --- | --- |
| No. | Array Method | Penjelasan |
| 1 | length | Menghitung Jumlah Element |
| 2 | join | Menggabungkan seluruh array dan bisa memberikan penanda - misalkan *join(‘-‘);* |
| 3 | push | Menambahkan element baru di akhir *push(‘Izuchii’);* |
| 4 | pop | Menghilangkan element terakhir pada array. |
| 5 | shift | Menghilangkan element pertama pada array. |
| 6 | unshift | Menambahkan element baru pada awal array *unshift(‘Izuchii’);* |
| 7 | slice | Mengambil beberapa bagian untuk membuat array baru.  *var arr2 = arr.slice(1, 3);* |
| 8 | splice | Menyisipkan sebuah element di tengah tengah.  *splice(index awal, mauDihapusBerapa, elementbaru1, elementabaru2)* |
| 9 | forEach | Mengembalikan array secara keseluruhan.  Ex : *angka.forEach(function(e) {  console.log(e);*  *});* |
| 10 | map | Map mengembalikan nilai array dan mengeluarkan dari array.  Ex : *angka.map(function(e) {  console.log(e);*  *});* |
| 11 | sort | Mengurutkan isi Array.  \*tapi cukup unik jika mengurutkan angka 10 keatas harus dengan function berikut.  Ex : *angka.sort(function(a, b) {*  *return a - b;*  *});* |
| 12 | filter | Mencari dengan mengembalikan banyak nilai  Ex : *var angka2 = angka.filter(function(e) {return e > 5; });* |
| 13 | find | Sama seperti filter tapi mengembalikan *1 nilai*. |

## OBJECT

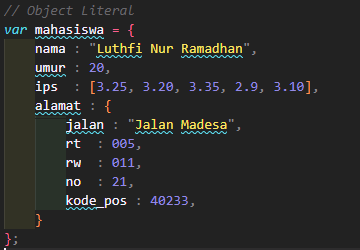
Object, merupakan kumpulan nilai yang memiliki nama. Maksudnya bagaimana ?

* Jika array menggunakan [key, value] tetapi jika object {name, value}
* Kita juga bisa memberikan function di dalamnya.
* **Variabel di dalam object disebut *“Property”.***
* **Function di dalam object disebut *“Method”.***

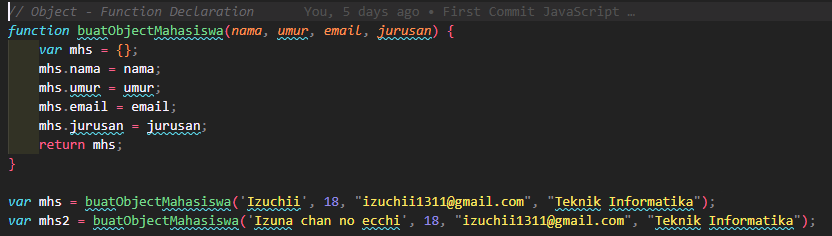
Membuat Object

* Object Literal (biasa).

Kita cukup membuat object dengan menggunakan { … }.

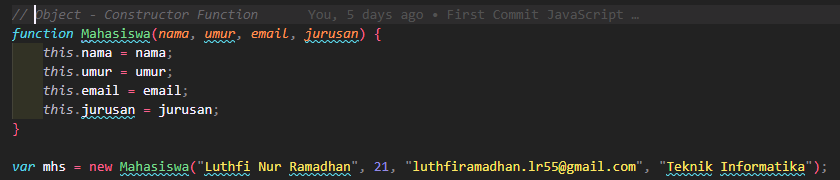


* Menggunakan function declaration.
* Di dalam function kita harus membuat variabel bahwa argument yang ditangkap akan dimasukkan ke dalam object.
* Return variabel tersebut, untuk mengebalikan nilai dan memberhentikan function.



* Menggunakan constructor function.
* Constructor function adalah function yang khusus digunakan untuk menerima object.
* Biasanya nama function diawali dengan huruf besar.
* Tidak memerlukan variabel kosong di awal, cukup dengan keyword *this.*
* Saat menambahkan data baru ke dalam object diawali dengan keyword *new*.

var mhs = *new* Mahasiswa(…, …);



> *This* merupakan sebuah keyword spesial, yang secara otomatis didefinisikan pada setiap function.

*\*Sebenarnya dalam membuat object ada this yang didefinisikan.*

*> jika di dalam object literal this mengembalikan object yang bersangkutan.*

*jika di dalam function declaration this mengembalikan object global (window).*

*jika di dalam constructor function this mengembalikan object yang baru dibuat*

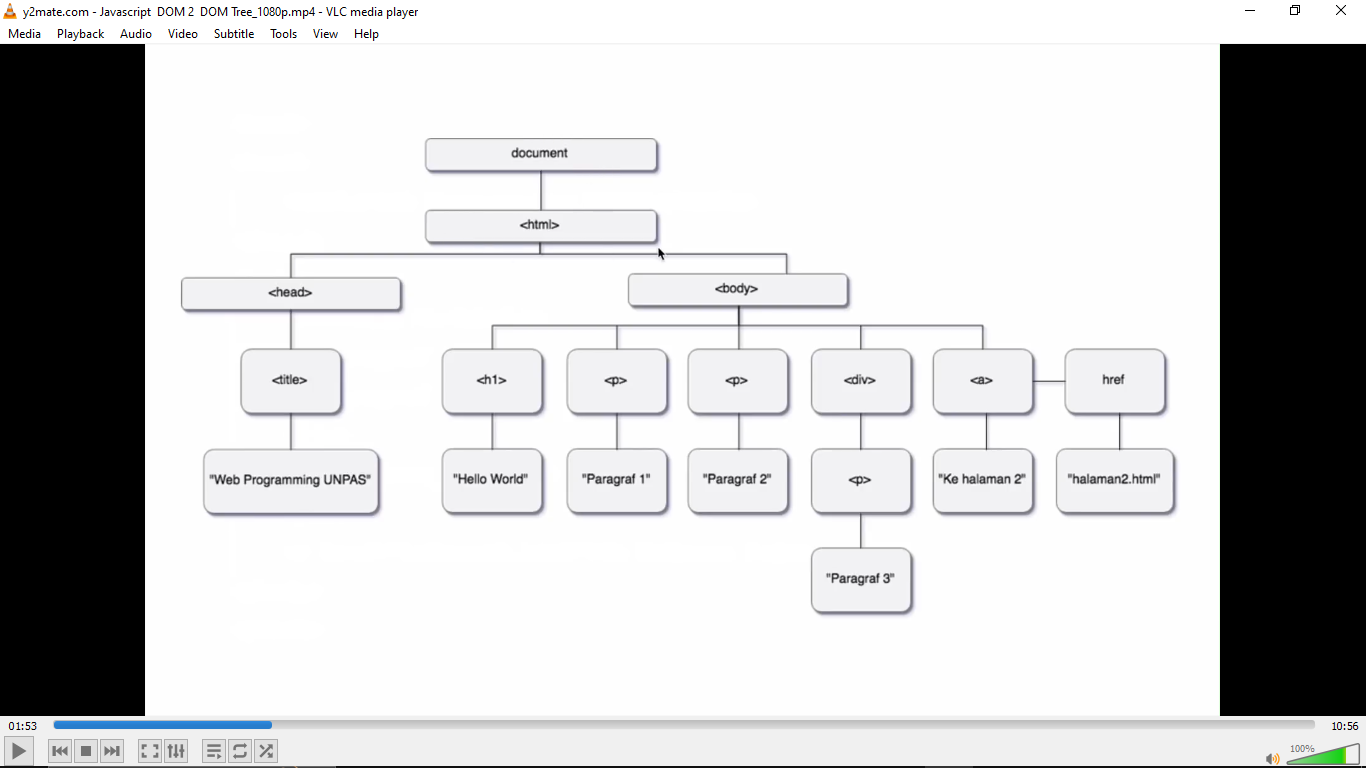
*(object yang baru dibuat yang diawali dengan keyword 'new').*

* Menggunakan object.create().

