**ADVENCED JAVASCRIPT PROGRAMMING LANGUAGE**

**JS**

Name: Muhammad Mu’izzuddin

Book: javascript

**Javascript Programming Language**

**1.Javascript lanjutan:  mengulang object**

dimateri ini kita akan mengulang atau mengingat ingat dulu materi materi

sebelumnya, kali ini kita akan coba membuat object dengan berbagai cara

kalo kalian masih inget, dimateri sebelumnya kita sudah pernah membuat

object dengan 3 cara yaitu  menggunakan

1.object literal

2.function declaration

3.constructor

nah sebenarnya ada cara lain yaitu menggunakan object.create yang belum

pernah kita coba misalnya saya mau buat object yang isinya nama lalu didalamnya ada energi lalu saya buat mehtod untuk menambah energinya dan masing masing object saya buat dengan cara yang berbeda beda

1. object literal

Yang pertama membuat object dengan object literal contoh

// let identitas = {

//     nama : 'muiz',

//     energi : 10,

//     tambahEnergi : function(energi){

//         this.energi += energi

//         return `nama ${this.nama} jurusan  menambahkan energi sebanyak

${energi}`

//     }

// }

nah disini ada yang baru yaitu saya menggunakan ${} tanda dolar lalu kurung

kurawal buka dan tutupnah didalam javascript untuk menulis string itu bisa

menggunakan kutip satu kutip dua atau ada yang baru kalian bisa pakai backtik(kutip yang berada sebelah tombol angka 1 `` ) ini namanya template string

nah dengan menggunakan string literal kalian bisa menulis seperti contoh

diatas  atau lebih jelasnya seperti ini

// let nama = 'muiz'

// let umur = 16

// console.log(`halo nama saya ${nama} umur saya ${umur}`);

jadi bisa langsung dimasukkan nilainya tanpa menggunakan operator concate

kalian tidak perlu keluar dari stringnya lalu ditambah atau menggunakan tanda + untuk menggabungkan lebih dari satu string kalian cukup tulis tanda dolar

lalu kurung kurawal buka dan kurung kurawal tutup ${} lalu didalam kurung kurawalnya  kalian kasih nilai yang mau digabungkan

2.function declaration

     yang kedua membuat object dengan function declaration contoh

// function identitas(nama, energi){

//     let i = {}

//     i.nama = nama;

//     i.energi = energi;

//     i.tambahEnergi = function(En){

//         this.energi += En

//         return `nama ${this.nama} menambahkan energi sebanyak ${En}`

//     }

//     return i

// }

// let muiz = identitas('muiz', 10);

3.constructor function (keyword new)

     yang ketiga membuat object dengan constructor function contoh

// function Identitas(nama, energi){

//     this.nama = nama

//     this.energi = energi

//     this.tambahEnergi = function(En){

//         this.energi += En

//         return `nama ${this.nama} menambahkan energi sebanyak ${En}`

//     }

// }

// let husain = new Identitas('husain', 20);

ketiga cara ini harus kalian pahami dulu sebelum melanjutnya kemateri yang

lebih dalam lagi nah dimateri selanjutnya kita akan bahas mengenai

Object.creater() ini

begitu contoh contoh bagaimana kita membuat object dengan berbagai cara

mudah mudahan kalian tambah inget lagi ya

sampai disini materi kali ini

dan satu lagi jangan lupa titik koma ;

**2.javascript lanjutan: Object.create()**

kali ini kita masuk mengenai konsep Object.create()

nah ini adalah salah satu cara untuk membuat object tapi sebelum kita masuk kesini coba kita review dulu beberapa cara untuk membuat object

nah sekarang kita akan lihat dulu menggunakan cara cara tersebut apa

bedanya, apakah ada kekurangan dan sebaiknya kita pakai yang mana

sebelum kita masuk kedalam Object.create() ini

1.menggunakan object literal

// let identitas = {

//     nama : 'muiz',

//     energi : 10,

//     makan : function(energi){

//         this.energi += energi

//         return `nama ${this.nama} menambahkan energi sebanyak ${energi}`

//     }

// }

yang pertama menggunakan object literal ini dari materi sebelumnya ya

nah dengan menggunakan object literal ini adalah yang paling simple ya

kalo kalian bikin objectnya cuma satu dan tidak terlalu besar objectnya ya

kalian bisa gunakan object literal

tapi ketika objectnya banyak gimana kalo 10 20 dan seterusnya itu akan

memakan banyak resource karena kalian melakukan sesuatu yang sama itu

berulang ulang itu kelemahan atau problem yang pertama

PROBLEM: tidak efektif untuk object yang banyak

nah sekarang kita lihat kalo menggunakan function decklaration

2.function declaration

sepertinya kalo menggunakan function declaration ini kita dapat mengatasi

masalah tadi karena kalian cuma bikin objectnya sekali

nanti kalian bisa isi valuenya berkali kali

// function Mahasiswa(nama, energi){

//     let mahasiswa = {}

//     mahasiswa.nama = nama

//     mahasiswa.energi = energi

//     mahasiswa.makan = function(energi){

//         return this.energi += energi

//     }

//     return mahasiswa

// }

// let muiz = Mahasiswa('muiz', 10)

nah dengan menggunakan cara ini pun kalo kita melakukan instantsiasi pada

objectnya itu ternyata dibelakang layar method ini  akan tetap dibuat

mau dipakai mau tidak tetap dibuat dan disimpan kedalam memory

nah jadi sama saja dengan menggunakan object literal mehtodnya tetap akan

dibuat berulang ulang kali pada masing masing object jadi dengan

menggunakan function declaration masih ada duplikasi, ada solusinya kalo

mau lebih efektif

nah caranya adalah dengan membuat object terpisah objectnya berisi method   
ini (makan) nah nanti objectnya kita tinggal panggil saja didalam object

mahasiswa jadi kita buat dulu objectnya

// const methodEnergi = {

//     makan : function(energi){

//         return this.energi += energi

//     }

// }

// function Mahasiswa(nama, energi){

//     let mahasiswa = {}

//     mahasiswa.nama = nama

//     mahasiswa.energi = energi

//     mahasiswa.makan = methodEnergi.makan

//     return mahasiswa

// }

// let muiz = Mahasiswa('muiz', 10)

nah gini ini kalo kita jalanin harusnya jalan

sekarang kalo saya tulis muiz tidak ada masalah nah sekarang bedanya

adalah gini setiap saya menjalankan program ini object methodEnergi

akan disimpan satu kali kedalam memory lalu ketika saya memanggil function mahasiswa dan mengisi methodnya

maka dia hanya mengacu kepada object yang sudah kita buat tadi, tidak dibuatulang kalo sebelumnya object literal setiap instantsiasi (pembuatan objectnya)

methodnya dibikin ulang

jadi ini lebih efektif lagi walaupun masih ada problem, problemnya apa?

kalo misalkan saya bikin sebuah method baru misalkan namanya tidur

// const methodEnergi = {

//     makan : function(energi){

//         console.log('selamat makan')

//         return this.energi += energi

//     },

//     tidur : function(jam){

//         console.log('selamat tidur')

//         return this.energi += jam \* 2

//     }

// }

// function Mahasiswa(nama, energi){

//     let mahasiswa = {}

//     mahasiswa.nama = nama

//     mahasiswa.energi = energi

//     mahasiswa.makan = methodEnergi.makan

//     mahasiswa.tidur = methodEnergi.tidur

//     return mahasiswa

// }

// let muiz = Mahasiswa('muiz', 10)

nah ceritanya ada method baru didalam objectnya, begitu ada method baru

kalian harus daftarin didalam function mahasiswanya artinya kalian mengelola

dua object sekarang, object yang dibikin pakai function declaration sama

object yang dibikin literal yang berisi function function

secara menegement memory ini efektif tapi kalian harus mengelola ketika buat function didalam methodEnergi

kalian harus daftarin kedalam mahasiswa nah alangkah baiknya kalo ada cara

otomatisnya jadi kalian tidak perlu nih daftarin kedalam mahsiswa

kalian boleh hapus

nah terus kalo dihapus bagaimana dong caranya function declaration ini tahu

bahwa 2 function yang ada didalam object lain ini punyanya object

mahasiswa? nah caranya menggunakan object.create tadi

// const methodEnergi = {

//     makan : function(energi){

//         console.log('selamat makan')

//         return this.energi += energi

//     },

//     tidur : function(jam){

//         console.log('selamat tidur')

//         return this.energi += jam \* 2

//     }

// }

// function Mahasiswa(nama, energi){

//      nah waktu kita daftarkan objectnya kita ganti nih tanda {} menjadi

//      Object.create() sama saja, sama sama membuat object

//      nah tapi bedanya disini

//     let mahasiswa = Object.create(methodEnergi)

//       jadi kalian bisa ngasih tahu bikin object baru sambil juga ikut membawa

property dan method dari object lain ketika dibutuhkan dalam hal ini

//       objectnya adalah methodEnergi

//     mahasiswa.nama = nama

//     mahasiswa.energi = energi

//     return mahasiswa

// }

// let muiz = Mahasiswa('muiz', 10);

jadi mahasiswa dibikin sambil membawa properti dan method dari

methodEnergi ini kalo misalkan didalam Object Oriented kalian mengenal

istilah pewarisan nah ini mirip lah dengan itu

nah coba kalian lihat

sekarang saya tidak perlu mendefinisikan method makan dan tidur karena

sudah ada didalam object.create kita coba ya

saya tulis muiz aman ya kalo saya tulis muiz.tidur(5)

harusnya sekarang energinya jadi 20 itu kalo pakai Object.create() ya

mudah mudahan sampai disini paham nih

nah tapi sebetulnya kalian tuh butuhnya mahasiswa ini problem berikutnya

kalian tuh sebelumnya hanya butuh mahasiswa saja tidak butuh methodEnergi

meskipun kalian hubungkan disini tetap saja kalian kerjanya 2 kali ya

bikin object satu untuk mendukung object intinya yang mahasiswa

sebetulnya ada caranya supaya kalian tidak perlu bikin object lain supaya

dihubungkan kemahasiswa

yaitu dengan menggunakan prototype dan itu akan kita bahas dimateri

berikutnya

oke jadi mungkin itu dulu materi kali ini kita bahas mengenai object.create

gimana cara penggunaannya ya walaupun sebetulnya saat kita belajar

mengenai prototype itu akan dilakukan secara otomatis

jadi itu saja mudah mudahan kalian paham

kita ketemu lagi dimateri berikutnya

dan seperti biasa jangan lupa titik koma ;

**3.Javascript lanjutan: prototype**

kali ini kita akan masuk kemateri prototype kita akan lihat apa itu prototype

dan bagaimana implementasinya didalam javascript

// const methodEnergi = {

//     makan : function(energi){

//         console.log('selamat makan')

//         return this.energi += energi

//     },

//     tidur : function(jam){

//         console.log('selamat tidur')

//         return this.energi += jam \* 2

//     }

// }

// function Mahasiswa(nama, energi){i

//     let mahasiswa = Object.create(methodEnergi)

//     mahasiswa.nama = nama

//     mahasiswa.energi = energi

//     return mahasiswa

// }

// let muiz = Mahasiswa('muiz', 10);

oke jadi ini adalah code dari materi sebelumnya dimana kita sudah membuat

funtion declaration dimana disini ada function mahasiswa yang didalamnya

kita sudah ditambahkan object.create()

kita tahu dengan menggunakan cara ini kita bisa menghemat penggunaan

memory tapi problemnya adalah kita jadi mengelola dua buah object

padahal yang kita mau fokuskan dimahasiswanya, kita malah harus bikin lagi

methodnya

nah sekarang dengan menggunakan prototype kita bisa bikin ini menjadi lebih efektif lagi caranya begini

sekarang yang akan saya lakukan adalah

mengubah function declaration ini menjadi constructor function

// function Mahasiswa(nama, energi){

//     this.nama = nama;

//     this.energi = energi

// }

kita ganti mahasiswanya menjadi this dan dibelakang layar yang terjadi

sebenarnya adalah javascriptnya membuatkan sebuah variable namanya this

yang bentuknya object

let this = {} dan return this

ya walaupun ini sebenarnya tidak perlu kalian lakukan

sebetulnya dibelakang layar ini yang terjadi

nah tapi sebetulnya tidak hanya membuat sebuah object kosong seperti ini

saja, sebenarnya yang terjadi adalah javascriptnya membuat object.create()

tapi didalam object.createnya

kan kalian inget object.create itu seperti memanggil class yang lain untuk

dijadikan sebagai class parentnya

sebenarnya didalamnya javascript itu memanggil Mahasiswa.prototype

let this = Object.create(Mahasiswa.prototype)

sebetulnya ada sebuah properti yang default dijadikan sebagai parent oleh

javascriptnya parent dari method yang akan kita buat jadi ini ada sebetulnya,

walaupun kalian tidak perlu tulis

jadi secara default constuctor function itu sebetulnya sudah punya parent

namanya prototype sehingga kalo kalian lihat kita tidak perlu membuat sebuahobject baru atau object lain yang dijadikan sebagai parent

nah kita lihat ya

// function Mahasiswa(nama, energi){

//     this.nama = nama;

//     this.energi = energi

// }

// let muiz = new Mahasiswa('muiz', 10)

ini kan tidak ada masalah kalo saya jalankan diconsole aman ya

nah sekarang bagaimana kita mengelola methodnya

kan kalo sebelumnya kita bikin object lain ya

nah sekarang kita cukup memanfaatkan prototypenya tadi

jadi kalo misalkan kita mau bikin method makan kalian cukup tulis begini

Mahasiswa.prototype lalu functionnya

// function Mahasiswa(nama, energi){

//     this.nama = nama;

//     this.energi = energi

// }

// Mahasiswa.prototype.makan = function (porsi){

//     this.energi += porsi

//     return `${this.nama} menambahkan energi sebanyak ${porsi}, selamat

 makan`

// }

// let muiz = new Mahasiswa('muiz', 10)

anggap saja prototype ini adalah sebuah object lain yang berada pada object

yang kita buat dan telah disediakan oleh javascript

seperti yang kita buat dimateri sebelumnya,  ya kan kita bikin object lain untuk

mengelola method ini nah kita tidak perlu bikin object tersebut karena sudah

disediakan oleh javascript, sekarang tinggal kita bikin method didalamnya

nah lihat ini bukan object yang harus kita bikin lagi yang baru

ini sudah ada menempel didalam sebuah object yang baru saja kita buat

paham ya ini kalo saya jalankan

kalo saya tulis muiz kelihatannya hanya ada nama dan energi

lalu method makannya mana? method makannya kalo kita klik, ada didalam

prototype jadi otomatis dimasukkan kedalam object mahasiswanya

sehingga kalo kita panggil muiz.makan(3) langsung terpanggil methodnya

kalo kita lihat energinya nambah jadi 13, keren ya jadi tidak perlu bikin object

baru

jadi kalo saya mau bikin method yang lain tinggal bikin lagi didalam

prototypenya

// function Mahasiswa(nama, energi){

//     this.nama = nama;

//     this.energi = energi

// }

// Mahasiswa.prototype.makan = function (porsi){

//     this.energi += porsi

//     return `${this.nama} menambahkan energi sebanyak ${porsi}, selamat

makan`

// }

// Mahasiswa.prototype.main = function (jam){

//     this.energi -= jam

//     return `${this.nama} mengurangi energi sebanyak ${jam}, selamat

bermain`

// }

// Mahasiswa.prototype.tidur = function(jam){

//     this.energi += jam \* 2

//     return `${this.nama} menambahkan 2 kalilipat energi sebanyak ${jam\*2},

selamat tidur`

// }

// let muiz = new Mahasiswa('muiz', 10);

jadi yang dikelola objectnya cuma satu kita tinggal ambil prototypenya

sehingga kalo ini saya jalankan saya panggil muiz muncul objectnya

lalu kita buka protonya ada 3 ya yang baru kita buat untuk ditempelkan

didalam objectnya

begitu temen temen mudah mudahan sampai disini paham ya

bagaimana kita membuat prototype untuk sebuah object

nah ini kalo kalian lihat mirip dengan sebuah konsep class

ya apa lagi kalo kalian yang sudah belajar mengenai object oriented ya

seperti java atau misalkan php yang oppnya konsep ini mirip dengan konsep

class yang memiliki inheritance atau pewarisan

nah karena itu didalam javascript konsep ini disebut dengan prototypel

inheritance jadi pewarisannya menggunakan prototype gitu ya

nah sekarang pertanyaannya kenapa dijavascript itu tidak pakai class kenapa

kok pakainya prototype? karena memang javascript ini adalah bahasa yang

waktu pembuatannya itu sangat singkat hanya beberapa hari javascript ini

dibuat jadi awalnya itu javascript tidak punya konsep class

saat ini memang sudah ada ya walaupun belum lama konsep class itu

dimasukkan kedalamnya jadi kalo misalkan kalian mau mengubah yang kita

buat ini kedalam class bisa juga

caranya gini, versi class

kalian tulis class lalu nama classnya Mahasiswa

yang didalamnya kalo kalian mau bikin properti itu bikin constructor

// class Mahasiswa {

//     constructor(nama, energi){

//         this.nama = nama

//         this.energi = energi

//     }

//     // kalau mau bikin methodnya kita tulis dibawah constructornya

//     // menulisnya tanpa menggunakan keyword function

//     // jadi langsung nama methodnya lalu kurung buka dan tutup lalu kurung

kurawal

//     makan (porsi){

//         this.energi += porsi

//         return `${this.nama} menambahkan energi sebanyak ${porsi}, selamat

makan`

//     }

//     main (jam){

//         this.energi -= jam

//         return `${this.nama} mengurangi energi sebanyak ${jam}, selamat

bermain`

//     }

//     tidur (jam){

//         this.energi += jam \* 2

//         return `${this.nama} menambahkan 2 kalilipat energi sebanyak ${jam\*2}

, selamat tidur`

//     }

// }

// let muiz = new Mahasiswa('muiz', 10)

// let husain = new Mahasiswa('husain', 20)

jadi ini kita bentuk dalam versi classnya hasilnya sama kalo misalkan kita

jalankan hasilnya akan sama

nah ini dengan menggunakan class kelihatannya lebih rapi lebih mudah dibaca dan kalo kalian yang  sebelumnya sudah belajar opp ini pasti lebih bisa

dipahami ya

nah kenapa kita tidak belajar ini langsung, kenapa pakai yang tadi prototype

ya karena sebetulnya versi class ini dibelakang layarnya itu menjalankan

prototype yang tadi jadi class ini hanya untuk membungkus saja supaya lebih masuk akal jadi sebenarnya dibelakang layar yang dijalankan yang   
prototypenya jadi kalian harus paham dulu konsep yang sebetulnya terjadi

jadi kalo misalkan kalian mau pindah pakai class tidak masalah asal tahu saja

dibelakang layar prototype yang terjadi lalu dengan memahami prototype

kalian jadi bisa paham kalo misalkan kalian menggunakan type data apapun

pada javascript karena kan didalam javascript itu sebetulnya adalah object

karena dia object pasti dia punya proto tadi

sama halnya ketika kalian misalnya punya sebuah array

// let angka = [1,2,3]

 nah ini yang kita tulis sebetulnya dibelakang layar yang terjadi itu begini

// let angka = new Array(1,2,3)

sehingga kalo kita pakai ini

ini kan sudah tahu ini adalah constructor function

array itu adalah object yang pasti didalamnya itu membawa sebuah prototype

jadi kalian anggap saja didalam javascriptnya itu ada constructor function

yang namanya Array

function Array(){

}

sama kan kayak mahasiswa tadi kalo kita panggil let muiz = new Mahasiswa()

pasti dibelakang layar itu ada constructor Mahasiswanya, array juga begitu , kalo kita bikin ini

let angka = new Array()

pasti dibelakang layar harusnya ada ini

function Array(){}

nah karena dia pakai constructor function pasti didalamnya ada ini

function Array(){

    let this = Object.create(Array.prototype)

}

pasti dia terhubung keparent classnya yaitu prototype makanya kalo misalkan kalian pakai array, misalnya begini

// let angka = [1,2,3]

nah tapi tiba tiba kita bisa menggunakan gini

// console.log(angka.reverse())

ini untuk mengembalikkan angkanya, coba kita lihat hasilnya akan menjadi

[3,2,1] tuh terbalik nah ini dari mana reversenya? atau misalkan angkanya

kita acak

// let angka = [3,1,2]

// console.log(angka.sort())

jadi ngurut tuh angkanya, dari mana sih method method ini?

dia merupakan prototype dari object Array gitu ya

kalo saya lihat didalam angkanya buka console klik tanda segitiga

didalamnya pasti ada proto, prototypenya isinya ada banyak tuh

dari mana ini ? ya dari object arraynya

jadi array angka ini mewarisi sebuah method dari object Array

jadi misalkan kalian mau mengecheck tulis saja diconsolenya

Array.prototype

nah nanti akan muncul didalamnya apa saja isinya, kalo object juga bisa

Object.prototype

dia punya mehtod apa saja didalamnya kalian bisa check bahkan kalian

bisa check Number atau String juga, begitu temen temen

jadi mudah mudahan dengan ini kalian paham apa itu prototype dan

bagaimana cara penggunaannya

baik mungkin itu ya untuk materi kali ini kita membahas mengenai prototype

apa itu prototype dan bagaimana implementasinya

jadi sampai disini dulu materinya

kita akan ketemu lagi dimateri berikutnya

dan seperti biasa temen temen jangan lupa titik koma ;

**4. javascript lanjutan: execution context, hoisting dan scope**

kali ini kita akan membahas sebuah topik yang cukup menarik didalam

javascript yaitu closures tapi sebelum kita masuk kesini kita akan bahas dulu

pengantarnya

ada tiga buah topik yang kita sebut dengan exsecution constext, hoisting

dan scope jadi konsep konsep ini merupakan konsep pada javascript yang

menurut saya cukup unik ya atau bahkan kita bisa bilang agak aneh  untuk

sebuah bahasa pemrograman dan ketiga hal ini menurut saya cukup penting

untuk kita pelajari agar kita nanti tidak kebingungan ketika kita masuk ketopik

topik yang lebih lanjut lagi

ya dan nanti ketika kalian menyimak penjelasan ini mungkin rasanya akan

membingungkan tapi tidak apa apa nanti coba sedikit sedikit dipahami saja

dan kalo perlu diulang ulang saja materinya

jadi yang pertama yang harus kalian ingat itu ada 2 fase pada

execution context, exsecution context ini adalah saat progam kalian dijalankan

nanti didalamnya ada yang disebut dengan fase creation dan yang kedua

ada execution, ini yang menurut saya yang membuat javascript menjadi

sebuah bahasa yang unik

coba kita langsung lihat saja study kasus pada progam kita tadi saya sudah

bilang bahwa javascript itu aneh ya, contohnya begini deh

kalo misalkan didalam source javascript kita, kita tulis begini

// console.log(nama)

coba kalian bisa nebak gak kalo saya tulis begini hasil yang ada didalam

consolenya kira kira apa? kalo saya jalankan itu hasilnya eror karena variable

nama tidak pernah kita definisikan sebelumnya tapi coba kalo misalkan

dibawahnya saya tulis var nama

ini kita masih pakai keyword var, tidak apa apa nanti kita akan lihat bedanya

ketika kita pakai keyword pembuatan variable yang lain

contohnya begini

// console.log(nama)

// var nama = 'muiz'

coba sekarang kalo saya jalankan tampilnya apa?, apakah muiz?

kita lihat kalo saya jalankan ternyata tampilnya undefined

padahal variable nama sudah kita definisikan dibawah consolenya

tapi kenapa tampilnya undefined dan tidak eror? sebelum saya jelaskan

alasannya coba saya pindahkan console.lognya dibawah variablenya

// var nama = 'muiz'

// console.log(nama)

kalo gini gimana? sekarang tampilnya baru bener, tampilnya muiz

tapi yang tadi mungkin agak aneh ya ketika console.lognya diatas itu undefined

// console.log(nama)

// var nama = 'muiz'

ini yang terjadi begini

saat progam dijalankan yang terjadi adalah ada sesuatu yang disebut dengan

creation fase atau fase pembentukan ya

fase creation ini terjadi dicontext global karena kita disini langsung bikinnya

didalam file javascriptnya tidak didalam function nanti kalo didalam function

itu beda lagi

A. creation phase pada global context

nah yang terjadi pada phase creation ini  javascriptnya akan mengechek

apakah ada variable atau function didalam codingannya ?

itu dulu yang dia check dia tidak akan pedulikan dulu perintah console.log ini

- ada keyword var didalam codingannya gak?

jadi yang pertama itu

misalkan kalian punya 10 baris code

dari 10 baris itu dicari ada gak keyword var? ada gak keyword function?

itu dulu yang akan dicari, kalo ada dia akan membuat nama variable yang

sama dengan  yang kita buat dan itu akan diisi dengan undefined

oke jadi semua variable akan diset dulu isinya dengan undefined

ya makannya ketika kita tulis console.lognya diatas variablenya sudah ada tapi

isinya undefined, jadi sebenarnya dibalik layar begini

var nama = undefined

console.log(nama)

var nama = 'muiz'

- ada keyowrd function didalam codingannya gak?

nah yang kedua kalo ada function itu akan diisi function itu sendiri

ya jadi diisi code function itu sendiri, nanti kita coba deh

nama function = function()

jadi ini ada dua hal yang dilakukan, konsep ini disebut dengan hoisting

kalo hoisting kalian terjemahkan kedalam bahasa indonesia itu anggap saja

seperti kalian mengerek bendera

jadi benderanya dinaikkan keatas, ini juga sama meskipun ditulisnya dibawah

variable dan function dia akan otomatis dinaikkan keatas jadi ini dulu yang

dilakukan, konsepnya namanya hoisting, jadi itu ya yang terjadi pada

execution context diphase creation

nah tapi sebetulnya selain dua hal ini ada lagi yang didefinisikan

yang didefinisikan adalah javascript mendefinisikan object window sebagai

global object dan javascript mendefinsikan this sebagai window

jadi ini yang dilakukan meskipun kalian tidak menuliskan apapun, jadi ada

dua object yang dibuat dan kalo ada variable diisi dengan undefined kalo ada

function diisi dengan tulisan function itu sendiri dan tidak dijalankan

functionnya

nah ini namanya creation phase, setelah ini baru execution phase

B.execution phase

jadi mengeksekusi progamnya dari atas kebawah

console.log(nama)

var nama = 'muiz'

nah baru dijalanin tuh console.log nama

nah begitu dia jalanin nama isinya apa? didalam memory nama isinya

undefined makanya kalo ini saya jalankan isinya undefined baru kebawahnya

lagi nama diisi dengan muiz, sekarang dimemory namanya muiz tapi dia sudah

terlanjur diconsole.log jadi munculnya undefined, kecuali kalo kita balik

// var nama = 'muiz'

// console.log(nama);

kalo begini aman pertama diisi dengan undefined dulu baru diisi muiz lalu

diconsole.log, begitu ya mudah mudahan sampai disini kalian paham

execution context dan hoisting ini atau kita coba contoh yang lain ya

sekarang saya punya variable nama diisi dengan muiz lalu saya punya variable

umur diisi dengan 18 misalkan

// var nama = 'muiz'

// var umur = 18

nah dibawah ini saya punya function

// function sayHallo (){

//     console.log(`hallo nama saya ${nama}, saya ${umur} tahun`)

// }

kalo begini harusnya sekarang kalian sudah tahu dibelakang layar yang

terjadi adalah creation phase dulu, kita lakukan hoisting untuk dua variable ini

dan kita isi function dengan tulisan functionnnya kalo mau, kita coba kita tulis

saja diatas variable console.log sayhallo

// console.log(sayHallo)

// var nama = 'muiz'

// var umur = 18

// function sayHallo (){

//     console.log(`hallo nama saya ${nama}, saya ${umur} tahun`)

// }

nah sekarang ini kalo kita jalanin tuh ada functionnya isinya adalah yang ditulis

beda dengan kalo saya jalanin

// console.log(sayHallo())

// var nama = 'muiz'

// var umur = 18

// function sayHallo (){

//     console.log(`hallo nama saya ${nama}, saya ${umur} tahun`)

// }

tuh jalan tapi tampilnya begini

“hallo nama saya undefined saya umur undefined tahun”

kenapa? karena tadi hoisting, kita nulis variablenya dibawah console.lognya

kalo saya pindahkan consolenya dibawah variable baru sekarang variablenya

muncul

// var nama = 'muiz'

// var umur = 18

// console.log(sayHallo())

// function sayHallo (){

//     console.log(`hallo nama saya ${nama}, saya ${umur} tahun`)

// }

oke nah ini kenapa masih ada tulisan undefined karena harusnya sebuah

function itu mengembalikan nilai kalo mau tidak ada undefinednya kalian

jangan tulis console.log tapi kalian tulisnya return begini

// var nama = 'muiz'

// var umur = 18

// console.log(sayHallo())

// function sayHallo (){

//     return `hallo nama saya ${nama}, saya ${umur} tahun`

// }

kalo ini saya jalanin tuh baru ilang, oke ya jadi kalo misalkan variable diisi

dengan undefined kalo function dia hanya diisi dengan function itu sendiri

makanya kalo saya tulis sebelum functionnya tidak masalah tapi kalo variable

saya tulis sebelumnya akan bermasalah jadi itu contoh lain dari execution

context dan hoisting

nah kalo misalkan kalian mau lebih detil lagi melihat context executionnya

itu kalian bisa kunjungi sebuah website, kalian google saja javascript visualizer

biasanya saya pakai dari python tutor untuk mengentahui visualisasi

bagaimana javascript mengeksekusi code progam kita

kalo misalkan saya copy yang code diatas itu ya, saya pindahkan kewebsitenya

lalu saya klik visualisasi nah nanti ada visualisasinya, ini kita bisa lihat step by

step bagaimana javascript menjalankan code progam kita

kalo warna hijau panahnya barisnya sudah dieksekusi, kalo warna merah akan

dieksekusi, sekarang kita fokus dibagian functionnya disini

// var nama = 'muiz'

// var umur = 18

// console.log(sayHallo())

// function sayHallo (){

//     return `hallo nama saya ${nama}, saya ${umur} tahun`

// }

tadi kan divisualizernya pada saat masuk menjalankan function itu seolah olah dia menjalankan execution contextnya sendiri, jadi kalo ketemu function

executionnya udah tidak global lagi, kalo ketemu function dia seolah olah

membuat local execution context yang didalamnya sama terdapat juga

creation dan execution phase

nah bedanya kalo local execution context itu selain kita bisa akses window kita juga punya akses ke yang namanya arguments jadi ada object arguments yang

nantinya bisa  berisi apapun yang kita masukkan didalam argumentnya dan

didalamnya ada hoisting juga jadi hoistingnya local dia akan mengecheck

didalam functionnya ada variable gak? kalo ada naikin keatas

atau didalam function ada function lagi gak? nah nanti dia juga melakukan

hoisting kalo ada variable isi dulu dengan undefined pada phase creation

kalo sudah execution baru diisi dengan nilainya kalo misalkan kalian bingung

kita coba pakai kasus

// var nama = 'muiz'

// var username = '@muizzuddin332'

// function cetakUrl (username){

//     var instagramUrl = 'https:instagram.com/';

//     return instagramUrl + username

// }

// console.log(cetakUrl(username))

coba kita lihat dulu hasilnya kelihatannya tidak aneh tapi gimana kalo coba

kita telusuri pakai visualizer tadi

sebelum progamnnya dijalankan creation phasenya sudah dijalankan dulu

dihoisting dulu, nama dan username diisi dengan undefined dan cetakUrl

diisi dengan function kalo kita klik next maka baris satu akan dijalankan

nama diisi dengan muiz next lagi baris dua dijalankan username diisi dengan

@muizzuddin332

functionnya diskip langsung dipanggil nama functionnya, begitu saya jalanin

dia akan memanggil functionnyakan, ini kita akan masuk kedalam local

execution context kalo kita next tuh ada execution context baru

namanya cetakUrl

username diisi dengan @muizzuddin lalu instagramUrl diisi dengan undefined

kalo kita next lagi baru diisi instagramUrl dengan https://instagram.com/

sekarang kereturn next lagi jalanin console.lognya,next lagi tampil

kelayar begitu ya

jadi intinya setiap function dipanggil maka akan membuat execution context

baru yang local tadi dan ditambahkan istilahnya itu execution stage atau

tumpukan execution kalo selesai dijalankan maka execution context

yang localnya akan dihilangkan dari tumpukannya, oke mudah mudahan

kalian paham nih atau biar lebih jelas kita kasih kasus lagi nih

misalkan sekarang saya punya 3 buah function

// function a(){

//     console.log('ini a')

//     function b(){

//         console.log('ini b')

//         function c(){

//             console.log('ini c')

//         }

//         c()

//     }

//     b()

// }

// a()

jadi kalian harus paham ini ada function didalam function karena ini dasar

banget ketika nanti kalian mau paham mengenai closure sekarang kita jalanin

functionnya coba kira kira tanpa kalian jalankan lihat saja codingannya

kira kira tampilnya apa ?

a b c atau c b a ?