## DOCUMENT OBJECT MODEL (DOM)

DOM (DOCUMENT OBJECT MODEL, Merupakan Representasi element HTML pada dokumen menjadi sebuah object.

## DOM TREE



DOM Tree, setiap attribute / element html disebut dengan *simpul* atau *node.*

\*DOM Tree juga merupakan representasi element HTML dalam bentuk tree.

> window (bisa mengelola seperti scroll, ukuran webnya, dl)

> document (mengelola element html)

> history

> location

Node List, memilih lebih dari 1 node dan tidak peduli jenis nodenya apa.

Misalkan : element <p> atau text “Hello World”

HTMLCollections, kumpulan node tapi khusus element HTMLnya saja.

Misalkan : element <p> atau element <body>

Struktur Hierarki Node

> root node (document)

> parent node (body, head)

> child node

> children (child tapi element html)

> siblings (child yang memiliki parent yang sama)

## DOM SELECTION

DOM Selection, adalah function javascript dimana kita bisa memilih element element HTML.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Function | Keterangan | Tipe |
| 1 | getElementById( ) | Memilih element html berdasarkan id. | Element |
| 2 | getElementByClassName( ) | Memilih element html berdasarkan nama class pada element html. | HTML Collection |
| 3 | getElementByTagName( ) | Memilih element html berdasarkan tag html. | HTML Collection |
| 4 | querySelector( ) | Mencari element berdasarkan selector. | Element |
| 5 | querySelectorAll( ) | Mencari semua element berdasarkan selector | NodeList |

HTML Collection / Node List : \*memilih semua element yang dicari (boleh lebih dari 1).

\*berbentuk seperti array dan dimulai dari index ke - 0.

\*agar bisa memberikan style kita panggil juga indexnya.

Selector : \*bisa dianggap seperti mencari selector di css.

\*menangkap 1x selector yang pertama saja ia temui.

*Contoh menangkap element tanpa ada selectornya*

*const sMerah = document.querySelector('input[name=sMerah]');*

Dengan begitu kita bisa memberikan *style* pada element yang kita pilih atau *innerHTML* untuk merubah isi textnya.

## DOM MANIPULATION

### Manipulation Element

|  |  |  |
| --- | --- | --- |
| No. | Function | Keterangan |
| 1 | element.innerHTML | Digunakan untuk merubah isi dari sebuah tag yang sudah diseleksi. |
| 2 | element.style.<property> | Digunakan untuk merubah styling css yang sudah diseleksi. |
| 3 | element.setAttribute(‘ ’, ‘ ’)  element.getAttribute(‘ ‘)  element.removeAttribute(‘ ‘) | Menambahkan attribute  Melihat isi dari attribute  Menghapus isi dari attribute  Memanipulasi attribute pada html.  \*attribute seperti id, name, class, href, label. |
| 4 | element.classList. ...()  classList.add( )  classList.remove( )  classList.toggle( )  classList.item( )  classList.contains( )  classList.replace(classLama, classBaru) | Memanipulasi untuk mengelola class.  Digunakan untuk menambahkan class baru.  Digunakan untuk menghapus class.  Jika belum ada class akan ditambahkan jika ada maka akan dihapus.  Digunakan untuk mengetahui class tertentu di dalam item.  Mengecek / mencari class tertentu.  Mengganti class yang ada dengan class yang baru |

### Manipulation Node

memungkinkan kita bisa menambahkan node baru / element baru

|  |  |  |
| --- | --- | --- |
| No. | Function | Keterangan |
| 1 | document.createElement( ) | Menambahkan element baru.  Contoh :  *const li = document.createElement('li');* |
| 2 | document.createTextNode( ) | Menambahkan text untuk element baru.  Contoh :  *const isiLi = document.createTextNode("Item Baru");* |
| 3 | node.appendChild( ) | Menyimpan isi textNode ke dalam Element.  Contoh:  *li.appendChild(isiLi);* |
| 4 | node.insertBefore(“ “, “ “) | Menyimpan Element baru sebelum element (… dan …)  Contoh:  *ul.insertBefore(li, li2);* |
| 5 | parentNode.removeChild( ) | Menghapus node  Contoh:  *const link = document.querySelector('section#a a');*  *sectionA.removeChild(link);* |
| 6 | parentNode.replaceChild( ) | Mengganti node lama dengan node baru ('nodeBaru', 'nodeLama').  Contoh:  *const sectionB = document.querySelector('section#b');*  *const p4 = sectionB.querySelector('p');*  *const h2 = document.createElement('h2');*  *const isiH2 = document.createTextNode('H2 ini dibuat dengan menggunakan JavaScript');*  *h2.appendChild(isiH2);*  *sectionB.replaceChild(h2, p4);* |

## DOM EVENTS

DOM Events adalah sebuah kejadian yang nantinya akan terjadi di dalam DOM, misalkan ketika di ‘klik’ jalankan fungsi berikut.

* Event Handler (Cara lama) – onclick
* Inline HTML Attribute (Akan menggabungkan hasil eventnya ke dalam element html – inline).
* Element Method (Menambahkan method pada element HTML).
* Kita hanya bisa memberikan event 1x saja.
* Perubahan terakhir akan menimpa perubahan sebelumnya.
* addEventListener()
* Kita bisa memberikan lebih dari 1 perubahan dalam 1 event.
* Perubahan terakhir akan menambah dari perubahan sebelumnya.

Event List, (mouse event, keyboard, resources, focus, view, form, dll)

contoh : mouse event (click, dblclick, mouseover, mouseenter, mouseup, wheel).

\*after() akan menampilkan / menambahkan data atau element setelahnya.

misal di latihan.js : tUbahWarna.after(tombol);

\*maka akan menyimpan tombol setelah tUbahWarna

fungsi pembulatan js / untuk menghilangkan angka koma setelah random number

round( )= pembulatan ke angka terdekat

floor( )= pembulatan kebawah

ceil( ) = pembulatan ke atas

window.innerWidth = untuk mengetahui ukuran document.

## DOM TRAVERSAL

DOM Traversal atau penelusuran DOM, adalah dimana kita menelusuri elementnya berdasarkan element yang kita seleksi di awal. Tujuannya adalah untuk menelusuri parentnya dengan menggunakan *parentElement*.

DOM Traversal Method

|  |  |  |
| --- | --- | --- |
| No. | Method | Keterangan |
| 1 | parentNode( ) | Node |
| 2 | parentElement( ) | Element |
| 3 | nextSibling( ) | Node |
| 4 | nextElementSibling( ) | Element |
| 5 | previousSibling( ) | Node |
| 6 | previousElementSibling( ) | Element |

\*node bisa saja mengambil enter

Kapan DOM Traversal ini digunakan, misalkan kita mempunyai sebuah card dan di dalam card terdapat tombol x. Nah x merupakan bagian dari card artinya tombol x adalah child dari card. Ketika tombol x di tekan maka card akan menghilang. Contoh :

*const close = document.querySelectorAll('.close');*

*const card = document.querySelectorAll('.card');*

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

*close[i].addEventListener('click', function(e) {*

*// close[i].parentElement.style.display = 'none';*

*e.target.parentElement.style.display = 'none';*

*});*

*}*

### Prevent Default

Adalah sebuah method yang ada di dalam sebuah object. Misalkan function (e) { … }

Object yang dimaksud adalah e, tujuannya untuk menghentikan aksi default yang dilakukan oleh element.

Misalnya kita membuat tombol close

*<!-- <span class="close">×</span> -->*

Diganti menjadi

*<a href="" class="close">×</a>*

\*kita ubah dengan menggunakan a.

karena dengan menggunakan a maka akan memindahkan ke halaman lain, maka akan berkedip(refresh halaman).

maka dengan kita menuliskan misalkan.