coba kalian lihat, tampilnya a b c ya kelihatannya tidak ada masalah

tapi coba kita telusuri menggunakan javascript visualizer

yang pertama kita melakukan hoisting dulu function a diisi dengan semua

yang ada didalam function a baru kita next jalankan function a nya

nah kita akan masuk kedalam function a nya dan sebelum menjalankan

progamnya kita hoisting dulu

hoistingnya mana ? b diisi dengan isi dari function b kalo sudah kita next

baru kita jalankan yang console.lognya tampil kelayar ini a

next kita jalankan yang b kita  masuk kedalam function b kemudian hoisting

dulu c diisi dengan isi dari function c next lagi jalanin console.log nya

lalu next jalanin c kita masuk kedalam function c lalu hoisting lagi ada variable gak? tidak ada, ada function gak? tidak ada, baru jalanin console.lognya

sampai disini apakah selesai? sekarang sampai disini yang selesai baru c

kan kita ada didalam c nih kalo c selesai kita next c akan hilang tuh dari

tumpulan executionnya kita next lagi b selesai b akan hilang dan yang terakhir a selesai

gitu ya jadi ini maksutnya ada tumpukan eksekusi lalu kalo sudah beres hilang dari tumpukan eksekusinya mudah mudahan sampai disini paham ya

nah sekarang kita kembali ke codingan instagram

// var nama = 'muiz'

// var username = '@muizzuddin332'

// function cetakUrl (username){

//     var instagramUrl = 'https:instagram.com/';

//     return instagramUrl + username

// }

// console.log(cetakUrl(username))

nah sekarang ini coba saya ubah progamnya saya tulis sekarang begini

// var nama = 'muiz'

// var username = '@muizzuddin332'

// function cetakUrl (username){

//     var instagramUrl = 'https:instagram.com/';

//     return instagramUrl + username

// }

// console.log(cetakUrl('@husain'));

coba kira kira yang tampil apa? tetep @muizzuddin atau @husain

kalo saya jalanin dia berubah menjadi @husainpadahal username ini diisi

dengan @muizzuddin

ini karena scope begitu kita nulis didalam argumentnya lalu datanya dikirimkankeparameter maka username yang ada didalam function itu ngambilnya yang

parameternya makanya tulisannya menjadi @husain jadi usernamenya dia

mengechek dulu parameter bukan sekarang coba kalo misalkan saya hapus

parameternya

// var nama = 'muiz'

// var username = '@muizzuddin332'

// function cetakUrl (){

//     var instagramUrl = 'https:instagram.com/';

//     return instagramUrl + username

// }

// console.log(cetakUrl('@husain'));

nah tampilnya apa kira kira? tetap @hasan atau balik lagi @muizzuddin ?

nah ini kalo saya jalankan balik lagi jadi @muizzuddin kenapa? ini @husain

dikirimkan keargument menuju parameter tapi diparameternya tidak ada yang menangkap argumentnya nanti dia akan masuk kedalam object arguments

nanti kita lihat deh

nah tapi didalam functionnya ada tulisan username kalo disini kalian menulis

variable yang pertama dia check adalah variable local dulu ada gak?

ternyata tidak ada, kalo gak ada dia check ke parameternya ada gak? tidak ada

juga kalo gak ada difunctionnya dia akan melihat keluar, diglobal ada gak?

oh ternyata diglobal ada makanya dia pakai yang globalnya

kalo misalkan diglobal tidak ada gimana? nah kalo tidak ada dia akan

menampilkan eror

nah tadi pertanyaannya @husainnya kemana? ternyata dia masuk kesebuah

object yang namanya arguments

// var nama = 'muiz'

// var username = '@muizzuddin332'

// function cetakUrl (){

//     console.log(arguments[0])

//     var instagramUrl = 'https:instagram.com/';

//     return instagramUrl + username

// }

// console.log(cetakUrl('@husain'));

kalo saya jalanin tuh ada ya dia ada diindex ke 0, jadi kalo saya tulis

arguments[0] dia tampil tuh, jadi tidak hilang @husainnya jadi bisa

dipergunakan meskipun kalian tidak tangkap diparameternya

mudah mudahan sampai disini paham ya

dan untuk memastikan kalian paham atau tidak saya kasih satu latihan terakhir

function satu(){

    var nama = 'muiz'

    console.log(nama)

}

function dua(){

    console.log(nama)

}

console.log(nama)

var nama = 'husain'

satu()

dua('hasan')

console.log(nama)

tidak usah dijalanin dulu ya terus lihat codingannya kira kira yang akan tampil

dilayar apa? dengan menggunakan konsep yang sudah kita pelajari tadi

execution context hoisting dan scope harusnya kalian sudah bisa paham ini ya

sekarang coba jalanin hasilnya sama gak dengan tebakan kalian? coba kita lihat

kita punya dua buah function dan satu variable pasti yang dilakukan creating

phase hoisting dulu 3 hal ini kalo sudah kalo sudah kita masuk ke exsecution

phase

console.log nama , nama tadi isinya apa? baru masuk ke nama diisi dengan

'muiz'

lalu jalanin satu() dan dihoisting dulu karena ini local execution context

jalanin executionnya dan hilangkan stage executionnya

sekarang function dua() dengan mengirimkan parameter hasan

masuk sini argumentnya tidak ditangkap argumentnya diabaikan

lalu hoisting gak? tidak ya terus console.log nama dicheck dulu ada gak variablenama dilocalnya? tidak lalu cari diargumentnya, tidak ada kemudian dia

keluar cari diglobal scope nah ada ya, lalu hilang dari tumpukan eksekusinya

lalu console.log nama

nama ngambil dari mana? ya jelas dari global ya, begitu ya mudah mudahan

kalian paham

oke ya jadi itu tadi kita sudah pelajari mengenai keunikan atau keanehan dari

javascript yang disebut dengan execution context hoisting dan scope dan ini

akan menjadi dasar kita untuk materi selanjutnya kita akan membahas

mengenai closure

oke ya sampai disini dulu materinya

kita akan ketemu lagi dimateri berikutnya

dan sepeti biasa jangan lupa titik koma ;

**5. javascript lajutan: closure**

kita sekarang akan masuk ketopik yang cukup penting yaitu closure setelah

dimateri sebelumnya kita sudah belajar mengenai  execution context hoisting

dan scope sekarang pemahaman tersebut akan kita terapkan kepada closure

nah sebelumnya kita harus paham dulu mengenai apa itu closure ? definisinya

seperti apa dan konsepnya seperti apa? karena sekali lagi konsep mengenai

closure ini meskipun sangat penting tapi bukan merupakan sesuatu yang

gampang untuk kita pahami

kita lihat ada beberapa definisi dari beberapa sumber ya

1. menurut mozila

*closure merupakan kombinasi antara function dan lingkungan leksikal*

*(lexical scope) didalam function tersebut*

ya nanti lexical scope ini erat kaitannya dengan yang sudah kalian pelajari dimateri sebelum ini kalo misalkan kalian masih bingung mengenai difinisi ini

tidak apa apa nanti lihat saja implementasi langsungnya

2.menurut w3school

*closure adalah sebuah function ketika memiliki akses keparent scopenya*

*meskipun parent scopenya sudah selesai dieksekusi*

nah ini juga sudah kita lihat contoh kasusnya dimateri sebelumnya jadi kita

tahu sebuah function itu bisa memiliki function didalamnya tapi kasus

tersebut bukan merupakan sebuah closure dan akan menjadi closure   
ketika function didalamnya memiliki akses ke parent scopenya

3.menurut code fellow

*closure adalah sebuah function dikembalikan oleh function yang lain yang*

*memiliki akses ke lingkungan saat ia diciptakan*

nah ini balik lagi kelexical scopenya jadi kita bisa bikin function yang

mengembalikan atau mereturn function  jadi bukan function didalam

function tapi function mereturn function

silahkan kalian pahami masing masing dari definisi itu karena masing masing

orang bisa memiliki interpretasi berbeda terhadap closure

dan sekarang kita akan membahas mengenai lexical scope

ya tadi sudah disebut bahwa ada sesuatu yang disebut dengan lexical scope

contoh saya punya code sederhana seperti ini

function init(){

    let nama = 'muiz'

    function tampilNama(){

        console.log(nama)

    }

    tampilNama();

}

init();

coba kalian lihat itu kita punya function diluar namanya init dan didalamnya

ada variable nama dan ada function lagi yang namanya tampilNama

didalam tampilNama menjalankan console.log nama lalu function tampilNama

itu dipanggil

lalu diluar function init function ini juga dipanggil coba dipahami dulu codenya

nah disini kalian sudah tahu kita punya local variable variable yang dibuat

didalam function init dan didalamnya kita punya sebuah function lagi  yang

kita sebut dengan inner function (closure)

nah disini beda dengan bahasa pemrogramman yang lain inner function itu

bisa memiliki akses keparent variablenya jadi ketika kita panggil   
console.log nama kalo kalian inget execution context dia akan mencari dulu

ada gak nama didalam tampilNama ? kalo tidak ada dia mencari keatas

ada gak didalam function initnya ? kalo tidak ada dia terus mencari keatas

sampai ketemu global scope

nah ini yang disebut dengan lexical scope jadi function tampilNama punya   
akses kevariable nama padahal ada diluar functionnya

begitu kasus ini terjadi inner function membutuhkan variable yang ada didalam parent scopenya maka inner function ini disebut dengan closure

oke  supaya lebih jelas lagi kita implementasikan code barusan didalam code

editor kita, kita akan coba jalankan code yang tadi

// function init(){

//     let nama = 'muiz'

//     function tampilNama(){

//         console.log(nama)

//     }

//     tampilNama();

// }

// init();

sebelum kita telusuri coba lihat hasilnya seperti apa jadi ini kalo saya jalankan

akan menampilkan muiz ya kalo kita telusuri execution contextnya didalam functionnya hoisting terjadivariable nama dibuat function tampilNama

dihoisting juga lalu variable nama diisi dengan muiz sekarang ketika kita

jalakan tampilNama console.log dijalankan nah dia membutuhkan data nama

sedangkan didalam local functionnya tidak ada kalo ini terjadi dia akan naik

keatas diparentnya ada gak? kalo misalkan didalam tampilNama saya buat

variable nama yang diisi dengan husain maka yang dipakai didalam

console.lognya adalah yang didalam function tampilNama yaitu husain

dan itu jadinya bukan closure karena tidak menggunakan lexical scopenya

tapi kalo code yang diatas itu adalah closure dan untuk memastikannya   
kita bisa lakukan ini, sekarang saya tidak akan menjalankan tampilNama

tapi saya akan simpang kedalam console.log

// function init(){

//     let nama = 'muiz'

//     function tampilNama(){

//         console.log(nama)

//     }

//     console.log(tampilNama)

// }

// init();

kalo misalkan saya hapus kurung buka dan tutupnya maka functionnya tidak dieksekusi jadi harusnya yang tampil hanya isinya saja tapi kalo misalkan saya

tulis console.dir

// function init(){

//     let nama = 'muiz'

//     function tampilNama(){

//         console.log(nama)

//     }

//     console.dir(tampilNama)

// }

// init();

dia akan menampilkan objectnya coba kalian jalankan, kita lihat dalamnya

kalo saya klik dibawah kita bisa menemukan ada scopes, kalo saya klik

katanya ada closure jadi function init ini didalamnya ada closure, kenapa ada

closure? karena membutuhkan variable nama dari parentnya jadi muncul

closure kalo misalkan saya punya variable lain

// function init(){

//     let nama = 'muiz'

//     let umur = 18

//     function tampilNama(){

//         console.log(nama)

//     }

//     console.dir(tampilNama)

// }

// init();

kalo saya jalankan umur ini tidak akan masuk kedalam closurenya kalian lihat discopesnya tidak ada umur, tapi ketika saya butuhkan didalam function   
tampilNama

// function init(){

//     let nama = 'muiz'

//     let umur = 18

//     function tampilNama(){

//         console.log(nama)

//         console.log(umur)

//     }

//     console.dir(tampilNama)

// }

// init();

sekarang function tampilNama butuh dua data dari luar makanya umur akan

masuk keclosure juga, lihat discopesnya, karena membutuhkan data dari luar

maka sekarang ada dua variable yang masuk kedalam closure scope gitu

mungkin sekarang sudah dapat gambaran kenapa tadi disebutnya gabungan antara function dengan lexical scopenya dan sekarang kalo codenya saya

kembalikan

// function init(){

//     let nama = 'muiz'

//     function tampilNama(){

//         console.log(nama)

//     }

//     tampilNama()

// }

// init();

nah pada kasus ini function tampilNama ya sebagai inner functionnya inikan

langsung dijalankan nah ini menarik nih kita bisa lakukan ini

// function init(){

//     let nama = 'muiz'

//     function tampilNama(){

//         console.log(nama)

//     }

//     return tampilNama

// }

// init()

kalo saya return tampilNama tapi tidak saya jalankan jadi saya mengembalikan function tampilNama tanpa mengembalikannya, kalo saya jalankan

sekarang tidak tampil apa apa kenpa ? karena lihat ini

ini kan saya panggil function init() dibawah, function initnya dijalanin terus   
didalamnya kita punya function tampilNama terus kita kembalikan tanpa

dijalankan, nah sampai disini istilahnya functionnya baru jalan sebagian lalu gimana supaya tampil namanya jalan?kita bisa simpan kedalam variable

// let panggilNama = init()

jadi sekarang funciton init yang sudah dijalankan sebagian masuk

kepanggilnama, baru dibawahnya kita bisa jalankan panggilnama

// panggilNama()

ya untuk menjalankan tampilNama ini kalo saya jalankan baru tampil tuh

oke ya mungkin ini agak aneh tapi ini memungkinkan kita bisa melakukan yang nanti kita sebut dengan function factories contohnya gini misalkan saya ingin

namanya itu tidak muiz tapi nanti tergantung user masukinnya apa gitu ya

// function init(){

//     function tampilNama(nama){

//         console.log(nama)

//     }

//     return tampilNama

// }

// init()

kita kasih parameter ke function tampilNama jadi dengan logika yang tadi itu

kan function initnya jalan lalu mengembalikan function tampilNama tapi belumdijalankan jadi sekarang kalo saya mau jalankan function panggilNama saya

butuh parameter

// let panggilNama = init()

// panggilNama('muiz')

ini kalo saya jalankan muncul muiz  jadi ini salah satu penggunaan dari closure kalo kalian lihat codenya sebetulnya kita mengembalikan function tampilNama dan ini bahkan bisa kita ringkas lagi

// function init(){

//     return function(nama){

//         console.log(nama)

//     }

// }

// init()

// let panggilNama = init()

// panggilNama('muiz')

// panggilNama('husain')

tidak perlu pakai nama function kita bisa langsung return functionnya

anonymous function atau function tanpa nama seperti itu, ini sama saja kalo

saya jalankan hasilnya sama jadi itupun salah satu contoh dari closure

nah mungkin sekarang pertanyaannya kenapa kita menggunakan closure?

ya tadi kita sudah tahu cara bikinnya sekarang alasan menggunakan closure

ada beberapa ya

1.untuk membuat function factories

    ya seperti tadi contohnya

2.untuk membuat private method

nanti kita lihat nih masing masing dari contohnya ya

1. untuk membuat function factories

ini lihat misalkan saya punya contoh kasus mirip seperti tadi bedanya function outernya atau function luarnya punya parameter ya

// function ucapkanSalam(waktu){

//     return function(nama){

//         console.log(`Hallo ${nama}, selamat ${waktu} semoga harimu

menyanangkan`)

//     }

// }

jadi ada parameter waktu yang nanti untuk memberitahu pagi siang atau

malam misalnya, nah function ini mengembalikan function lagi inner

functionnya untuk ngasih tahu namanya siapa misalkan dengan menggunakan   
logika yang tadi kita bisa membuat pesannya itu ditampilkan sesuai waktu

misalkan saya punya variable seperti ini

// let selamatPagi = ucapkanSalam('pagi')

// let selamatSiang = ucapkanSalam('siang')

// let selamatMalam = ucapkanSalam('malam')

nah sekarang sudah selesai belum? belum ya kalo saya mau jalankan saya

tinggal tulis begini

// selamatPagi('muiz')

kalo saya jalankan tampil ya selamat pagi sekarang kalo saya tulis

selamatMalam

// selamatMalam('husain')

maka akan muncul selamat malam kalo misalkan saya coba lihat menggunakanconsole.dir isi dari selamatMalam

// console.dir(selamatMalam)

nah nanti kita bisa lihat bahwa inner function ini sudah jalan setengahnya cobakita jalankan pasti didalam scopenya ada closure waktu: malam, berarti, ini

closure dengan menggunakan lexical scopenya waktu yang isinya malam

jadi meskipun selamat malam ini belum kita jalankan ucapkan salamnya

sudah berjalan dengan mengisikan data waktu berisi malam jadi sudah jalan

setangahnya tinggal nunggu data dari namanya

// console.dir(selamatMalam('muiz'))

kalo ini saya jalanin baru ada gitu ya ini disebut dengan factories function

kita bikin function sesuai dengan function yang lain mudah mudahan paham ya

dan tadi alasan yang lain adalah agar kita seolah oleh punya privat method

atau privat variable

2.privat method

saya kasih contoh dulu ini contoh classik misalnya saya punya sebuah variable namanya counter

// let counter = 0

jadi ceritannya saya mau bikin sebuah progam sederhana untuk menghitung

sudah berapa kali sebuah tombol itu diklik lalu saya bikin function

// let add =  function(){

//     return ++counter

//     // jadi ditambah satu dulu baru dicetak

// }

lalu saya console

// console.log(add())

kalo saya jalankan harusnya counter yang tadinya 0 jadi 1 kalo saya duplicat

harusnya 1 2 3

// console.log(add())

// console.log(add())

sip ini tidak ada masalah dan ini bukan closure kelihatannya ini tidak ada

masalah tapi coba bayangkan kalo progam kalian sudah panjang puluhan

bahkan ratusan baris dan ada salah satu baris yang mengubah counter kalian

mengisi ulang sicounternya

// let counter = 0

// let add =  function(){

//     return ++counter

// }

// counter = 10

// console.log(add())

// console.log(add())

// console.log(add())

nah sekarang counternya jadi 10 dan karena hoisting counter yang ada didalamfunction jadi terpengaruh, nah solusinya gimana supaya counter ini tidak

terganggu? sebenarnya kita bisa simpan kedalam

// let add =  function(){

 let counter = 0

//     return ++counter

// }

// counter = 10

// console.log(add())

// console.log(add())

// console.log(add())

cuma sekarang kalo saya jalankan hasilnya jadi satu semua karena setiap

manggil add counter diisi dengan 0 lalu tambah 1, berarti salah ya

tujuannya pengen tidak terganggu tercapai tapi masalahnya selalu diset

kembali ke 0, sekarang gimana kalo kita pakai closure

// let add = function (){

//     let counter = 0

//     return function(){

//       return ++counter

//     }

// }

// counter = 10

// console.log(add())

// console.log(add())

// console.log(add())

kalo kita jalanin tampilnya apa ? tampilnya sebuah function karena functionnyayang jalan baru add inner functionnya belum jalan solusinya gimana ?kita

harus simpan didalam sebuah variable

// let add = function (){

//     let counter = 0

//     return function(){

//       return ++counter

//     }

// }

// let a = add()

// counter = 10

// console.log(a())

// console.log(a())

// console.log(a())

dan setiap kita jalanin inner function counter akan selalu mengacu ke nilai

counter sebelumnya coba kita jalankan, hasilnya baru bener 1 2 3

begitu jadi seolah olah counter ini jadi privat karena tidak bisa diakses dari   
luar tapi nilainya tetap dipertahankan karena dia menjadi closure

begitu mudah mudahan paham nih tapi ini kalo temen temen lihat agak

aneh ya kita bikin sebuah variable buat nampung

nah sebenarnya ada caranya supaya otomatis menjalankan si function add nya tanpa harus memindahkan didalam variable dan tetap pakai add

let add = (function (){

    let counter = 0

    return function(){

      return ++counter

    }

})()

counter = 10

console.log(add())

console.log(add())

console.log(add())

nah caranya supaya dijalankan secara penuh, ini bisa kita bungkus kedalam

sebuah imideately invoked function jadi kalian kasih kurung lalu function ini

kita jalankan menggunakan kurung juga

jadi ditutup dengan kurung lalu kita kasih kurung lagi diluarnya jadi pada saat diassignment divariable add langsung menjalankan inner functionnya jadi

sekarang kalo saya jalanin normal ya dan kalo saya kasih counter yang diisi

dengan 10 tidak akan mengganggu functionnya

gitu temen temen ya jadi ini salah satu contoh dari closure juga mudah

mudahan sampai disini kalian paham, baik jadi itu ya penjelasan mengenai

closure

sekali lagi seperti yang saya bilang diawal topik ini bukan merupakan topik

yang mudah untuk dipahami tapi sangat penting jadi saran saya kalo masih

bingung diulang lagi materinya dicoba sendiri semua study kasus yang sudah

kita coba

silahkan diubah ubah parameternya silahkan buat contoh kasus yang lain

supaya kalian makin paham lagi

ya jadi mungkin itu saja untuk materi kali ini

kita akan ketemu lagi dimateri berikutnya

dan seperti biasa jangan lupa titik koma ;

**6.Javascript lanjutan: var let dan const**

dimateri kali ini kita akan membahas tiga hal ini

- var

- let

- const

kalian mulai sekarang jangan pernah menggunakan var pada saat

mendeklarasikan variable ganti var dengan let dan sekarang kita akan

pahami kenapa kita tidak menggunakan var saat membuat variable dijavascript

jadi sekali lagi ada tiga cara untuk membuat variable didalam javascript

dan harusnya kalian sudah paham mengenai apa itu var dan cara

penggunaanya bagaimana karena syarat untuk mengikuti seri ini kalian harus

paham mengenai keyword var tapi sakarang kita coba dulu untuk yang

keyowrd varnya

nah misalkan saya punya variable yang namanya i yang diisi dengan 10

// var i = 10;

// console.log(i);

kalo misalkan saya console log harusnya didalam console akan muncul isi dari

variable I ini tidak aneh sekarang bagaimana kalo kita hilangkan keyowrd

varnya kira kira bisa gak? coba ya

// i = 10;

// console.log(i);

kita lihat hasilnya, masih bisa ya jadi yang terjadi javascriptnya seolah olah

menambahkan keyowrd var didepan i nya sekarang kita lihat kalo saya

simpan console lognya diatas variablenya kita panggil dulu baru kita

deklarasikan variablenya kira kira eror gak?

// console.log(i);

// var i = 10;

kira kira hasilnya apa? coba kita jalankan tampilnya undefined

nah kalian mesti paham undefined adalah sebuah nilai yang ada didalam

javascript

nah ini kenapa kira kira ada nilai undefined padahal kita tidak mengisikan nilai

tersebut kedalam variable i? jadi didalam javascript ada istilah yang

dinamaka dengan hoisting

kita pahami dulu apa terjemahan dari hoisting anggap saja hoisting itu seperti

orang yang mengerek bendera keatas itu namanya benderanya dihoisting

atau dinaikkan dulu keatas

nah sama seperti javascaript kalo kita melakukan hal tadi sebenarnya yang

terjadi itu begini sebelum console lognya itu dibuat dulu var i begini

// var i = undefined

// console.log(i);

 lalu setelah console.lognya jadi seperti ini

// i = 10;

jadi variblenya dinaikkan dulu keatas dan dia diisi dengan nilai undefined gitu

makanya begitu kita menuliskannya begini maka nilainya undefined karena

tadi nilainya didefinisikan dulu dengan undefined nah ini namanya hoisting

untuk variable

sebetulnya konsep hoisting ini berlaku terhadap function dan yang lainnya ya

intinya untuk variable itu perilakunya seperti ini jadi dia akan dinaikkkan dulu meskipun kalian menuliskannya seperti ini

// console.log(i);

// var i = 10

atau pada saat kalian menuliskannya begini

// var I = undefined

// var j = undefined

// console.log(i);

// i = 10, j = 'muiz'

paham ya kenapa nilainya undefined padahal harusnya itu eror kalo kalian

pernah menggunakan bahasa pemrograman yang lain seperti c atau java

kalian melakukan hal ini pasti hasilnya eror karena kita tidak mungkin

memanggil sebuah variable sebelum mendefinisikannya kalo dijavascript masihboleh tapi tadi perilakunya agak aneh lalu sekarang gimana cara

mengatasinya ? ganti saja varnya menjadi let seperti ini

// console.log(i)

// let i = 10;

maka ini akan eror, nah ini sama seperti bahasa pemrogaman yang lainnya

itu jadi salah satu alasan kenapa menggunakan let atau const  ini juga sama

sekarang perilakuknya juga sama seperti bahasa bahasa pemrogaman yang lain karena seharusnya sebelum kita memanggil sesuatu harus dibuat dulu

variablenya

// const i = 10;

// console.log(i);

jadi kalo ini saya simpan diatas kita ganti const baru tidak ada masalah

paham ya sekarang kita kembali lagi menggunkan var untuk alasan yang

selanjutnya contohnya gini saya menggunakan looping

// for(var i = 0; i < 10; i++){

//     console.log(i)

// }

 coba kalo ini kira kira tampilnya apa ? pengulangannya akan dilakukan

sebanyak 10 kali begitu i nya sampai 10 maka dia akan keluar dari

pengulangannya jadi harusnya tampil angka 0 – 9, tidak ada yang aneh

kalo saya tambahkan console log i kira kira tampilnya apa

// console.log(i);

tuh hasilnya angka 10 karena i nya diincrement untuk yang terakhir kalinya

oke harunya disini tidak ada masalah, tapi sekarang kalian lihat didalam

javascript itu memungkinkan kita mengakses sebuah variable didalam

loopingnya

nah kalo didalam bahasa pemrogaman yang lain ini juga tidak bisa karena

bahasa pemrogaman yang lain menganut konsep yang dinamakan dengan

block scope (lingkup pada block progam)

block progam  itu  kalo kalian ketemu dengan kurung kurawal buka dan tutup

itu disebutnya block kalo dibahasa pemrograman yang lain ini eror karena

tidak bisa mengakses i yang ada didalam block

javascript tidak menganut ini, karena javascript menganut function scope

nah ini harusnya kalian sudah pernah pelajari dimateri sebelumnya memang

harusnya ini eror karena kita tidak bisa mengakses isi variable yang ada

didalam block seperti ini

nah gimana cara mengatasinya? supaya variable i ini tidak bisa kita akses kita

harus ubah scopenya menjadi function scope misalnya saya simpan kedalam

sebuah function

// function tes(){

//     for(var i = 0; i < 10; i++){

//         console.log(i)

//     }

// }

jadi sekarang fornya ada didalam function, sebelum kita console log i nya kita

panggil dulu functionnya functionnya tidak akan bisa jalan kalo tidak kita

panggil namanya

// tes();

tapi sekarang kalo saya tulis console log i

// console.log(i);

kira kira kita bisa akses variable i gak? gak bisa ya ini kalo saya jalankan akan

eror ini lah yang disebut dengan function scope makanya para programmer

javascript itu ngakalin gimana agar supa variablenya itu berperilaku sama

seperti Bahasa pemrograman yang lain tapi menggunakan function scope

nah biasanya kita menggunakan sesuatu yang disebut dengan ini

IIFE atau SIAF ini sama saja ya

IIFE (Imediately Invoked Function Exspression) jadi function exspression yang

dipanggil langsung ketika dibuat sama seperti SIAF (Self Invoking Anonymous Function)

itu sama saja, ini kita buat supaya kita menjaga agar variable yang ada didalam

functionnya itu tidak bisa diakses dari luar kita coba ya caranya begini

kan tadi SIAF itu adalah self invoking anonymous function

anonymous function itu artinya tidak bernama kita bisa bikin function tidak

bernama didalam javascript menggunakan apa? menggunakan function

exspression, function exspression itu gini

var x = function(){

}

nah ini function exspression tapi kita juga bisa bikin si function ini tidak perlu

kita simpan kedalan variable tapi kita langsung jalanin caranya kita bikin

kurung dulu

tadi kan katanya kurung itu untuk menjalankan functionnya kan lalu

didalamnya kita bikin function dan diakhir kurung kurawal tutup kita

tambahkan kurung buka dan tutupnya begini jadinya

// (function(){

// }());

ini adalah self invoking anonymous function atau imidiately invoked function

exspression sekarang kalo saya tulis didalamnya for tadi

// (function(){

//     for(var i = 0; i < 10; i++){

//         console.log(i);

//     }

// })()

kalo kita jalankan itu tidak ada masalah ya tapi ketika kita tulis console log i

diluar functionnya

// console.log(i);

nah maka dia akan eror ini memang yang kita harapkan

nanti akan kebayang ketika kalian bikin progam javascriptnya besar atau

nanti kalian juga berkolaborasi itu memungkinkan kalian punya nama

variable yang sama tapi fungsinya beda atau misalnya kalian script

punya orang lain library yang ada diinternet

misalkan kalian pakek kebetulan nama variablenya sama tapi fungsinya beda

itu akan menyebabkan bentrok karena perilakunya beda makanya gunakan IIFE / SIAF buat menjaga satu file javascript ini kedalam sebuah function

exspression jadi nanti kalian tulisnya didalam function, jadi kita ngakalinnya

pakai function exepression paham ya

nah itu cara lamanya sekarang dengan adanya javascript yang baru ya es6

itu kita cukup menggunakan let saja dan itu sudah bisa melakukan perlaku

yang seperti ini jadi kita tidak perlu ribet lagi bikin function kita ganti varnya

jadi let saja begini

// for(let i = 0; i < 10; i++){

//     console.log(i);

// }

// console.log(i)

tuh sama persis jadi kita tidak bisa mengakses dari luar blocknya karena   
sekarang begitu kalian pakai let dia menganut block scope bukan lagi

function scope begitu temen temen mudah mudahan sampai disini paham

nah sekarang pertanyaannya gini, kenapa sih kok javascript itu menganut

function scope padahal bahasa pemrogaman yang lain sudah menganut block

scope ? karena pada saat itu block scope yang benar

kenapa pada saat pembuatan javascript itu langsung saja menganut block

scope? kata pak brandon each waktu 10 hari itu tidak cukup untuk membuat

block scope nah itu lah kenapa agak aneh ya perilakuknya jadi seperti itu

awalnya memang pak brandon each sudah tahu bahwa function scope itu

bukan lah sesuatu yang tepat

oke jadi sekarang kita tahu ya kalo let dan const itu menganut block scope dan itu yang bener maka saya sarankan jangan pernah lagi menggunakan var

silahkan gunakan dua ini karena dia menganut block scope

kita sudah tahu let itu seperti apa

nah sekarang bedanya yang const itu apa ? coba kita lihat kalo misalnya kita

punya let i sama dengan 10

// let i = 10;

nah ini caranya sudah benar tapi kalo misalnya kita yakin nanti didalam progamkita, kita tidak akan berubah I nya  gunakan const

// const i = 10;

jadi kalo misalkan saya ubah isinya begini

// i = 20;

// console.log(i);

maka kalo saya jalankan itu dia eror kalo pakai let tidak akan eror jadi kalo

kalian bisa pastikan nilai dari variable ini isinya tidak pernah diganti ya gunakanconst bahkan ada yang bilang gunakan let pada saat kalian menggunakan   
looping saja seperti yang sudah kita coba tadi ya

// for(let i = 0; i < 10; i++){

// }

karena i nya itu berubah dia increment ya terus menerus sampai 10 kali

maka gunakan let karena kalo pakai const dia tidak bisa di ++ pahamya

jadi nanti kita akan banyak gunakan const ini ketika nilainya tidak berubah

nah sekarang pertannyaan selanjutnya adalah kenapa kok ada const ?

kenapa tidak let saja begitu ? karena kita menggunakan const itu supaya

meminimalisit perubahan state

jadi sebisa mungkin nilai dari variable itu tidak berubah tapi sebetulnya const

ini tidak sepenuhnya tidak bisa diubah ya atau istilahnya imiutable

apa itu imiutable dan imiutabelity? itu akan saya bahas dimateri yang lain

jadi const itu tetep bisa diubah tapi caranya lain misalnya saya punya mhs yangberisi object

// const mhs = {

//     nama : 'muiz',

//     umur : 18

// }

// console.log(mhs)

kalo misalkan kita ubah umurnya

// mhs.umur = 20;

// console.log(mhs);

nah ini masih bisa asalkan kita tidak mengganti objectnya kalo misalnya kita

ganti mhsnya dengan object baru misalnya gini

// mhs = {

//     jurusan : 'teknik informatika'

// }

// console.log(mhs);

kita ganti bener bener menjadi object yang baru dia akan eror, kita tidak bisa   
mengisikan sesuatu yang baru paham ya begitu pun dengan array, kalo kalian

bikin array 123

const myArr = [1,2,3];  
nah sekarang isinya diganti dengan array yang beru

// myArr = [1,2,3,4];

// console.log(myArr);

ini akan eror tapi kalo kalian mau menambahkan angka 4 saja diindex

terakhirnya

myArr.push(4);

console.log(myArr);

coba kita lihat nilainya masuk dan array tidak berubah jadi kita bisa ganti

isinya tapi kita tidak bisa benar benar mengganti semua isinya mudah

mudahan paham

ya sekali lagi kenapa kita gunakan const untuk meminimalisir perubahan state

karena kalo kita nimpa nimpa isi variable itu makin lama progamnya makin   
besar itu akan semakin merepotkan

nah merepotkannya gimana? gini deh contoh sederhananya, pernah gak kalian

dirumah pakai internet terus suatu saat internet kalian jadi lambat lah atau

misalkan jadi gak jalan internetnya yang kepikiran sama kalian apa pasti

televon ke customer servicenya ya

terus nanya ini kenapa internet saya lambat dan gimana cara mengatasinya

pasti disuruhnya untuk merestart modemnya terus kalian restart nah ternyata

jadi jalan internetnya nah kenapa bisa jalan? karena begitu kita restart pada

modem kita itu akan membersihkan keadaan sebelumnya jadi statenya dimulailagi dari 0 semua pengulangan pengkondisian semua balik ke keadaan awal

nah itu maksutnya dengan meminimalisir perubahan state tadi

baik jadi itu ya temen temen materi kali ini mudah mudahan kalian paham

mengenai penggunaan let dan const ini

sekali lagi sekarang tidak usah lagi menggunakan var gunakan saja

const kalo misakan kalian yakin nilainya tidak akan berubah

kalo nilainya akan berubah gunakan saja let

jadi sampai disini dulu materinya ya

dan satu lagi jangan lupa titik koma ;