*e.preventDefault();*

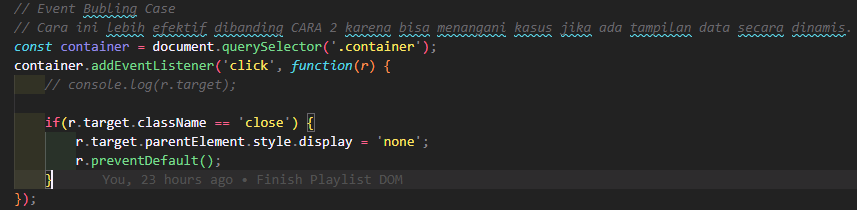
maka aksi a ketika di klik, tidak akan berkedip atau merefresh halamannya.

### Event Bubling

Dalam kasus traversal, kita sudah memberikan class close, dan kita sudah memberikan event pada sebuah element. Maka event tersebut akan berlaku untuk element element pembungkusnya juga. Dan akan berlaku juga untuk event pembungkusnya juga.

Event Bubling ini juga sangat terpakai untuk menangani kasus yang dinamis / menambahkan element secara inspect. Karena jika kita menggunakan selectorAll maka berbentuk node dan jumlahnya pasti sama dengan jumlah awalnya.

Maka oleh begitu event bubling akan menangkap element pembungkus terluarnya, lalu ketika di klik jalankan fungsi yang mengecek jika ada className dengan nama ... maka lakukan sesuatu.

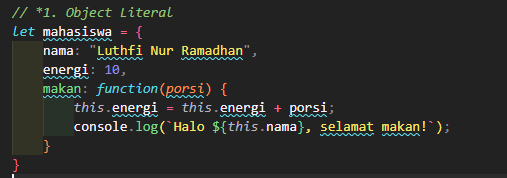
Contoh : 

## PROTOTYPE

### Object Literal (Biasa)

Kita bisa membuat object dengan literal / biasa seperti berikut. Namun tetapi object literal ini mempunyai kekurangannya, yaitu :

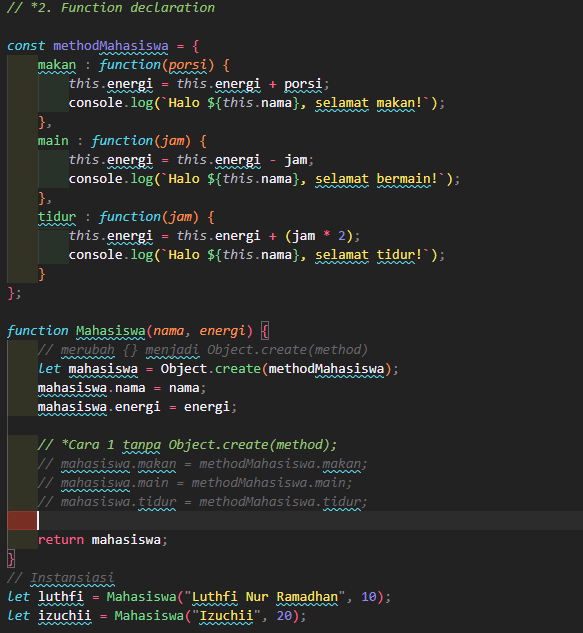
* Jika ingin membuat object baru kita harus melakukan duplikasi code dengan nama object yang berbeda.
* Jika ingin melakukan instansiasi object baru kita tidak boleh memiliki nama yang sama.



### Function Declaration & Object.create( )

Kita juga bisa membuat object dengan function declaration, namun tetap dengan menggunakan ini kita masih memiliki kekurangan :

* Jika tanpa menggunakan let mahasiswa = { } dan kita langsung membuat method di dalam objectnya. Itu yang terjadi ketika kita menggunakan objectnya & *kita tidak memasukkan nilai ke dalam methodnya maka method tetap dijalankan* dan itu mempengaruhi optimalisasi.
* Maka diganti dengan menggunakan Object.create( ) tidak lagi menggunakan { }. Tujuannya agar kita tidak menjalankan method di dalam object yang tidak ingin kita jalankan, namun syaratnya kita harus membuat methodnya di luar functionnya. Lalu kita panggil method tersebut di dalam Object.create( ).
* *Sebenarnya kita tidak memerlukan method di luar function yang membuat kita menuliskan baris code secara 2x.*



* Sebenarnya dengan ***Object.create()*** itu cukup membantu, karena kita bisa saja memberikan method di luar function demi tidak menjalankan method yang tidak kita inginkan.

Meskipun kita menggunakan let mahasiswa = { };

Kita tetap bisa memanggil object diluar function dengan memanggil nama objectnya ketika ingin memberikan isi dari property.

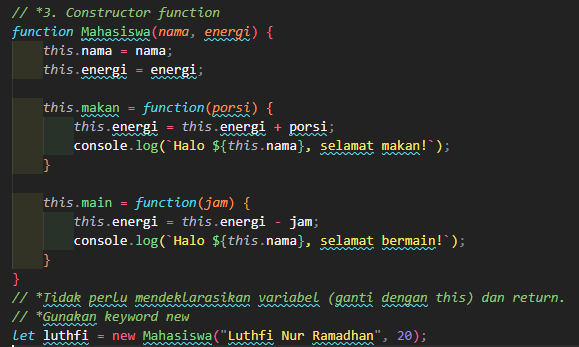
***Inisialisasi =*** *adalah dimana kita memberikan nama pada sebuah function atau variabel.*

***Instansiasi =*** *adalah dimana kita menjalankan object dengan mengisikan object tersebut.*

### Constructor Function

Constructor Function ini memang khusus untuk membuat Object, karena jika kita membuat method di dalamnya dan tidak digunakan maka tidak akan terpanggil, alhasil methodnya akan di jalankan dengan kita memanggilnya saja.

Kita juga tidak memerlukan variabel awal untuk menandakan bahwa dia object, kita juga tidak memerlukan return variabel awal tersebut, cukup ganti dengan keyword **this.** Saat melakukan instansiasi kita hanya perlu menambahkan keyword **new NamaObject.**



### Prototype

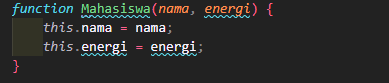
Sebelumnya kita kenali dulu dengan prototype, Prototype awalnya merupakan object kosong, dan dapat ditambahkan anggota ke dalamnya. Array, function, object merupakan sebuah object. Prototype mirip seperti konsep pewarisan atau inheritance, tetapi di js disebutnya prototyple inheritance.

Cara dengan *Object Prototyple Inheritance.*

> Jika dengan constructor func sebenarnya di dalamnya kita tetap membuat variabel dengan nama this dan return this. Dan this tidak hanya membuat {} object kosong saja, tapi membuat Object.create() tapi didalamnya memanggil Object mahasiswa.

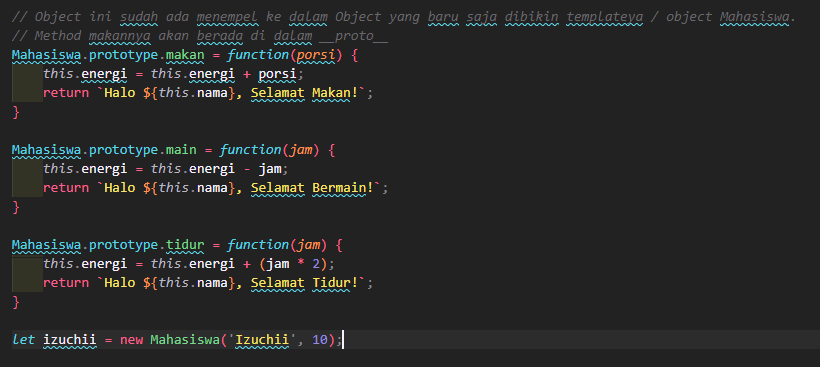
> let this = Object.create(Mahasiswa.prototype);

> ada sebuah property yang secara default dijadikan parent oleh js nya.