**7.Javascript lanjutan: arrow function**

kali ini kita akan masuk ke topik yang namanya arrow function

apa itu arrow function? kita lihat dulu contoh code barikut ini

jadi mungkin kalian sudah familiar dengan cara pembuatan function

dijavascript yang disebut dengan function declaration

function tampilPesan (nama){

    alert('halo' + nama)

}

tampilPesan('muiz)

nah disini kita mendeklarasikan function seperti bahasa bahasa pemrograman

lainnya kita tulis keywordnya function lalu nama function lalu gunakan kurung untuk menyimpan parameter lalu kurung kurawal buka dan tutup lalu diisi

dengan body dari functionnya

nah jadi kita deklarasikan dulu baru dipanggil dan untuk panggilnya cukup tulis nama function diikuti dengan kurung buka dan tutup, ini kayaknya tidak

ada masalah ya

nah dijavacript itu kita juga bisa membuat function menggunakan yang

namanya function exspression

let tampilNama = function(nama){

    alert(nama)

}

tampilNama('muiz)

dimana functionnya disimpan dulu kedalam sebuah variable untuk cara

penggilnya sama tinggal tulis nama variable lalu diikuti dengan kurung buka

dan tutup untuk mengisikan parameter, gitu jadi ada dua jenis function

nah selanjutnya arrow function itu gimana? arrow function ini kalo menurut

mozila developer network

*arrow function adalah bentuk lain yang lebih ringkas dari function exspression*

jadi ini adalah cara lain untuk kita menuliskan function tapi menjadi lebih

ringkas, contohnya tadi kita punya function exspression seperti ini

let tampilNama = function(nama){

    alert(nama)

}

tampilNama('muiz);

kalo mau kita ringkas berubah menjadi seperti ini

let tampilPesan = (nama) => {

    alert(nama)

}

tampilPesan('muiz');

jadi keyword functionnya hilang ya hanya ada kurung buka dan tutup untuk

menyimpan parameternya lalu tanda panah gabungan dari sama dengan dan

lebih besar dari lalu isinya sama bisa menggunakan kurung kurawal

jadi simplenya begitu untuk mengubah penulisan function exspression agar

lebih ringkas lagi tapi nanti kita lihat ada beberapa hal yang berbeda dengan

function exspression, itu akan kita bahas dimateri kali ini dan dimateri

berikutnya

oke ya langsung kita coba saja bagaimana cara membuat arrow function pada

javascript

// const tampilNama = function (nama){

//     return `halo ${nama}`

// }

// console.log(tampilNama('muiz'));

saya punya variable yang namanya tampilNama yang isinya adalah function

yang didalamnya ada return dan template literal dan untuk template ini

akan kita bahas nanti dimeteri terpisah

kita coba jalankan, tampil ya muiz, sekarang gimana caranya mengubah

code tersebut menjadi arrow function ? jadi caranya kita bikin dulu variable

// const tampilNama = (nama) => {return `hallo ${nama}`}

// console.log(tampilNama('muiz'))

nah sekarang variablenya diisi dengan arrow function tidak perlu

menggunakan keyword function langsung saja kurung buka dan tutup untuk menyimpan parameter lalu tanda panah, sama dengan, lebih

besar dari, dan kurung kurawal

kalo kita jalankan nah muncul jadi kita bisa bikin lebih ringkas seperti ini, paham ya dan cara penulisan arrow function ini ada beberapa bentuk, code yang diatas kalo misalkan kita punya satu parameter

1.parameternya lebih dari satu

nah kalo misalkan parameternya lebih dari satu, ya tinggal tambahkan

koma saja didalam kurung buka dan tutupnya

2.parameternya hanya satu

selanjutnya kalo misalnya parameternya satu sebenarnya kalian tidak perlupakai kurung seperti ini

// const tampilNama = nama => {return `hallo ${nama}`}

// console.log(tampilNama('muiz'));

3.isi function hanya return

lalu kalo misalnya isi functionnya hanya return saja seperti code diatas

kalian bahkan tidak perlu menulis returnnya dan bahkan kalian tidak perlu menulis kurung kurawalnya

// const tampilNama = nama => `hallo ${nama}`

// console.log(tampilNama('muiz'))

jadi sama saja,ini disebutnya dengan implisit return jadi returnnya tidak

usah ditulis walaupun ini sangat ringkas kalian harus bener bener paham

maksutnya satu parameter dan isinya hanya return saja tidak ada baris

yang lain

4.tanpa parameter

dan kalo misalnya tanpa parameter kalo kalian tidak ada parameternya

wajib menulis kurung buka dan tutupnya, begini

// const tampilNama = () => {return `hallo world`}

// console.log(tampilNama());

jadi ini kalo tidak ada parameternya lebih ringkas lagi

begitu ya jadi ini beragam cara membuat arrow function pada javascript

mudah mudahan sampai disini paham, sekarang pada kenyatannya nanti

kalian tidak akan membuat fungsi yang sangat sederhana seperti ini

kita akan buat contoh lain dari arrow function misalnya dengan menggunakan

function map pada javascript

contohnya begini misalnya saya punya sebuah array namanya mahasiswa

yang isinya ada tiga orang mahasiswa, ceritannya kalian ingin menghitung

jumlah huruf dari masing masing mahasiswa kalo menggunakan function biasa

itu seperti ini

// let mahasiswa = ['muiz', 'husain', 'hasan'];

// let jumlahHuruf = mahasiswa.map(function(nama){

//     return nama.length

// })

// console.log(jumlahHuruf)

kalo ini kalian jalankan maka akan ada array baru yang isinya jumlah huruf dari tiap tiap value pada array, sekarang gimana kalo menggunakan arrow

function? coba kita ubah function diatas menjadi arrow function ya

// let mahasiswa = ['muiz', 'husain', 'hasan'];

// let jumlahHuruf = mahasiswa.map(nama => nama.length);

// console.log(jumlahHuruf);

nah karena parameternya cuma satu kalian boleh tidak menulis kurung buka

dan tutupnya lalu tulis tanda panah karena didalam functionnya hanya berisi

return kita bisa hilangkan returnnya dan hilangkan kurung kurawal buka dan

tutupnya

jadi sekarang jauh lebih ringkas kalo kita jalankan harusnya sama dan terakhir

kalo misalkan kalian ingin mengembalikannya dalam bentuk object bukan arraylagi, kalo object kita bisa tulisan kurung kurawal didalam objectnya

let mahasiswa = ['muiz', 'husain', 'hasan'];

let jumlahHuruf = mahasiswa.map(nama => {})

nah tapi problemnya adalah kalo saya tulis begini javascriptnya akan

menganggap kita mau melakukan return bukan membuat object supaya jadi

object kalian harus bungkus lagi menggunakan kurung buka dan tutup

// let mahasiswa = ['muiz', 'husain', 'hasan'];

// let jumlahHuruf = mahasiswa.map(nama => ({nama : nama, jumlahHuruf :

nama.length}))

// console.log(jumlahHuruf)

sekarang baru didalamnya kalian buat objectnya jadi propertinya ada nama

yang diisi dengan value arraynya dan jumlahHuruf, kalo saya jalankan, maka

tampil menjadi object dan kalo kalian ingin tampilnya rapi tidak bentuknya

object kalian bisa ganti console.lognya dengan console.table

// let mahasiswa = ['muiz', 'husain', 'hasan'];

// let jumlahHuruf = mahasiswa.map(nama => ({nama : nama, jumlahHuruf :

nama.length}))

// console.table(jumlahHuruf);

nanti ini akan berubah menjadi table begitu dan ini kalo misalkan kalian mau

mengembalikan object yang propertinya sama dengan nilainya kalian cukup

menulisnya satu saja

let mahasiswa = ['muiz', 'husain', 'hasan'];

let jumlahHuruf = mahasiswa.map(nama => ({nama, jumlahHuruf :

nama.length}))

console.table(jumlahHuruf);

gitu ya temen temen jadi mungkin itu dulu penjelasan awal mengenai arrow

function ini, bagaimana cara kita membuat arrow function dan beragam cara

pembuatannya mudah mudahan kalian paham dan dimateri berikutnya kita

akan lanjutkan lagi membahas perilaku lain dari arrow function ini

ya jadi itu saja materinya

kita ketemu dimateri berikutnya

dan satu lagi jangan lupa titik koma ;

**8.Javascript lanjutan: this pada arrow function**

nah dimateri kali ini kita akan cari tahu mengenai context dari this jika kita

menggunakan arrow function jadikan kalian tahu kalo kita menggunakan

function declaration atau function exspression didalamnya terdapat konsep

this, jika kita mengubah function exspression dengan arrow function

maka context thisnya itu menjadi berbeda

untuk lebih jelasnya kita lihat contoh codenya, sekarang saya mau ngasih

contoh jika kita mau membuat constructor function, misalnya saya mau

membuat sebuah object mahasiswa

// const Mahasiswa = function(){

//     this.nama = 'muiz',

//     this.umur = 18

//     console.log(this)

// }

// let muiz = new Mahasiswa()

nah kalo kita mau membuat constructor function maka jika kita menuliskan

this didalam functionnya itu akan mengacu kepada objectnya misalnya saya

akan masukkan kedalam console thisnya kalo kita jalankan maka akan muncul object Mahasiswa

ini adalah context thisnya, jadi thisnya mengacu ke objectnya, misalkan

sekarang kalo kita mempunyai sebuah method misalkan sayHallo yang isinya

juga adalah function exspression

// const Mahasiswa = function(){

//     this.nama = 'muiz',

//     this.umur = 18,

//     this.sayHallo = function(){

//     console.log(`halo nama saya ${this.nama} umut saya ${this.umur} tahun`)

//     }

// }

// const muiz = new Mahasiswa()

kalo saya jalanin muncul ya, kalo gini tidak ada masalah kita buat function yangdidalamnya punya this yang otomatis mengacu keobjectnya ketika diinstansiasi

nah sekarang gimana kalo kita ubah ini menjadi arrow function, untuk

menggunakan arrow function kalian tidak bisa begitu saja mengubah semua

function menjadi arrow function karena kalo ini kita ubah menjadi arrow

function hasilnya gini

// const Mahasiswa = () =>{

//     this.nama = 'muiz',

//     this.umur = 18,

//     this.sayHallo = function(){

//         console.log(`halo nama saya ${this.nama} umut saya ${this.umur} tahun`)

//     }

// }

// const muiz = new Mahasiswa()

nah ini kalo dijalankan maka akan eror, katanya mahasiswa is not a constructor

jadi tidak bisa kalo bikin constructor function menggunakan arrow function

tapi kalo method bisa, kalo method kita ganti menjadi arrow function jadinya

begini

// const Mahasiswa = function (){

//     this.nama = 'muiz',

//     this.umur = 18,

//     this.sayHallo = () =>{

//         console.log(`halo nama saya ${this.nama} umut saya ${this.umur} tahun`)

//     }

// }

// const muiz = new Mahasiswa()

nah kalo ini dijalankan tidak akan eror, kelihatannya sama saja ya kalo pakai

arrow function atau function exspression fungsinya kelihatannya sama saja

tapi sebetulnya pada saat kita menggunakan arrow function, arrow function

sebetulnya tidak menyimpan context this jadi tidak ada context this didalam

functionnya dia akan mencari keluar dilexical scopenya

kelihatannya sama saja menggunakan function biasa atau menggunakan arrow function sekarang lihat ini akan berbeda ketika kita bikin objectnya tidak

menggunakan constructor function tapi pakai object literal

// const mahasiswa = {

//     nama : 'muiz',

//     umur : 18,

//     sayHallo : function (){

//         console.log(`halo nama saya ${this.nama} umur saya ${this.umur} tahun`)

//     }

// }

halo ini saya jalanin muncul ya isinya, jadi kalo menggunakan function thisnya

berisi object yang bersangkutan tapi ini akan beda ketika saya ganti

methodnya dengan arrow function

// const mahasiswa = {

//     nama : 'muiz',

//     umur : 18,

//     sayHallo : () =>{

//         console.log(`halo nama saya ${this.nama} umur saya ${this.umur} tahun`)

//     }

// }

kalo ini saya jalankan saya tulis diconsole.lognya mahasiswa.sayhallo()

sekarang dia nilainya undefined karena ini arrow function maka dia akan

mencari dilocal scopenya ada this gak? gak ada

cari ke atasnya ada this gak? tidak ada lalu keluar dari object mahasiswa dia

mencari thisnya ada gak this nah ketemu diglobal scope this adalah window

maka this.nama nilainya undefined

kalo tadi pakai constructor function begitu this ini tidak ada dia akan mencari

keatas ketemu yang this.nama, sekarang kalo misalkan saya console.log

this nya maka isinya adalah object window

itulah kenapa saya bilang tadi arrow function tidak memiliki konsep this

gitu ya mudah mudahan sampai disini kalian bisa paham kalo kalian masih

bingung kita coba contoh lain, nah sekarang saya mau kasih kasus gini

didalam constructor function ini saya mau bikin sebuah function yang otomatisdijalanin

misalnya saya mau menggunakan function yang namanya setInterval

yang otomatis dijalankan pada interval tertentu yang didalamnya saya ingin

setiap 0,5 detik umurnya nambah 1

// const Mahasiswa =  function(){

//     this.nama = 'muiz'

//     this.umur = 18

//     this.sayHallo = function(){

//         return `hallo nama saya ${this.nama} umur saya ${this.umur}`

//     }

//     setInterval(function(){

//         console.log(this.umur++)

//     },500)

// }

// const muiz = new Mahasiswa()

kalo misalkan kita menggunakan function declaration, setInterval adalah

function declaration, kalo sayHallo itu adalah function exspression maka   
sayhallo ini tidak akan kena hoisting

nah tapi begitu kita menggunakan function declaration ini akan kena hoisting

sehingga pada saat thisnya dipanggil dia tidak mengecheck didalam object

Mahasiswanya tapi dia mengecheck diglobalnya

nah coba kita jalankan saja, lihat tuh munculnya NaN bukan angka katanya

nah kenapa bukan angka? karena yang dia jumlah bukan 18 tapi coba kita lihat, kita console this saja, this itu window jadi kalo saya tulis this.umur itu nilainya undefined tidak ada

nah itu scope this kalo kita bikin function declaration karena function

declaration berada diobject diwindow maka this itu window lalu gimana dong

cara mengetasinya? nah kita bisa gunakan arrow function tadi

// const Mahasiswa =  function(){

//     this.nama = 'muiz'

//     this.umur = 18

//     this.sayHallo = function(){

//         return `hallo nama saya ${this.nama} umur saya ${this.umur}`

//     }

//     setInterval(() =>{

//         console.log(this)

//     },500)

// }

// const muiz = new Mahasiswa()

nah coba pahami logikanya, sekarang kalo ini dijalanin arrow function

tidak punya konsep this karena dia tidak punya konsep this maka thisnya akan mencari kelexical scopenya dia mencari keluar nah ketemu lah this kalo

sekarang saya jalanin thisnya maka isinya adalah Mahasiswa, sehingga kalo

saya tulis this.umur++ maka dia akan nambah tuh umurnya

// const Mahasiswa =  function(){

//     this.nama = 'muiz'

//     this.umur = 18

//     this.sayHallo = function(){

//         return `hallo nama saya ${this.nama} umur saya ${this.umur}`

//     }

//     setInterval(() =>{

//         console.log(this.umur++)

//     },500)

// }

// const muiz = new Mahasiswa()

coba jalanin maka dia pasti nambah, gitu temen temen jadi itu perbedaan

context this pada arrow function jadi sekali lagi kita tidak boleh mengubah

secara sembarangan pengen keren nih ubah saja menjadi arrow function

tidak bisa begitu ya tergantung kebutuhan mudah mudahan sampai disini

mulai ada gambaran ya

sekarang pertannyaannya arrow function ini dipakainya dimana sih contoh realdari arrow function itu kita pakai dimana? kita coba sekarang kasus yang bisa

kita terapkan direal lifenya, kita coba buat kotak yang saya kasih class box

didalam web kita tidak perlu lebar lebar cukup 100px aja lalu kita kasih css dulu

.box{

    width: 100px;

    height: 100px;

 border: 1px solid;

    margin: 20px auto;

    transition:   // untuk transition tiap tiap perubahannya

                .3s width,

                .3s height .3s,

                .3s margin-top .3s,

                .3s background-color;

}

.size{

    width: 150px;

    height: 150px;

    margin-top: 40px;

}

.b{ background-color: black;}

nah jadi ada kotak 100 x 100, terus saya punya dua buah class ceritanya saya

mau buat animasi ketika tombol diklik nanti dengan menggunakan javascript

ketika diklik kotak tersebut akan saya kasih class yang namanya size

nah nanti setelah nunggu 0.6 detik, ya karena itu ada durasinya 0.3 detik dan

delaynya 0.3 detik jadi 0.6 detik ya saya mau tambahkan class lagi yang

namanya b, sekarang yang akan kita lakukan adalah menyeleksi kotak tersebut

menggunakan DOM

// const box = document.querySelector('.box')

// box.addEventListener('click', function(){

//     console.log(this)

// })

coba saya tulis didalamnya ketika diklik munculkan this, coba kita jalankan jadi

this disinya adalah elementnnya, sekali lagi ini kalian tidak bisa ubah menjadi

arrow function kalo kalian ubah menjadi arrow function isinya adalah object

window, oke ceritannya kalo diklik saya ingin memberikan class pada kotak ini

nah kita bisa lakukan ini karena kita tahu this disini adalah elementnya maka

saya bisa tulis begini

// const box = document.querySelector('.box')

// box.addEventListener('click', function(){

//     this.classList.toggle('size')

// })

nah kalo kita jalanin, kalo kotaknya saya klik nah dia akan menbahkan class

size sekarang yang akan saya lakukan adalah saya akan menunggu selama

0.6 detik baru saya kasih class b jadi kita bisa pakai setTimeout()

// const box = document.querySelector('.box')

// box.addEventListener('click', function(){

//     this.classList.toggle('size');

//     setTimeout(function(){

//         this.classList.toggle('b')

//     }, 600)

// })

kalo setTimeout dia akan menunggu dulu lalu jalankan codenya selama delay

tertentu kalo saya jalankan tapi setelah menggungu 0.6 detik yang b tidak jalannih kira kira kenapa? karena ketika kita menggunakan function maka this

disini dia akan berisi apapun yang ada diluarnya

nah karena setTimeout ini dijalankan ketika hoisting maka dia posisinya ada

diglobal scope jadi jika kalian console this ini isinya adalah object window,

harusnya this disini isinya adalah box

nah sekarang gimana cara mengatasinya ? kalo dulu sebelum ada arrow

function kita bisa akalin caranya dengan membuat sebuat variable yang diisi

this

// const box = document.querySelector('.box')

// box.addEventListener('click', function(){

//     let that = this

//     this.classList.toggle('size');

//     setTimeout(function(){

//         that.classList.toggle('b')

//     }, 600)

// })

jadi menulisnya didalam setTimeoutnya that bukan this sekarang kalo dijalanin

tuh berubah tapi sekarang kita sudah tahu ada cara lain yaitu dengan

menggunakan arrow function

// const box = document.querySelector('.box')

// box.addEventListener('click', function(){

//     this.classList.toggle('size');

//     setTimeout(() =>{

//         this.classList.toggle('b')

//     }, 600)

// })

nah sekarang kalo dijalanin tuh sudah bener ya thisnya mengacu kebox

nah karena dilexical scopenya ada this maka this isinya adalah box gitu ya

tapi ini animasinya harus diperbaiki lagi ya

kalo misalkan saya klik lagi hilangnya harusnya b dulu baru sizenya tapi disini

adalah animasi kotaknya dulu baru backgroundnya ya jadi kebalik tuh

ini kalo mau dibenerin kita bisa akalin gini deh kita masukkan 2 class tersebut

kedalam variable lalu kita check dulu apakah sudah ditambahkan class sizenya

kalo sudah ditambahkan, kita balik variable pertama diisi dengan b dan

variable kedua diisi dengan size

// const box = document.querySelector('.box')

// box.addEventListener('click', function(){

//     let satu = 'size'

//     let dua = 'b'

//     if(this.classList.contains(satu)){

//         satu = 'b'

//         dua = 'size'

//     }

//     this.classList.toggle(satu);

//     setTimeout(() =>{

//         this.classList.toggle(dua)

//     }, 600)

// })

nah biasanya begini tapi dengan menggunakan javascript yang baru kita bisa

gunakan sintax untuk membalikkan isi dari variable namanya

destructuring assignment syntax jadi saya bisa tulis begini

const box = document.querySelector('.box')

box.addEventListener('click', function(){

    let satu = 'size'

    let dua = 'b'

    if(this.classList.contains(satu)){

        [satu, dua] = [dua, satu]

    }

    this.classList.toggle(satu);

    setTimeout(() =>{

        this.classList.toggle(dua)

    }, 600)

})

nah sekarang kita lihat hasilnya coba kalian klik sudah bener ya begitu jadi

mudah mudahan kalian paham bagaimana penggunaan dari arrow function

oke temen temen mungkin itu saja untuk materi kali ini

mudah mudahan paham dan tidak bingung

kita akan ketemu lagi dimateri berikutnya

dan seperti biasa jangan lupa titik koma ;

**9. Javascript lanjutan: higher order function**

kita masuk ketopik yang baru mengenai sesuatu yang dinamakan dengan

higher order function ini cukup penting ketika kalian ingin belajar mengenai

modern javascript

nah dimateri kali ini kita akan bahas dulu teorinya apa itu definisinya dan

contohnya seperti apa nanti dimateri berikutnya kita akan bahas implementasi

dari higher order function ini tapi sekarang kita lihat dulu definisi dari higher

order function ini

jadi menurut eloquent javascript.net

*higher order function pada javascript adalah function yang beroprasi pada*

*function yang lain, baik itu digunakan sebagai argument maupun sebagai*

*return value*

nah karena didalam javascript itu sebuah function disebut dengan first class

function artinya inti dari javascript itu adalah function dimana function ini

diperlakukan sebagai object didalam javascript dan kalo kalian tahu object itu merupakan value sama seperti intejer atau string misalnya

maka function juga kita bisa simpan sebagai argument maupun return

value dari sebuah function yang lain, sebetulnya ini sudah kita coba ketika

kita belajar mengenai closure dimateri sebelumnya

contoh misalkan saya punya sebuah function seperti ini

function kerjakanTugas(mataKuliah,  selesai){

    console.log(`Mulai mengerjakan tugas ${mataKuliah} `)

    selesai()

}

nah jadi kita punya sebuah function yang memiliki dua buah parameter

nah parameter mataKuliah nanti kita simpan berupa string misalnya mata

kuliahnya pemrograman web lalu parameter kedua adalah selesai yang

merupakan function

jadi diluar functin kerjakanTugas ini kita harus membuat

sebuah function  yang tulisannya seperti ini

function selesai(){

    alert('selesai mengerjakan tugas');

}

jadi kalian bisa lihat nih kita menggunakan function sebagai argument

dan ketika kita panggil bisa tulis begini

kerjakanTugas('pemrograman web', selesai);

parameter yang pertama adalah string dan parameter yang kedua adalah

function nah jadi ketika kita jalankan diconsole akan tampil dan dibrowsernya

akan tampil alertnya

tapi kalian bisa langsung saja simpan functionnya didalam argumentnnya

menggunakan anonymous function, ini hanya contoh saja bahwa ketika kita

menggunakan function kita bisa menyimpan function lagi sebagai argument

ya kalo sudah begini function kerjakan tugas ini bisa kita sebut dengan higher

order function dan jika kalian punya function sebagai argument nantinya ini

kita sebut dengan callback

oke jadi function yang kita simpan kedalam argument kita sebut sebagai

callback sedangkan function yang memiliki argument berupa function itu

disebut dengan higher order function

mudah mudahan sampai disini paham nih atau kita lihat contoh yang lain ya

dibeberapa materi sebelumnya kita sudah coba menggunakan setTimeout()

atau setInterval()

setTimeou(function(){

    console.log('hello world')

}, 1000)

nah setInterval() ataupun setTimeout() memiliki dua parameter, parameter

yang pertama adalah function lalu parameter yang kedua adalah intejer untuk menyimpan milisecond waktunya  tapi kalian bisa lihat parameter pertamanya adalah sebuah function atau tadi istilahnya adalah callback

sehingga setTimeout ataupun setInterval kita bisa sebut dengan higher order

function atau misalkan contoh lainnya

const submit = document.querySelector('.submit')

submit.addEventListener('click', function(){

    console.log('tombol ditekan!')

})

ketika kalian ingin membuat sebuah tombol yang begitu tombolnya diklik lalu melakukan aksi tertentu kalo menggunkan DOM itu kan kita bisa bikin variable lalu kita ambil elementnya bisa menggunakan querySelector lalu kita

tambahkan addEventListener lalu eventnya apa dan callbacknya apa

nah ini juga bisa kita anggap sebagai higher order function dan contoh lainnya

ketika kita punya function yang return valuenya adalah function lagi

function ucapkanSalam(waktu){

    return function (nama){

        console.log(`halo ${nama}, selamat ${waktu}  semoga harimu menyenangk

an)

    }

}

let selamatMalam = ucapkanSalam('malam')

let muiz = selamatMalam('muiz')

nah ini sudah kita coba ya dimateri closure kita punya function yang namanya

ucapkanSalam yang didalamnya mereturn function lagi, sekarang function

ucapkanSalam ini juga kita bisa sebut dengan higher order function karena

didalamnya memiliki return value yang juga function

sekarang mungkin pertanyaannya adalah kenapa sih kita harus menggunakan

higher order function kenapa kita tidak bikin function biasa saja? ada beberapa alasannya

1.abstraksi

    alasan yang pertama untuk sesuatu yang dinamakan dengan abstraksi untuk

membuat agar code yang kita buat itu bisa lebih sederhana atau bisa lebih

simple karena dengan menggunakan function itu artinya kita

menyembunyikan kerumitan code kita

karena semakin besar sebuah progam semakin tinggi

kekomplexitas didalamnya maka akan semakin membingungkan

programmernya

    jadi makin banyak codenya nah itu akan semakin membingungkan

    nah ada baiknya kita pisahkan dengan function function sendiri

dan ketika kita punya function didalam function nah itu harusnya akan

membuat progamnya menjadi lebih simple

nah terus ada sebuah anekdot

ada dua buah cara merancang sebuah softwere cara pertama adalah untuk

membuat progamnya sesederhana mungkin sehingga jelas jelas tidak ada

kekurangannya cara lainnya adalah untuk membuat progamnya sekomplex   
mungkin sehingga tidak ada kekurangan yang jelas

nah cara yang kedua ini kekurangannya ada tapi kekurangannya tidak jelas

dari sini kita bisa lihat bahwa semakin simple progamnya semakin kecil

kemungkinan ada kesalahan didalamnya

nah sekarang kita lihat contoh berikut ini jadi misalkan saya punya progam

sederhana seperti ini

let total = 0,  count = 1

while(count <= 10){

    total += count

    count += 1

}

console.log(total)

jadi kita punya looping untuk totalnya ya jadi total akan nambah 1 nambah 2

sesuai dengan nilai countnya dengan menggunakan while kita bisa tulis seperti ini sekarang bandingkan dengan saya menulisnya begini misalnya

console.log(sum(range(1, 10)))

ini kalo misalkan kalian jalankan mungkin eror ya tapi bayangkan saja ada

definisi function yang namanya sum dan range

ketika kalian manggil dua function ini secara bersamaan jadi ada function yang menjadi parameter function yang lain maka akan lebih simple  cukup satu barissaja karena kerumitannya kita sembunyikan kedalam function

oke atau misalnya kita punya looping seperti ini lah

for(let  i = 0; i < 10; i++){

    console.log(i)

}

seperti tadi tapi sekarang kita gunakan yang for ini mencetak console.log i nya

kelihatannya simple tapi gimana kalo misalkan saya ingin loopingnya tidak 10

kali bisa 100 kali atau 1000 kali

nah kan berarti kita harus merubah angka 10 itu ya jadi tidak dinamis

nah gimana kalo misalkan kita gunakan sebuah function

function repeatLog(n){

    for(let i = 0; i < n; i++){

        console.log(n)

    }

}

repeatLog(10)

jadi nulis fornya cuma sekali kita ganti angka 10nya menjadi variable n yang

menjadi argument dari repeatLog sehingga kalo kita mau mengulang sebanyak

apapun tinggal kasih repeat log lalu kasih argumentnya berapa ya kan simple

mau 10 kali tinggal tulis saja 10 mau 100 tinggal tulis 100 tanpa mengubah lagi

loopingnya

oke jadi dengan menggunakan function itu abstraksi tapi kita bisa bikin ini

lebih efektif lagi, pertannyaannya gimana kalo misalkan aksi yang dijalankan

itu bukan console.log tapi saya punya aksi yang lain

misalnya saya mau tampilnya kedalam alert kita bisa bikin menjadi seperti ini

function repeat(n, action){

    for(let i = 0; i < n; i++){

        action(i)

    }

}

repeat(10, console.log)

repeat(10, alert)

jadi sekarang function repeatLognya saya ganti dengan repeat lalu

parameternya ada dua n untuk menentukan aksinya mau diulang berapa

kali dan action untuk aksi apa yang mau diulang kalo kalian mau console.log

tinggal tulis ada console.log kalo mau alert tinggal tulis saja alert

jadi kasus ini bisa kita buat lebih efektif dengan menggunakan higher order

function, gitu ya temen temen

dan alasan yang lain adalah dengan terbiasa

kita membuat segala sesuatunya dengan function itu kita masuk kesebuah

pendekatanyang namanya functional programming

ya saya tidak akan bahas lebih lanjut disini tapi dengan melakukan pendekatan kefunctional programming karena sebetulnya javascript bukan bahasa

pemrograman functional murni tapi bisa kita buat menjadi bahasa

pemrograman functionaldan ketika kita menggunakan paradikma ini dengan

benar itu akan membuat progam kita lebih efektif lebih aman dan lebih

efisien lagi

terakhir ini ada beberapa contoh dari higher order function ya terutama untuk pengelolaan array jadi inilah yang akan kita bahas dimateri materi berikutnya

1. Array.prototype.map()

2. Array.prototype.filter()

3. Array.prototype.reduce()

jadi harusnya kalian sudah punya minimal gambaran mengenai apa itu higher

order function dan kenapa kita gunakan ya

sehingga nanti dimateri berikutnya kita langsung praktekkan saja penggunaan higher order function dengan tiga function ini

oke jadi mungkin itu saja untuk materi kali ini ya

mudah mudahan paham

kita akan ketemu lagi dimateri berikutnya

dan seperti biasa jangan lupa titik koma ;

**10. Javascript lanjutan: filter, map, dan reduce**

nah dimateri kali ini kita akan membahas mengenai 3 contoh dari higher order function yang paling sering digunakan ya yaitu filter map dan reduce

nah jadi 3 function ini merupakan prototype dari array ya walaupun nanti

didalamnya isinya bisa macam macam ya dan sebetulnya kalo kalian mengikutiseri javascript dasar kita sudah membahas beberapa dari ini ya

terutama map sama filter

tapi waktu itu kita bahasnya masih menggunakan function biasa

kalo sekarang kita akan bahas menggunakan contoh arrow function

nah contoh higher order function yang lain misalnya seperti find forEach fill dll itu tidak akan kita bahas dan sekarang kita akan lihat ilustrasi dari penggunaan

masing masing function ini

1. filter

   let arr = ['rambutan', 'apel', 'bayam', 'wortel', 'jambu'];

array ini nantinya saya akan masukkan kedalam higher order function filter

dan saya akan memfilter mana diantara array ini yang termasuk sayuran

maka nanti function ini akan menghasilkan array baru hasil dari filternya

jadi diarray yang baru bahan makanan yang bukan sayur itu tidak akan terpilih

nah mudah mudahan terbayang kalo  kalian menggunakan filter

2. map

   let arr = ['rambutan', 'apel', 'bayam', 'wortel', 'jambu'];

nah dengan menggunakan kasus yang sama kita akan gunakan function yang

lain sekarang saya akan gunakan map

nah dengan menggunakan map  itu artinya kita akan lakukan sesuatu untuk

setiap element pada array kedalam sebuah function sendiri

setiap element yang ada pada array arr itu akan saya lakukan potong

jadi saya nanti akan memberikan callback function didalam mapnya

adalah potong

maka setiap element pada arr ini akan menghasilkan array baru berupa buah   
atau sayuran yang sudah dipotong

ini kalo kita gunakan map, jadi akan ada array baru hasil dari higher order

function tersebut

3. reduce

   let arr = ['rambutan', 'apel', 'bayam', 'wortel', 'jambu'];

nah hasil dari function map ini  bisa kita kasih higher order function berikutnya yaitu reduce

nah kalo reduce ini kita bisa lakukan misalnya kita bisa menggabungkan semua value  yang ada pada array arr untuk menjadi hasil yang baru

jadi ini adalah ilustrasi dari filter map dan reduce mudah mudahan kalian

kebayang dan dengan reduce ini pun sama dia menghasilkan array yang baru

dan nanti kerennya ketiga higher order function ini kita bisa gabungkan denga menggunakan yang namanya method chaining, nah nanti kita lihat ya

mudah mudahan kalian paham mengenai konsep filter map dan reduce

dan supaya lebih jelas lagi gimana kalo sekarang kita coba implementasikan kedalam code kita apa yang sudah kita pahami

nah misalnya saya punya array yang isinya angka

// const angka = [1, 4, 3, 0, -1, 5, 9, 8];