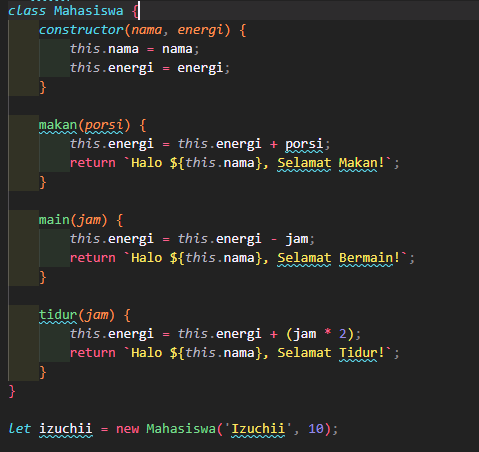
> Jadi secara default constructor function sudah punya parent 'prototype'. Maka kita tidak perlu membuat object baru yang kita jadikan sebagai parent.

> untuk mengelola methodnya kita cukup memanfaatkan prototypenya.



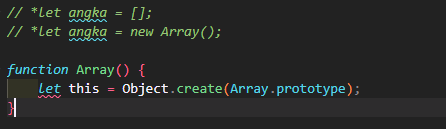
Contoh dengan *Object Class.*

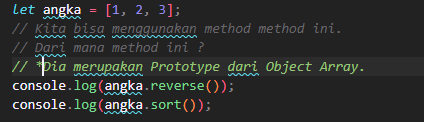
Sama saja dengan konsep prototype karena di belakang layar yang dilakukan sama dengan cara diatas.



Dengan memahami prototype memungkinkan kita bisa tahu tipe data apapun pada js, yang apapun di dalamnya adalah object dan pasti mempunyai \_\_proto\_\_.

Misalkan array maka di dalam js-nya pasti ada function yang namanya :



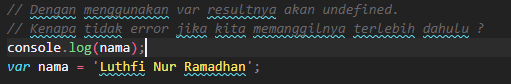


Makannya jika kita cek \_\_proto\_\_ dari array nya maka akan ada banyak method method di dalamnya.

Array ini mewarisi semua method dari object Array. Kita bisa mengecek methodnya dengan Array.prototype atau Object.prototype atau Number.prototype

## CLOSURE

### Execution Context, Hoisting & Scope.



Di dalam execution context terdapat 2 phase yaitu *Creation phase* dan *Execution phase.*

\**Creation Phase pada global context.*

Javascript akan mengecek apakah ada var atau function. Jika ada dia akan membuat nama variabel & akan diisi dengan udefined jika ada function maka akan diisi function itu sendiri.

var nama = undefined; -> nilai diisi dengan undefined

nama function = fn() -> mengembalikan function itu sendiri

\*ini merupakan konsep *Hoisting*.

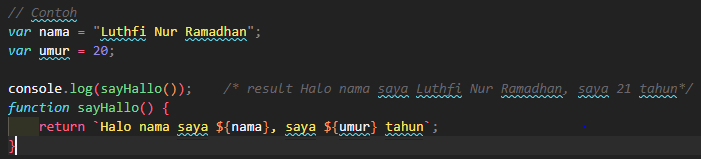
Meskipun kita menuliskan dibawah secara otomatis akan dinaikkan ke atas.

Selain 2 hal diatas ada juga yang didefinisikan.

Window = global Object

this = window

\*Execution Phase (menjalankan baris programnya dari atas ke bawah).



- Ketika bertemu function execution seolah olah dia menjalankan execution contextnya sendiri.

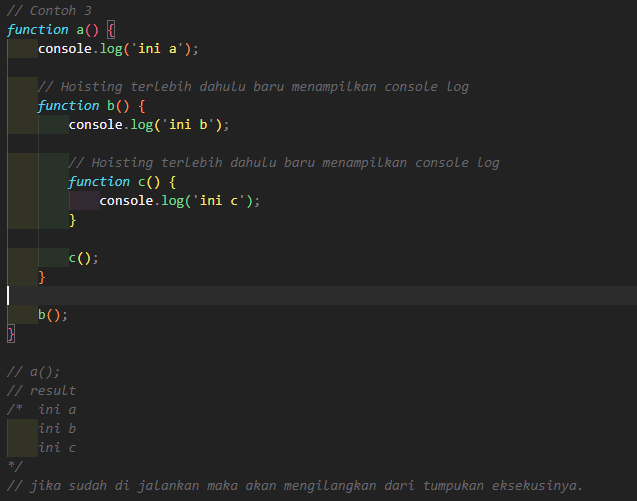
- jadi jika ketemu function excecutionnya tidak jadi global context lagi.

- jika bertemu function dia seolah olah membuat local execution context yang di dalamnya sama ada creation & execution phase.

- kita bisa akses window, ke arguments (yang nantinya bisa berisi apapun yang kita masukan ke dalam parameter dari function). Dan di dalamnya ada Hoisting juga.



Kita definisikan terlebih dahulu nama di dalam global, lalu kita jalankan *hoisting* yang akan menaikkan functionnya terlebih dahulu kemudian function itu akan menjalankan terlebih dahulu execution context local di dalam function. Lalu jalankan code nya



Contoh lainnya, function ini akan menjalankan atau melakukan hoisting terlebih dahulu kemudian menjalankan execution local milik function a. Didalam fn a kita hoisting dahulu fn b lalu jalankan console.log milik a. Begitu pula seterusnya lalu tinggal kita panggil functionnya agar bisa menampilkan console.log dari masing masing function.

**\*hoisting itu seperti kita menaikkan tiang bendera. Disini kita menaikkan variabel & function untuk di definisikan lebih dahulu.**

Kita bisa mengecek kosep execution context dan hoisting dengan menggunakan bantuan website <https://pythontutor.com/>

### Closure

- Merupakan kombinasi antar function dan lingkungan leksikal (lexical scope) di dalam function tersebut.

- Sebuah function ketika memiliki akses ke parent scope-nya, meskipun parent scope-nya sudah selesai dieksekusi.

- kemampuan suatu fungsi (function) untuk mengakses variabel-variabel yang ada di luar fungsi tersebut, bahkan setelah fungsi tersebut selesai dieksekusi.

\*sebuah function bisa memiliki function di dalamnya, tapi kasus tersebut bukan sebuah closure, **akan disebut closure jika di dalamnya memiliki akses ke parent scopenya**.

\*jadi closure itu di dalam sebuah function memiliki function lagi yang dapat mengakses parent scope nya, meskipun functionnya sudah selesai dieksekusi.

Lexical Scope

\*inner function, itu bisa memiliki akses ke parent variabel-nya, hingga mencarinya ke window.

\*bisa mengakses variabel di luar functionnya.

\*inner function membutuhkan variabel di parent scopenya maka disebut dengan (closure)

Misalkan :

function init() {

let nama = "Luthfi"; // Local variabel

function tampilNama() { // Inner Function (closure)

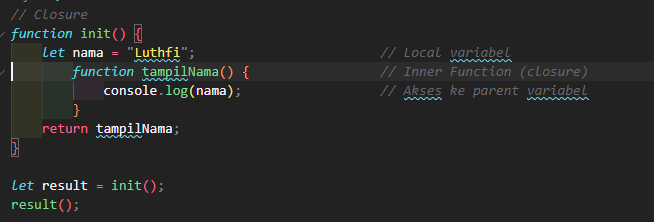
console.log(nama); // Akses ke parent variabel

}

tampilNama();

}

init();



* jika kita menggunakan console.log kan itu tidak mengembalikan nilai, oleh karena itu diganti dengan menggunakan return tampilNama.
* Namun hasilnya akhir akan kosong, karena functionnya baru berjalan sebagian

console.dir(tampilNama);

* jika kita menggunakan dir(tampilNama) maka akan melihat isi function nya.

dimana akan ada scopes dan ada closure di dalamnya

Untuk menjalankan dan melihat hasil akhir dari functionnya, kita harus memasukkan nya ke dalam variabel.

Lalu kemudian menjalankan variabel tersebut.

Jika console.log() akan menghasilkan nilai string tanpa kutip.

Sedangkan return akan menghasilkan nilai string dengan kutip.

**Kenapa sih harus membuat closure ?**

1. **Bisa untuk membuat function factories**
2. **Bisa untuk membuat private method**

#### Function Factories

sudah mengisikan data waktu tinggal menunggu data dari nama, jadi functionnya jika tidak diisikan argument untuk nama maka function baru berjalan setengahnya.

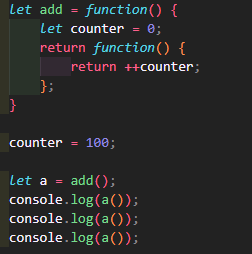


Cara berikut adalah dengan Exspression function jadi tidak memiliki nama dan langsung mereturn functionnya. Jika ingin ada nama pada functionnya, kita bisa melakukannya dengan memberikan nama pada *inner functionnya* lalu return setelahnya nama dari *inner functionnya*.

#### Private method

misalkan nilai awal counter 0, tiba tiba karena semakin banyak code counter tertimpa jadi 100.

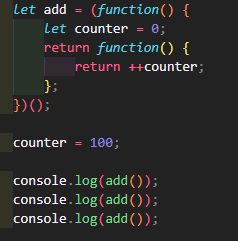
Maka semuanya akan terpengaruh



*\*Sebenarnya agak aneh kita memasukkan nya ke dalam variabel a untuk menjalankan function exspressionnya. Kita bisa merubahnya dengan menggunakan Immediately Invoke Function.*

#### **Immediately Invoke Function**

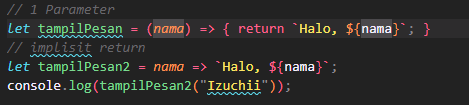
Maka menjadi seperti berikut



## ARROW FUNCTION

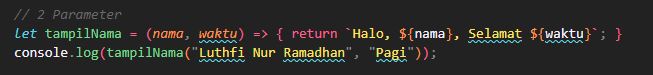
Arrow Function adalah fungsi yang dideklarasikan dengan tanda arrow(panah) =>. Arrow function juga merupakan bentuk lain yang lebih ringkas dari Function Expression.

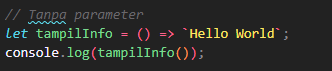
Berikut ini adalah contoh Arrow Function :



\*kita bisa melakukan dengan cara { return “Hello World”; }

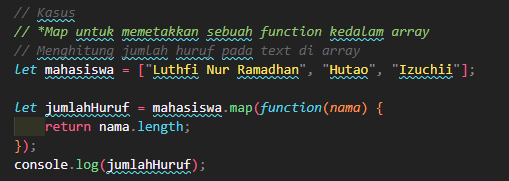
\*namun bisa juga dengan implisit return dimana kita tidak perlu menuliskan { dan return }



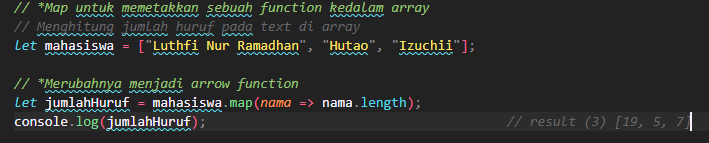


Contoh dalam penggunaan Arrow Function :

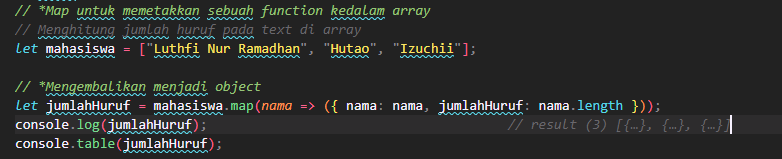
\*jika kita tidak menggunakan arrow function, biasanya baris function yang kita tulis seperti berikut.



Sekarang jika code diatas diubah menjadi arrow function

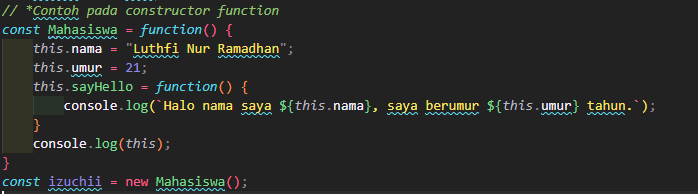


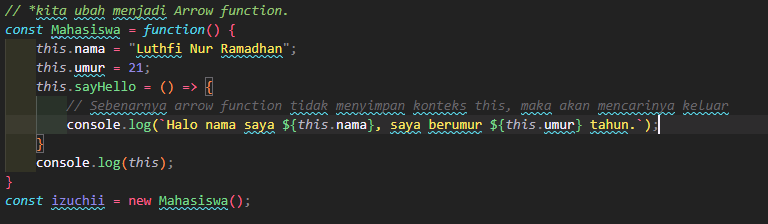
Contoh kasus 2 yang merubah array dan dikembalikan menjadi object



***\*map = untuk membuat array baru dari memanggil sebuah function pada setiap elemen array.***

### This pada arrow Function

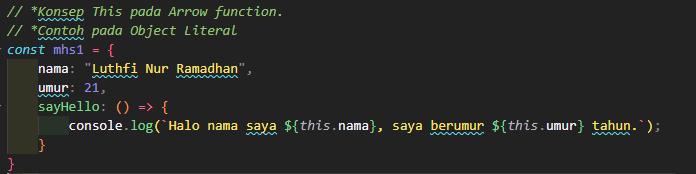




Hasilnya akan menampilkan data nama dan umur

\*kita tidak bisa merubah semuanya menjadi arrow function. Sehingga tidak bisa constructor function diubah semuanya menjadi arrow function kecuali method.

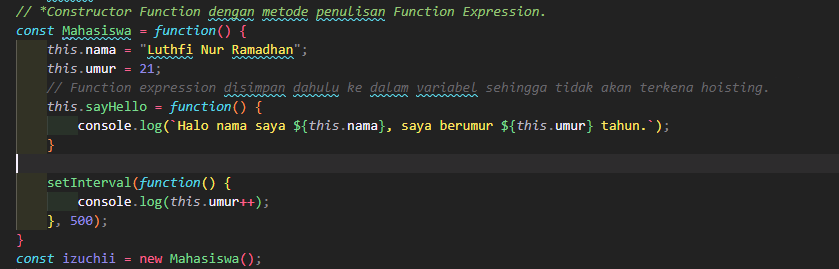
\*sebenarnya ***Arrow Function tidak memiliki konsep This***, maka ia akan mencari ke lexical scope nya atau ke Mahasiswa.



Hasilnya akan undefined

\*Jika dalam cosntructor function this.nama akan mencari ke dalam lexical scope nya.

\*namun konteksnya berbeda jika kita menggunakan arrow function karena this nama akan mencari this.nama hingga ke global / window

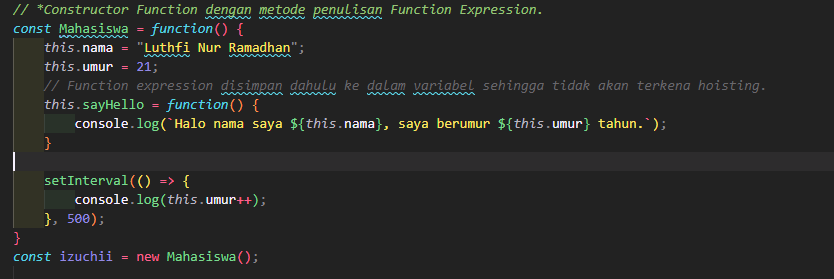


\*Method yang disimpan ke dalam function expression tidak akan terkena hoisting sehingga dapat mencari this.nama ke dalam Mahasiswa.

\*Namun akan berbeda dengan method setInterval.

setInterval tidak mencari ke dalam Mahasiswa tetapi langsung mencari umur ke konteks global, karena itu this yang ada di dalam setInterval adalah window dan jika di ++ maka akan NaN;

Jika diperbaiki seharusnya seperti ini



Dimana kita merubah set intervall menjadi arrow function karena ***arrow function tidak menganut this*** oleh karena itu this.umur akan dicari terlebih dahulu ke lexical scope hingga ke global.