1.filter

yang pertama kita akan coba filter

nah ceritannya gini saya ingin mencari angka yang lebih besar sama dengan 3

didalam array angka

nah sebelum kita gunakan filter coba kalian gunakan for.....

sakarang kita coba pakai filter

// const newAngka = angka.filter(function(a){

//    return a >= 3

// })

// console.log(newAngka)

kita tampun kedalam variable newAngka lalu kita langsung panggil saja array

mana yang mau kita filter kita tulis angka.filter() yang didalam argument filter

ada callbacknya

didalam parameter callbacknya kita bisa tulis sembarang ya untuk representasidari tiap tiap nilai pada array angka untuk tiap tiap value pada angka kita

return jika a nya lebih besar sama dengan 3

apakah 1 lebih besar 3 false ya

apakah 4 lebih besar 3 true masukkan kedalam variable newAngka dan seterusnya lalu coba saya console.log newAngkanya

nah muncul tuh nilai yang lebih besar sama dengan 3

keren ya dan kalo kalian inget mengenai arrow function,callbacknya kita bisa

ganti dengan arrow function

// const angka = [1, 4, 3, 0, -1, 5, 9, 8];

// const newAngka = angka.filter(a => a >= 3)

// console.log(newAngka)

kita tulis a untuk parameternya lalu tanda panah dan kalo isinya hanya return

kita tidak perlu menulis kurung kurawal dan return cukup tuliskan apa yang

akan direturn

lihat ini jadi satu barisnya dan kalo kita save lalu jalanin akan sama hasilnya

seperti for gitu ya untuk filter

2.map

sekarang kita coba untuk yang map

kalo untuk map kan artinya untuk setiap nilai pada array kita lakukan sesuatu

misalnya saya mau kalikan semua angka dengan 2

ini artinya saya akan punya array baru yang isinya kali 2 dari semua isi value

yang ada didalam array

// const angka = [1, 4, 3, 0, -1, 5, 9, 8];

// const newAngka = angka.map(a => a \* 2)

// console.log(newAngka)

newAngka diisi dengan map

nah didalam argumentnya kita gunakan arrow function yang mengembalikan

a \* 2 ini artinya telusuri semua value pada array angka dan tiap tiap valuenya

kali dengan 2

kalo saya console.log maka akan ada array baru yang isinya tiap tiap nilai yang didalam value angka dikali 2 dan coba kalian jalankan

jadi semua dikalikan 2 dan ini tidak mengubah isi dari array angkanya

kalo kita console.log angka nilainya akan tetap sama

// console.log(angka)

itu ya untuk map , dan fungsinya bisa macam macam bisa tambah bisa kurang bagi dan lain lain jadi itu untuk map ya mudah mudahan kalian paham

3.reduce

dan yang terakhir kita gunakan reduce

reduce itu untuk melakukan sesuatu terhadap seluruh value pada arraynya

misalkan saya ingin arraynya saya jumlankan semua isinya jadi 1 + 4 + 3 , saya

jumlahin semuanya caranya sama, kita buat varible baru karena ini

menghasilkan array yang baru ya

// const angka = [1, 4, 3, 0, -1, 5, 9, 8];

// const newAngka = angka.reduce()

nah kalo reduce ini dia punya dua argument didalam functionnya

argument yang pertama itu adalah accumulator dan parameter yang keduanyaadalah currentvalue atau value pada array yang sedang diloopingnya

sedangkan accumulator adalah hasil dari prosesnya

misalnya ditambah nah berarti kalo kita lihat didalam arraynya awalnya 0 lalu 1nah 1 itu adalah currentvalue lalu jumlahkan 0 + 1 hasilnya 1, 1 disini adalah

accumulator

lalu dia geser hasilnya tadi yaitu 1 + 4 ,1 adalah accumulator, 4 adalah

currentvalue hasilnya 5, 5 adalah accumulator dan seterusnya jadi cara

bacanya begitu

// const angka = [1, 4, 3, 0, -1, 5, 9, 8];

// const newAngka = angka.reduce((accumulator, currentvalue) => accumulato

r + currentvalue)

// console.log(newAngka)

 karena kita mau menjumlahkan kita isi dengan return

accumulator + currentvalue dan kalo kita lihat hasilnya maka akan muncul

jumlah dari semua value pada array angka

nah ini kalian perhatikan, secara default ini ada satu nilai lagi

// const angka = [1, 4, 3, 0, -1, 5, 9, 8];

// const newAngka = angka.reduce((accumulator, currentvalue) => accumulato

r + currentvalue, 0)

// console.log(newAngka)

nah kalo kita lihat itu sebenarnya nilai awalnya adalah 0

ya jadi awalnya itu 0 + 1, 0 adalah nilai awal jadi kalo saya jalanin tidak akan

berubah nah nanti nilai awal ini bisa kalian ubah

terserah kalian mau berapa nilainya misalkan 5 nah kalo disini kalian tulis 5

maka nilai awalnya adalah 5 jadi 5 + 1

// const angka = [1, 4, 3, 0, -1, 5, 9, 8];

// const newAngka = angka.reduce((accumulator, currentvalue) => accumulato

r + currentvalue, 5)

// console.log(newAngka)

yang tadinya hasilnya 29 jadi 31 ya nanti ini akan bermanfaat ketika kalian maubikin nilainya udah ada isinya tidak 0 gitu tapi berdasarkan misalkan history

dari nilai user

kalo tidak ditulis defaultnya adalah 0 begitu ya jadi kita sudah coba nih filter

map dan reduce sekarang yang terakhir yang akan saya bahas adalah

mengenai method chaining yang artinya kita bisa menggabungkan fungsi fungsipada higher order function ini dalam satu kali eksekusi

tidak perlu disimpan kedalam variable dulu seperti tadi

ini disebut method chaining atau method berantai

contohnya saya punya kasus begini

dari array yang kita punya saya mau mencari angka yang lebih besar dari 5

setelah itu hasilnya saya mau kalikan dengan 3 terakhir saya mau jumlahkan

kalian bisa lakukan satu satu kayak tadi atau kita bisa gunakan method

chaining

const angka = [1, 4, 3, 0, -1, 5, 9, 8];

const hasil = angka.filter(a => a > 5)/\* hasilnya [9, 8]\*/.map(a => a \* 3)/\* hasilnya [27, 24]\*/.reduce((acc, curr) => acc + curr); // hasilnya 51

console.log(hasil)

nah yang pertama kita cari dulu angka yang lebih besar 5

lalu sekarang kita chaining kita tambahkan method disebelah filter

jadi kita bisa pakai map lalu kita kalikan tiap tiap nilai dengan 3

lalu kita jumlahkan semua menggunakan reduce yang isinya acc + curr

kalo kalian jalankan harusnya hasilnya 51

jadi ini kalo misalkan kita lakukan menggunakan method chaining

ini kalian boleh tulisnya berulang ulang lagi misalkan setelah map adalah

map lagi yang hasilnya dibagi 2 nah ini bisa juga ya

oke jadi begitu ya untuk filter map dan reduce mudah mudahan kalian paham

dan mungkin dimateri selanjutnya supaya lebih paham lagi kita akan bahas

mengenai filter map dan reduce tapi untuk kasus yang lebih real lagi ya

jadi sampai disini dulu materinya

dan seperti biasa jangan lupa titik koma ;

**11. Javascript lanjutan: template literal**

kali ini kita masuk ketopik baru yaitu mengenai template literal atau disebut

juga dengan template string dan dimateri sebelum sebelumnya sebetulnya

kita sudah pernah menggunakan template literal ini

ya walaupun sebelumnya itu namanya bukan string literal tapi template literal

nah untuk lebih jelasnya kita coba lihat definisi dari template literal ini

*“template literal adalah string literal yang memungkinkan ada exspression*

*didalamnya*”

apa itu string literal dan maksutnya exspression itu seperti apa? nanti kita lihat

ya jadi string literal itu string yang biasa kita buat ya, string literal itu biasanya   
kita buat dengan beberapa cara

bisa menggunakan kutip satu atau kita bisa gunakan kutip dua, dan fungsinya

itu sama saja, beda dengan php kutip dua itu lebih sakti karena dia bisa

melakukan interpolasi, nah ada cara lain untuk membuat string literal dengan

menggunakan backtick ``

dengan membuat string literal menggunakan backtick dapat memungkinkan

kita membuat template literal, jadi template literal ini hanya bisa dibuat

menggunakan backtick

template literal

dengan menggunakan backtick ini kita bisa melakukan banyak hal contohnya

- Multi-line string

    kita bisa membuat multi line string, sebelumnya kita tidak bisa melakukan

 multi line string dengan kutip satu atau dua

- embeded expression

   nanti kita lihat contohnya ya

- HTML fragments

    nah ini yang saat ini sering banget digunakan kalo misalkan kalian

menggunaka fond-end framework

- expression interpolation

- tagged template

Contoh penerapan template literal

- multi line string

    console.log('string text baris 1 \nstring text baris 2')

kalo kalian menggunakan kutip satu atau kutip dua terus kita ingin mencetak

string menggunakan console.log lalu stringnya akan ditampilkan dibaris

terpisah itu kalian harus menggunakan escape karakter \n untuk

menambahkan new line

    nah sedangkan kalo kita menggunakan backtick atau template literal

    kalian cukup tambahkan enter saja sesuai dengan kebutuhan satu enter

atau dua enter

    console.log(`string text baris 1    string text baris 1

    string text baris 2`) // hasilnya   string text baris 2

nah multi line string ini bisa kita gunakan juga ketika kita mau bikin html

fragments, misalnya saya punya object mahasiswa yang didalamnya terdapat

beberapa properti

// const mhs = {

//     nama : 'M.Muizzuddin',

//     umur : 18,

//     nrp : 45232,

//     email : 'muizzuddin332@gmail.com'

// }

lalu kita ingin menampilkannya kedalam halaman html kita, nah kalo

sebelumnya kita rangkai seperti ini misalnya saya punya variable yang

namanya element

// let el = ''

// el += '<div class="mhs">';

// el += '<h2>' + mhs.nama + '</h2>';

// el += '<span class="nrp">' + mhs.nrp + '</span>';

// el += '</div>';

// console.log(el)

awalnya string kosong lalu kita rangkai kita concatenasi dengan element yang

kita butuhkan lalu elementnya bisa kita simpan menggunakan DOM

menggunakan innerHTML, cukup ribet ya kalo mengguakan concate

nah ketika kalian membuatnya menggunakan template literal kalian cukup

membuatnya seperti ini

// let el = `<div class="mhs">

//     <h2> ${mhs.nama} </h2>

//     <span class="nrp"> ${mhs.nrp}</span>

//     </div>`

kalian tinggal tambahkan backtick lalu tulis semua sekaligus tag htmlnya, jadi

tidak usah digabung gabung menggunakan concate lagiwalaupun didalam

htmlnya itu terdapat white space ya karena waktu kalian bikin elementnya kan pakek tab begitu ya tapi itu tidak akan dipedulikan oleh browsernya

- embeded expression

jadi kita bisa menyimpan expression kedalam template literal

misalnya kalian ingin masukkan sebuah variable

// const nama = 'muiz'

// let umur = 18

// console.log(`hallo nama saya ${nama} umur saya ${umur} tahun`)

kalian cukup tambahkan dolar lalu kurung kurawal buka dan tutup dan

didalamnya tuliskan nama variablenya nah tapi expression ini juga bisa

langsung kita interpolasi, mislanya didalam expressionnya terdapat

penjumlahan

// let a = 10

// let b = 15

// console.log('jika a = 10 dan b = 15 maka hasil penjumlahnya adalah : ' + (a +  b))

nah sebelumnya kalo kalian pakek kutip satu atau dua ini kita harus buka tutupkutipnya karena kalo kita tuliskan tanda tambah didalam kutip satu atau kutip dua itu akan dicetak sebagai string bukan operasi matematik

jadi cukup ribet ya untuk menghasilkan seperti ini tapi jika kalian menggunakantamplate literal cukup gunakan backtick

// let a = 10

// let b = 15

// console.log(`jika a = 10 dan b = 15 maka hasil penjumlahnya adalah : ${a + b}

`)

lalu didalam dolar kurung kurawalnya itu kalian bisa melakukan interpolasi

expressionnya hasilnya akan sama persis, jadi tidak perlu ribet lagi buka dan

tutup stringnya dan expressionnya bisa sesimple penjumlahan seperti itu

atau kalian bisa juga memanggil function, bahkan kalian juga bisa menyimpan

ternary operator didalamnya

// const x = 10

// console.log(`${(x % 2 === 0)? 'genap' : 'ganjil'}`)

kalo saya jalankan hasilnya genap ya

- HTML fragments

kalo pakai kutip satu kita harus rangkai satu satu tapi kalo pakai backtick itu

kalian cukup tuliskan saja htmlnya didalam backticknya

nah tapi masalahnya kita tidak bisa menuliskan html menggunakan emmet

didalam file javascript kalo kalian pakai vs code

misalkan kalian mau bikin div yang classnya mahasiswa harusnyakan div.mhs

lalu enter, nah tapi kalian harus menuliskannya secara manual begini

 <div class="mhs"> </div>

kalo simple sih tidak apa apa tapi kalo misalkan htmlnya komplex kalian akan kesusahan, nah itu kita harus jalankan sebuah fitur di vs code silahkan

jalankan settingnya

nah kita akan buat emmetnya jalan dijavascript, kalian cari setting yang

namanya includelang lalu klik setting disebelahnya lalu pilih copy

setting as JSON kalo sudah kalian klik edit setting in JSON (kalian cari setting JSONnya)

kalian tinggal tambahkan diakhir setingannya itu tanda koma lalu enter lalu

paste yang tadi  "emmet.includeLanguages": {}

nah didalam kurung kurawalnya kalian tulis begini

"emmet.includeLanguages": {

    "javascript" : "html"

}

nah ini bacanya ketika filenya javascript jalankan fitur htmlnya kita save dulu nah sekarang didalam template literalnya saya tulis div.mhs itu langsung jalan

const mhs = {

    nama : 'M.Muizzuddin',

    umur : 18,

    nrp : 45232,

    email : 'muizzuddin332@gmail.com'

}

const el = `<div class="mhs">

    <h2>${mhs.nama}</h2>

    <span class="nrp">${mhs.nrp}</span>

</div>`

console.log(el)

jadi gampang ya, nah ceritannya ini tinggal kalian simpan misalnya ke DOM

menggunakan innerHTML atau kalian mau consolenanti akan ada white space

atau ruang kosong

ya jadi mungkin itu beberapa contoh penggunaan dari template literal

dan untuk contoh contoh lain seperti tagged template tadi itu akan kita bahas dimateri selanjutnya

oke jadi itu saja ya materinya mudah mudahan kalian paham dan ini akan

sering sekali dipakai ketika kalian menggunakan modern javascript terutama

ketika kalian pakai framework vue react atau angular

jadi itu saja

kita akan ketemu lagi dimateri berikutnya

dan seperti biasa jangan lupa titik koma ;

**12. Javascript lanjutan: latihan template literal**

nah dimateri kali ini kita akan bahas lebih lanjut mengenai html fragments

jadi nanti kita akan lihat bagaimana menggunaan template literal ini untuk

membentuk sebuah element html yang nantinya akan kita simpan di DOM

ya karena nanti didalamnya itu bisa kita simpan banyak hal, kita bisa simpan

pengulangan, pengkondisian, bahkan nanti kita bisa bikin exspressionnya

bersarang jadi untuk lebih jelasnya kita langsung saja coba contoh kasusnya ya

- HTML fragments

sekarang  yang pertama akan kita coba adalah  kita lihat dulu bagaimana cara membuat html fragments ya

misalnya saya punya sebuah object yang didalamnya terdapat beberapa

properti

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     nrp : 83748,

//     email : 'muizzuddin392@gmail.com'

// }

misalnya saya punya sebuah variable untuk menyimpan elementnya yang akandirangkai sebagai html fragments, kita bikin menggunakan template literal

// const el = `<div class="mhs">

//                 <h2>${mhs.nama}</h2>

//                 <span class="nrp">${mhs.nrp}</span>

//             </div>`;

nah misalnya kita mau tampilkan didalam DOM jadi kita bisa tulis begini

// document.body.innerHTML = el

kalo saya jalankan didalam DOMnya teradapat elementnya ya dan ini sudah

kita coba sebetulnya dimateri sebelumnya harusnya kalian sudah paham ya

sekarang coba bayangkan gimana kalo misalkan kalian punya object

mahasiswanya isinya banyak

mungkin saja begitu ya, kalian query ke database atau kalian minta data

kesebuah API data mahasisawa dikembalikannya banyak

- looping

nah kita bisa gunakan looping untuk mengulang data yang didapatkan

misalkan saya punya array yang didalamnya terdapat object object

// const mhs = [

//     {

//         nama : 'muiz',

//         email : 'muizzuddin392@gmail.com'

//     },

//     {

//         nama : 'husain',

//         email : 'husain322@gmail.com'

//     },

//     {

//         nama : 'hasan',

//         email : 'hasan292@gmail.com'