Ada contoh dalam penggunaan kasus arrow function pada kasus nyata terdapat pada file thisarrowfunc.js

Arrow function tidak menganut konsep this, sehingga jika kita menggunakan this di dalam arrow function ia akan memanggil ke lexical scope nya. kapan ini digunakan bisanaya kita menggunakan this untuk memanggil data dalam konteks lexical scope.

Misalkan kita menggunakan object literal yang dimana object ini tidak memiliki this layaknya function.

ketika di dalamnya kita memiliki method dengan arrow function dan kita memangilnya dengan this maka akan mencarinya dalam konteks GLOBAL / window.

namun ketika di dalamnya memiliki method dengan anonymouse function maka this akan mencari ke lexical scope nya.

Tapi jika di dalam function constructor dimana kita memiliki this, namun di dalamnya kita memiliki function yang otomatis di jalankan misalkan setInterval() lalu kita jalankan function seperti biasa di dalamnya dan mengarahkan this, maka this yang dimaksud akan ke konsep global nya.

Namun jika kita gunakan arrow function maka konteks yang dimaksud akan function constructornya.

## HIGH ORDER FUNCTION

High Order Function itu merupakan function yang beroperasi pada function lain baik itu digunakan dalam argument, ataupun return value.

First Class Function

Javascript memperlakukan function seperti object. Yang dimana berarti function sama seperti value / integer atau string, maka function juga dapat kita simpan sebagai argument, atau return value dari function yang lain.

Contoh : function *kerjakanTugas* (mataKuliah, selesai) {  
 …

Selesai();

}

* Function *kerjakanTugas* disebut dengan High Order Function
* Selesai disebut dengan callback.

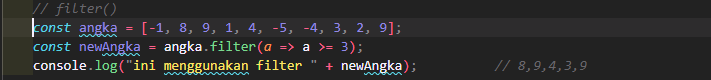
Callback adalah function yang disimpan sebagai argument.

Kenapa menggunakan High Order Function ? karena abstraksi, untuk membuat code yang dibuat terlihat lebih simpel / sederhana.

|  |  |
| --- | --- |
| Function | Keterangan |
| Array.prototype.map() | Memetakkan setiap element arraynya ke dalam setiap function sendiri dengan memiliki callback dan menghasilkan array baru.  Map = akan menghasilkan array baru.  ForEach = tidak menghasilkan array baru. |
| Array.prototype.filter() | Akan melakukan filter dan menghasilkan array baru dari hasil filternya. |
| Array.prototype.reduce() | Menggabungkan semua array baru menjadi 1 utuh dan tetap menghasilkan array baru. |

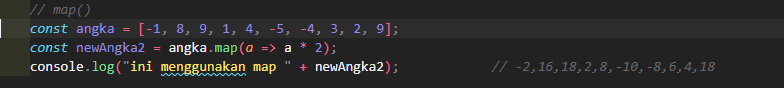
Filter()

\*memfilter semua angka yang lebih besar dari sama dengan 3 akan ditampilkan.



Map()

\*mengkalikan 2 dari semua element arraynya.



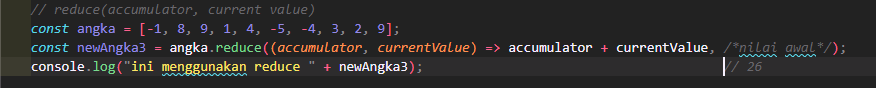
Reduce()

\*reduce memiliki 2 parameter yaitu (accumulator, currentValue), nilai awal

\*accumulator = hasil dari proses, misalkan dijumlahkan (nilai).

\*currentValue = element array yang sedang di loopingnya, misalkan array[1]

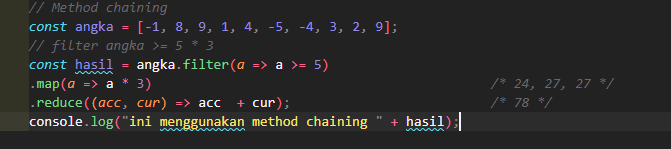
Disini kita akan menggabungkan semua nilainya dan menjumlahkan semua isinya.



Method Chaining

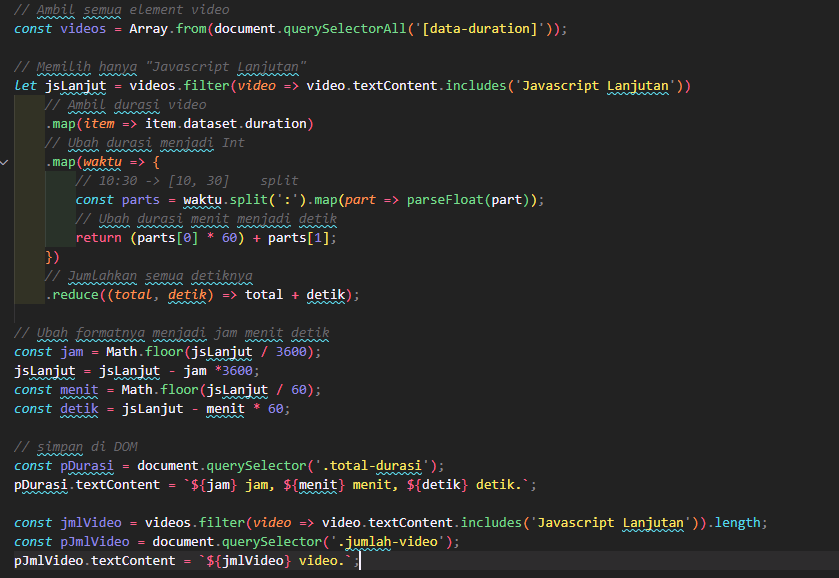
Kita dapat menggabungkan fungsi fungsi High order function dalam 1x eksekusi tidak perlu ke dalam variabel 1 per satu.

Contoh di bawah, kita filter >= 3 lalu map \* 3 dan reduce (totalkan semua).



Catatan node list tidak akan bisa menggunakan fungsi high order function karena berupa node, sedangkan untuk menggunakan map, filter, reduce harus yang berupa array.

Contoh : pada javascript lanjutan latihan High Order Function.



## TEMPLATE LITERALS

Template Literals adalah string literal yang memungkinkan ada expression di dalamnya.

String Literal

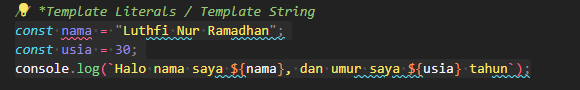
* Bisa menggunakan ‘ .. ’ , “ .. “ , atau ` .. `

\*dengan menggunakan back tick ` ` memungkinkan kita untuk membuat template literal.

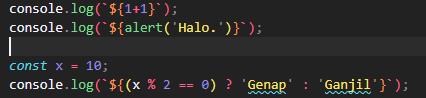
Template Literal

* Menggunakan back tick ` `
* Multi Line String
  + bisa menambahkan enter atau new line tanpa concatenation
* Embedded Expression
  + ${expression} ternary dan function termasuk kedalam expression
* HTML Fragments
  + kita bisa menuliskan tag html di dalamnya
* Expression Interpolation
  + Misalkan jika biasanya kita ingin melakukan penjumlahan di dalam string literal menggunakan concatenation, tetapi jika menggunakan template literal kita cukup gunakan expression ${a + b}
* Tagged Templates
  + Bentuk yang lebih kompleks dari template literals, dimana memungkinkan kita untuk membaca template literal melalui sebuah function.

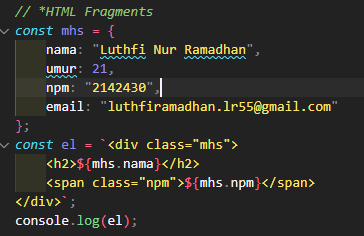
Penulisan String Literal / Template Literal



Penulisan Embedded Expression



Penulisan HTML Fragments

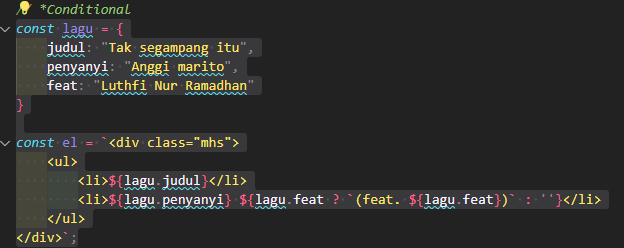


### Penerapan Template Literals Pada Kasus yang lebih nyata

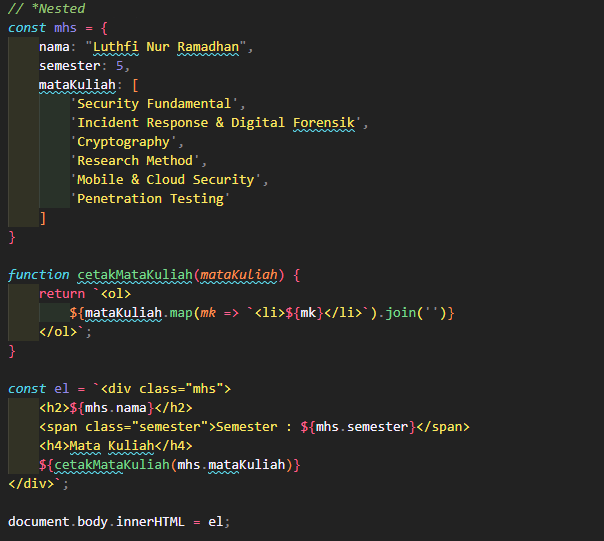
#### Looping



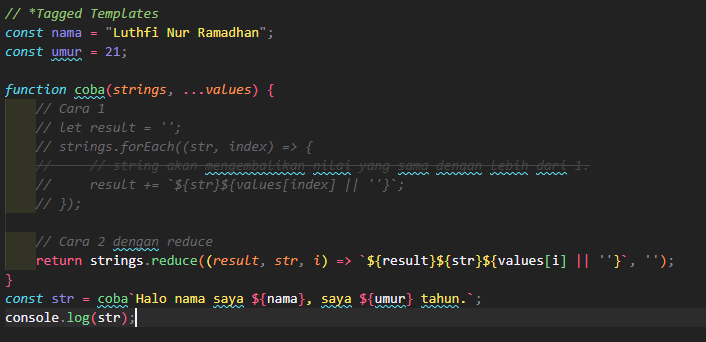
#### Conditional



#### Nested



#### Tagged Templates



\*secara defaults tag templates ini bisa menerima parameter / argument. ***Strings*** *akan berisi argument stringnya (kalimat di dalam ``) saja yang dipisahkan oleh expression ${}, parameter berikutnya bisa kita isikan nama, umur karena kita memanggil expression ${nama} ${umur}, jika saja kita tidak mengetahui berapa banyak parameter yang akan diterima maka gunakan* ***rest parameter …(namanya)*** *sehingga berapa banyak parameter pun kita tidak perlu menuliskannya 1 per 1 karena rest parameter sudah menampung semuanya.*

\*kita akan menggunakan reduce(accumulator (anggap sebagai result untuk menggabungkan - kosong), element ke 2 yang dianggap str adalah strings, parameter ke 3 *optional*index **(values)** supaya bisa melooping expressionnya.

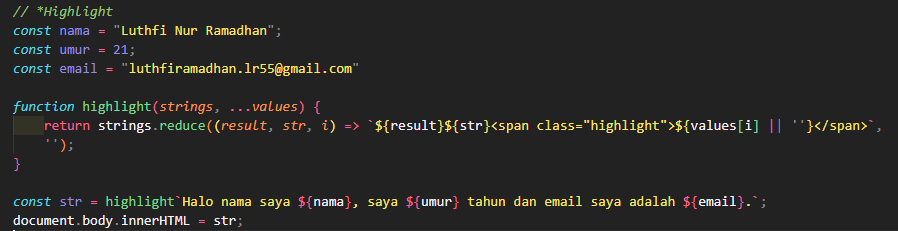
Pertanyaan?

Kenapa kita menggunakan tag templates padahal bisa melakukan stringnya secara langsung dengan console.log().

= Karena kita bisa menampilkan sebuah string hasil pencarian berdasarkan keyword tertentu misalkan ‘nama’.

Kalian ingin menampilkan tampil hasil pencarian dengan ‘nama’ yang kalian masukkan itu ada highlightnya sehingga orang jika melihat akan dimudahkan.

#### Highlight



Penggunaan lain?

* Bisa kita gunakan sebagai **Escaping HTML Tag** (sanitasi html tag, untuk menghindari adanya script atau karakter yang tidak diinginkan).
* **Translation & Internalization** bisa menggunakan library alih bahasa dari website.
* **Styled Components**, melakukan styling pada components biasanya di react.

## DESTRUCTURING ASSIGNMENT

Destructuring Assignment, merupakan sebuah expression (syntax) pada javascript yang membuat kita dapat ‘membongkar’ nilai dari array, atau property object ke dalam variabel yang terpisah.

Contoh Sederhana :

*const coba = [‘satu’, ‘dua’, ‘tiga’];*

*const [a, b, c] = coba;*

*console.log(a); // satu*

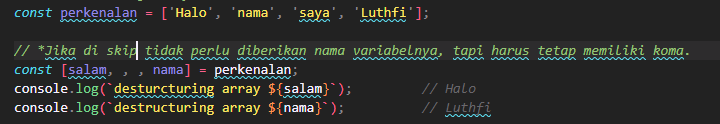
*console.log(b); // dua*

*console.log(c); // tiga*

### Destructuring Array

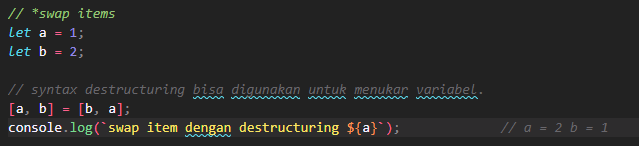
Contoh menggunakan Destructuring Array :

\*Kita bisa memetakkan isi array ke variabel baru namun jika tidak ingin melakukannya (skip) kita bisa mengosongkannya namun tanda koma wajib ada.



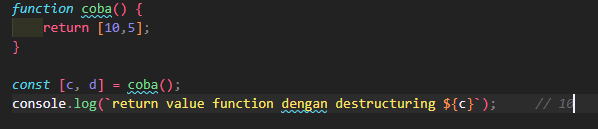
#### Swap Items

Kita bisa juga menukarkan isi array dengan menggunakan destructuring.



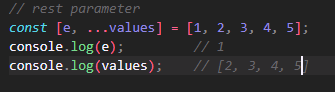
#### Return Value Function

Mamasukkan return value function ke dalam Destructuring



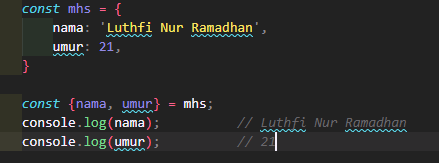
#### Rest Parameter

Jika kita memiliki banyak array, namun kita tidak mengetahui berapa variabel baru yang kita perlukan, kita bisa membungkusnya ke dalam variabel dengan menggunakan rest parameter yaitu **…(nama parameter).** Dengan menggunakan rest parameter nilai yang dimasukkan ke dalamnya akan berupa array baru.

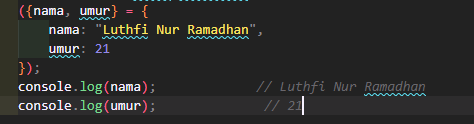


### Destructuring Object

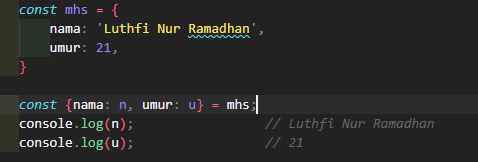
Selain kita bisa melakukan destructuring object pada array kita juga bisa melakukannya pada Object. Namun memiliki syarat *variabel baru yang kita buat harus sama dengan nama property pada objectnya.*



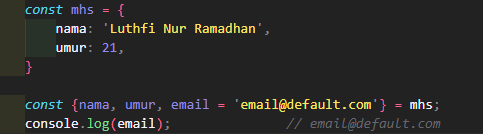
#### Assignment Tanpa Deklarasi Object



#### Assignment Ke Variabel Baru



#### Default Value

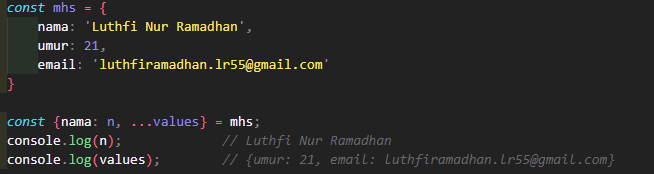


\*Kita juga bisa melakukan assignment ke variabel baru sekaligus memberikan default value seperti berikut

*email: e = 'email@default.com'*

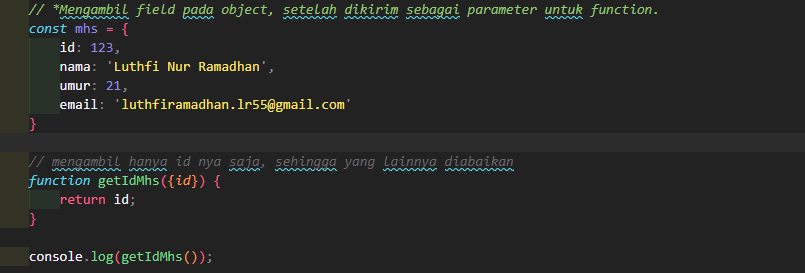
*console.log(e); //* [*email@default.com*](mailto:email@default.com)

#### Rest Parameter



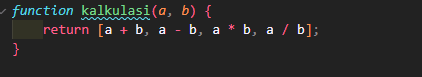
#### Parameter untuk Function

Kita bisa memanggil property tertentu saja pada object ke dalam parameter dari function, di dalamnya kita cukup panggil *{propertyObject}*



### Destructuring Function Values

Kita bisa memetakkan return value dari function ke dalam destructuring.



Kita bisa melakukannya dengan berbagai cara :

*Cara Manual*



*Cara 1 – skip bagian yang tidak digunakan*

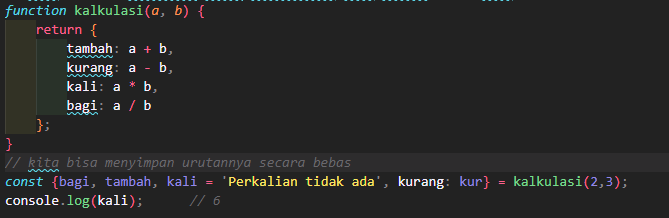


*Cara 2 – default value*

*\*dengan menuliskan semuanya ke variabel baru, karena ini melakukan operasi matematik nama variabel harus terurut seusai.*

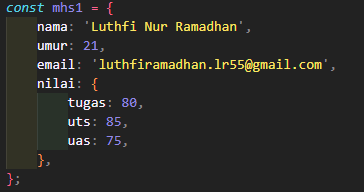


*Cara 3 – memetakkan ke dalam object*



### Destructuring Function Arguments

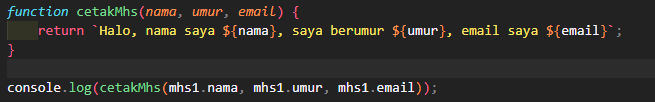
Kita bisa memetakkan destructuring ke dalam function argument.



Kita bisa melakukannya dengan berbagai cara :

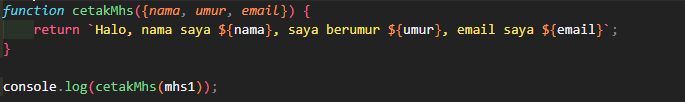
*Cara Manual*

*\*dengan cara manual ini kita bisa memecahnya dan memanggilnya saat pemanggilan function atau di dalam return kita panggil 1 per 1.*



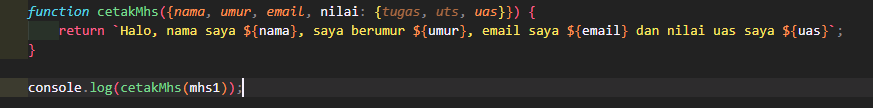
*Cara 1 – destructuring arguments*

*\*dengan cara destructuring argument kita hanya cukup memanggil objectnya dan di dalam parameter kita gunakan {} karena kita sedang memanggil object dan disini kita sudah bisa langsung memanggilnya.*



*Cara 2 – cara ini dilakukan ketika object compleks*

*\*seperti dengan cara sebelumnya kita bisa memanggil property objectnya, namun jika ada object di dalam object kita bungkus lagi seperti berikut* ***nilai: {tugas, uts, uas}***



## For … of vs. For … in

### For … of

Adalah sebuah looping yang bisa mengulang atau menelusuri object object yang iterable.

**Object Iterable** adalah jenis object di Javascript yang memungkinkan untuk mengakses element element di dalamnya. Berikut adalah Object Iterable yang dimaksud :

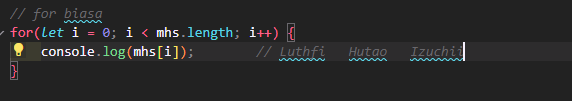
* String
* Array
* Arguments / NodeList
* Typed Array
* Map
* Set
* User-Defined Iterables

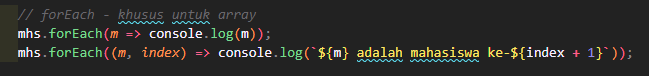
#### Looping Array

Berikut adalah contoh penerapannya pada perulangan array



Looping dengan cara biasa

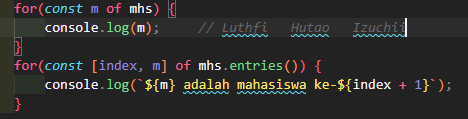


Cara Looping dengan menggunakan forEach 

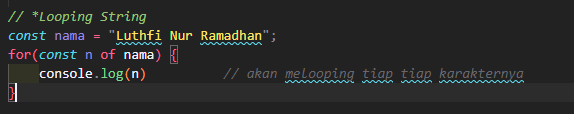
Dengan Cara For … of

\**kita harus merepresentasikan-nya menjadi variabel, dalam contoh kita representasikan menjadi m.*

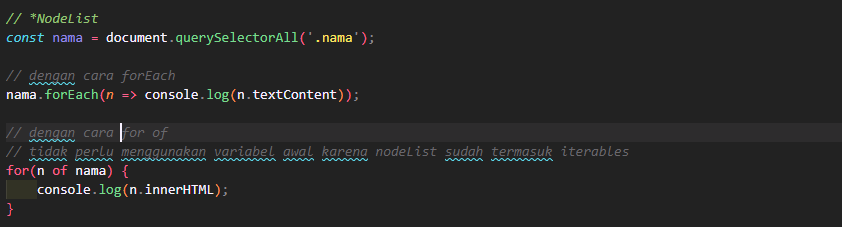
*Secara default tidak memiliki index.*



#### Looping String



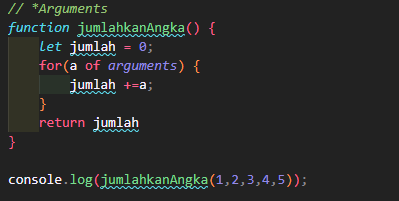
#### Looping NodeList



#### Looping Arguments

\**jika menggunakan reduce dan forEach tidak akan bisa karena tidak berbentuk array tetapi berbentuk arguments.*

*\*jika tidak diberikan argumen maka akan ditangkap oleh arguments.*



### For … in

Hanya untuk **enurable** (property pada object. Object bisa di looping isinya dengan menggunakan for … in.