//     },

// ]

misalkan datanya ada 3 ya, nah sekarang gimana caranya kita menampilkan

data keDOMnya menggunakan html fragments yang kita looping langsung

didalam template literalnya

// const el = `<div class="mhs">

//                 <ul>

//                     ${mhs.map(a => `<li> ${a.nama}</li>

//                                    <li> ${a.email} </li>`).join('')}

//                 </ul>

//             </div>`;

// document.body.innerHTML = el

nah didalam ul misalnya kita mau looping, gimana cara kita looping? kita bisa

gunakan exspression yang didalamnya kita panggil method map ke array mhs

jadi dia akan mengembalikan array yang baru lalu kita join

karena seperti yang kalian ketahui didalam kurung kurawalnya kita bisa isi

expressionnya, dan map adalah expression yang isi mapnya kita bisa

kembalikan li yang didalamnya ada nama dan email didalam objectnya

kalo terdapat tanda koma kita tambahkan join untuk mengubahnya jadi

string, itu kalo kalian mau melakukan looping

- pengkondisian

nah sekarang gimana kalo mau melakukan pengkondisian

sebetulnya kita bisa pakai ternary ya

contohnya kalian punya daftar lagu ya ada judul lagu dan ada penyanyinya

// const lagu = {

//     judul : 'tetap dalam jiwa',

//     penyanyi : 'isyana sarasvati'

// }

kalo misalkan kita mau menampilkan kedalam html fragments ini gampang ya

sama kayak yang diatas ya

// const el = `<div class="lagu">

//                 <ul class="">

//                     <li class="">${lagu.penyanyi}</li>

//                     <li class="">${lagu.judul}</li>

//                 </ul>

//             </div>`

// document.body.innerHTML = el

nah kalo gini kan tidak ada yang aneh ya, tapi problemnya adalah ada lagu

yang punya featuring karena tidak semua lagu itu ada featnya

nah kita bisa kasih kondisi seperti ini

// const lagu = {

//     judul : 'kau adalah',

//     penyanyi : 'isyana sarasvati',

//     feat : 'rayi putra'

// }

// const el = `<div class="lagu">

//                 <ul class="">

//                     <li class="">${lagu.penyanyi}</li>

//                     <li class="">${lagu.judul} ${(lagu.feat) ? `feat ${lagu.feat}` : ''}</li>

//                 </ul>

//             </div>`

// document.body.innerHTML = el

jadi disini saya jalankan expression lagi yang isinya jika ada lagu.feat maka

tambahkan string feat siapa gitu ya dan jika tidak ada maka kita kosongkan

nah kita bisa kasih kondisi seperti ini kalo saya jalankan sudah bener ya

- HTML fragments bersarang (nested)

dan yang terakhir kalo kita mau bikin html fragmentsnya bersarang

misalnya kita punya mahasiswa yang bentuknya object

const mhs = {

    nama : 'muiz',

    semester : 5,

    mataKuliah : ['rekayasa web', 'analisis dan perancarang sistem informasi',

'pemrograman sistem interaktif', 'perancarang sistem berorientasi object']

}

jadi ceritanya ada mata kuliah yang typenya array karena mata kuliah

disemester 5 itu banyak nah gimana cara menampilkan semua data ini

kedalam html fragments

function cetakMataKuliah(mataKuliah){

       return   `<ol>

                    ${mataKuliah.map(a =>`<li>${a}</li>`).join('')}

                </ol>`

}

const el = `<div class="mhs">

                <h2>${mhs.nama}</h2>

                <span class="">semester ${mhs.semester}</span>

                <h4>Mata Kuliah</h4>

                    ${cetakMataKuliah(mhs.mataKuliah)}

            </div>`

document.body.innerHTML = el

dan untuk mata kuliahnya saya akan gunakan ol dan li yang dilooping yang sayakerjakan difunction terpisah yang menerima parameternya adalah array

mataKuliah jadi saya punya function namanya cetakMataKuliah lalu

didalamnya tinggal kitareturn saja sebuah template literal jadi kalian bisa   
lihat sekarang

nama ngambil dari object, semester ngambil dari object juga dan mata

kuliahnya kita looping menggunakan map yang kita simpan kedalam sebuah

function jadi bisa bersarang seperti ini mudah mudahan kalian paham

jadi mungkin itu contoh penggunaan dari HTML fragments yang kita buat

menggunakan template literal dan dimateri selanjutnya kita akan bahas

mengenai template literal ini, kita akan belajar mengenai tagged template

jadi itu saja materinya mudah mudahan tidak bingung

kita ketemu lagi dimateri berikunya

dan seperti biasa jangan lupa titik koma ;

**13. Javascript lanjutan: tagged template**

dimateri kali kita masih akan membahas mengenai template literal dan kali ini

topik yang akan kita pelajari adalah mengenai tagged templates

apa itu tagged template?

tagged template adalah bentuk yang lebih komplex dari template literal yang memungkinkan kita membaca template literals melalui sebuah function

jadi kalo kita sudah membuat template literal, string tersebut bisa kita

masukkan kedalam function untuk mengubah bentuk dari string tadi

kita lihat langsung saja implementasinya misalnya saya punya dua buah

variable

// const nama = 'muiz', umur = 18

sekarang kalo saya ingin membuat template literal biasa

// const str = `Hallo, nama saya ${nama}, saya ${umur} tahun`

// console.log(str)

nah sekarang bagaimana jika kita jadikan template literal ini menjadi

tagged template ?

nah itu caranya tinggal kita tuliskan nama function didepan template literalnya

dan nanti string template literal ini akan seolah olah menjadi argument dari

function tersebut, contohnya gini

// const nama = 'muiz', umur = 18

// function coba(){

//     return 'coba'

// }

// const str = coba`Hallo, nama saya ${nama}, saya ${umur} tahun`

// console.log(str)

misalnya didalam functionnya saya return coba, maka string ini akan masuk

ke parameter functionnya dan functionnya akan mengembalikan coba tapi kita

lihat secara default tagged template ini menerima parameter misalnya

difunctionnya saya kasih parameter string yang mengembalikan isi dari

parameter stringnya

// const nama = 'muiz', umur = 18

// function coba(string){

//     return string

// }

// const str = coba`Hallo, nama saya ${nama}, saya ${umur} tahun`

// console.log(str)

kalo saya return stringnya maka dia akan mengembalikan array yang isinya

string ["Hallo, nama saya ", ", saya ", " tahun"]

maka hasilnya akan ada sebuah array yang berisi 3 element jadi ketika kita

jalankan template literal ini maka tagged templatenya akan memecah tiap tiap string yang ada didalmnya dipisahkan oleh expression, nah expressionnya tidakmasuk kedalam parameter string

jadi itu yang pertama tiba tiba ada parameter string yang bisa kita pakai

nah sekarang expressionnya masuk kemana? dia masuk sebagai parameter

berikutnya

// const nama = 'muiz', umur = 18

// function coba(string, nama, umur){

//     return nama

// }

// const str = coba`Hallo, nama saya ${nama}, saya ${umur} tahun`

// console.log(str)

jadi kalo saya tulis returnnya nama atau umur maka akan muncul tuh isi dari

parameternya , nama isinya atau muiz umur isinya 18, ya ini karena kita punya

dua buah expression, ada kemungkinan nanti ketika kita membangun sebuah

string itu kita tidak tahu ada berapa expression didalam tempalte literalnya

jadi misalnya kita punya 3 buah expression dan kemudian kita tangkap satu

satu jadi nanti kita akan punya 3 buah parameter untuk menampung setiap

nilainya, itu kalo 3 expression, kalo 4 gimana kalo 5 atau bahkan 10 gimana?

nah tha’s the problem

nah javascript modern itu punya sebuah parameter yang namanya rest

parameter, jadi dia bisa menampung seluruh expression template literalnya

cara nulisnya tinggal kalian tulis titik 3 kali lalu nama parameternya apa

// const nama = 'muiz', umur = 18

// function coba(string, ...values){

//     return values

// }

// const str = coba`Hallo, nama saya ${nama}, saya ${umur} tahun`

// console.log(str)

nah ini nantinya values akan menjadi sebuah array yang isinya semua

exspression yang ada didalam template literal jadi kalian tidak perlu ambil satu

satu lagi

jadi kita ambil semua argument selain array yang didalamnya berupa string

karena itu akan masuk keparameter string nah sisanya kan itu adalah

expression yang kita bisa tangkap satu satu

nah dari pada kita tangkap satu satu mending kita tangkap semuanya

menggunakan rest parameter maka nanti dia akan mengembalikan array yang isinya sisa dari argumentnya

kalo saya return values maka yang tampil adalah array yang berisi semua

expression yang ada didalam template literalnya jadi parameter string

buat mengambil string biasa values buat mengambill expressionnya

paham ya

nah kedepannya kalo misalkan kita mau menampilkan string yang sama

artinya ktia harus gabungkan string dengan valuesnya untuk menampilkan

string yang persis sama, kita coba

// const nama = 'muiz', umur = 18

// function coba(string, ...values){

//     return string.map((value, i) => `${value}${values[i]}`).join('')

// }

// const str = coba`Hallo, nama saya ${nama}, saya ${umur} tahun`

// console.log(str)

nah jadi kita pakai map saja ya, untuk tiap tiap value pada string saya

kembalikan valuenya dan saya gabungkan dengan values index ke i

jadi misalnya value dari string itu hallo, nama saya nah itu kan dia index ke 0

dan i nya juga akan berisi 0

maka saya kembalikan dengan menggabungkan hallo, nama saya dengan

values[i] yang isinya adalah muiz lalu looping lagi dan seterusnya lalu kita

join deh kalo kita jalankan maka muncul tuh hasilnya

Hallo, nama saya muiz, saya 18 tahunundefined

nah ini kenapa ada undefinednya karena string itu memiliki index 0 sampai 2

sedangkan values itu memiliki index 0 smapai 1

maka string index 2 saya gabungkan dengan values index 2

karena values index ke 2 itu tidak ada maka isinya adalah undefined

cara mengatasinya kita kasih pengkondisian saja pakai ternary ya

atau kita bisa pakai operator logika OR

// const nama = 'muiz', umur = 18

// function coba(string, ...values){

//     return string.map((value, i) => `${value}${(values[i] || '')}`).join('')

// }

// const str = coba`Hallo, nama saya ${nama}, saya ${umur} tahun`

// console.log(str)

jadi ini artinya kalo values index ke i ada isinya maka tampilin isinya

kalo values index ke i tidak ada isinya maka tampilkan string kosong

kalo ini kita jalanin hasilnya  Hallo, nama saya muiz, saya 18 tahun

tuh udah bener ya

jadi string literal ini dijalankan kedalam function coba menghasilkan string yangsama mungkin sampai sini kalian merasa agak aneh ya

ngapain kita membuat tagged template untuk mengasilkan string yang sama ?

nah nanti kita lihat manfaatnya berikutnya ya

sebelum kesitu selain pakai map kita bisa juga pakai higher order function

reduce, reduce itu bisa menggabungkan semua isi dari array

// const nama = 'muiz', umur = 18

// function coba(string, ...values){

//     return string.reduce((acc, curr, i) => `${acc}${curr}${(values[i] || '')}`, '')

// }

// const str = coba`Hallo, nama saya ${nama}, saya ${umur} tahun`

// console.log(str)

kita langsung return panggil arraynya string lalu kita reduce

nah reduce itu kalo kalian ingat memiliki 2 parameter accumulator sama

currentvalue

nah sebenarnya ada parameter selanjutanya ya itu indexnya kita butuh index

untuk melooping valuesnya

lalu kita gabungkan acc, curr dan values index ke i dan terakhir kita kasih nilai

awal dari currnya kalo tidak kita tulis nilainya adalah 0 karena ini adalah string maka saya kasih string kosong '' jadi ini kalo kita jalanin hasilnya akan sama  ya

nah sekarang pertanyaannya tadi ngapain kita capek capek lakukan ini kalo

hasilnya sama persis ? nah sekarang bayangkan contoh implementasi

sederhananya kalo misalkan kalian ingin menampilkan sebuah string, hasil

pencarian berdasarkan keyword tertentu

misalkan kalian ingin mencari berdasarkan nama nah kalian ingin tampil hasil

pencariannya dengan nama yang kalian masukkan itu sudah ada highlightnya

misalnya tampil hallo nama saya muiz tapi muiznya ada background colornya

sehingga orang kalo lihat dimudahkan, stringnya tampil beserta highlightnya

// const nama = 'muiz', umur = 18

// function highlight(string, ...values){

//     return string.reduce((acc, curr, i) => `${acc}${curr}<span class="hl">

${(values[i] || '')}</span>`, '')

// }

// const str = highlight`Hallo, nama saya ${nama}, saya ${umur} tahun`

// console.log(str)

sekarang gini, setelah ditampilkan saya ingin nama dan umurnya itu ada

highlightnya nah nama dan umur ini dia adalah values ya, jadi valuesnya

sebelum saya tampilkan saya bungkus dulu menggunakan span yang nama

classnya hl misalnya, kita jalankan ya

nah maka setiap expressionya akan dibungkus dengan span, walapun diakhir

ada spannya ya tidak apa apa karena ini tidak akan kelihatan

jadi kalo sekarang saya punya didalam htmlnya itu sebuah style yang ada

selectornya hl lalu saya kasih background color misalnya salmon

const nama = 'muiz', umur = 18

function highlight(string, ...values){

    return string.reduce((acc, curr, i) => `${acc}${curr}<span class="hl">

${(values[i] || '')}</span>`, '')

}

const str = highlight`Hallo, nama saya ${nama}, saya ${umur} tahun`

document.body.innerHTML = str

terus saya ingin tampilkan didalam bodynya, maka kalo saya jalankan

nah lihat tampil stringnya yang expressionnya sudah ada higlightnya

dan ini kerennya kalo misalkan saya punya expression lagi maka dia akan

menambahkan higlight baru

keren ya ini adalah penggunaan sederhana dari tagged template gitu ya

dan kalo misalkan kalian nanya penggunaan lain dari tagged template ini ada

beberapa ya

1.Escaping html tags

    untuk menghindari script atau karakter yang tidak diinginkan pada html

menggunakan framework

2.translation dan internationalization

    alih bahasa dari website kalian (menggunakan framework)

3.styled components

    dan yang terakhir yang saat ini sering digunakan untuk front end framework   
 itu untuk styled components

    ya misalkan kalian pakai veu atau react

ya jadi mungkin itu saja penjelasan mengenai tagged template ini

mudah mudahan kalian paham ya

memang cukup unik ya silahkan kalian coba coba sendiri saja

ya jadi itu saja materinya

kita ketemu lagi dimateri berikutnya

dan satu lagi jangan lupa titik koma ;

**14. Javascript lanjutan: desturcturing assignment**

kali ini kita akan membahas mengenai destructuring variable atau

destructuring assignment jadi fitur ini adalah fitur yang unik dimodern

javascript ini ya

apa itu destructuring assignment?

*“destructuring assignment adalah expression pada javascript yang membuat*

*kita bisa membongkar value dari array atau properti dari object kedalam*

*sebuah variable terpisah”*

nah contoh sederhananya itu kalian bisa lihat seperti ini

misalnya kita punya sebuah variable yang dia adalah array

// const coba = ['satu', 'dua', 'tiga']

nah ini bisa kita buat tiap tiap nilainya menjadi nilai pada variable terpisah

kalo sebelumnya kan gampang saja ya kita bikin variable a yang kita isi dengan coba index ke 0 bikin lagi b kita isi coba index ke 1 dan seterusnya

nah tapi dengan menggunakan destructuring assignment ini kita cukup tulis

deklarasi variablenya lalu tulis kurung siku buka dan tutup

// const [a,b,c] = coba

// console.log(a) // hasilnya satu

// console.log(b) // hasilnya dua

// console.log(c) // hasilnya tiga

lalu didalamnya kita isi dengan nama nama variable yang ingin kita gunakan

lalu kita assignment dengan array coba tadi sehingga secara otomatis isi dari

expression destruction ini akan dimasukkan tiap tiap nilai dari arraynya

misalnya a index ke 0 diisi dengan satu index ke 0 juga dan seterusnya

sehingga kalian tidak perlu assignment satu persatu lagi jadi kalo saya console

a muncul satu, b muncul dua c muncul tiga jadi harus urut, keren ya

nah begitu pula dengan object

misalkan kita punya object mhs yang punya beberapa properti

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     jurusan : 'teknik informatika'

// }

nah kalo kita mau simpan tiap tiap propertinya kedalam sebuah variable

terpisah, kan biasanya kita bikin variable nama diisi dengan mhs.nama dan

seterusnya kalo sekarang tidak usah, kita tinggal bikin sintax destructuringnya

kurung kurawal

jadi kalo kurung kurawal untuk object dan kurung siku untuk array

// const {nama, umur, jurusan} = mhs

// console.log(nama) // hasilnya muiz

// console.log(umur) // hasilnya 18

// console.log(jurusan) // hasilnya teknik informatika

saya punya nama, umur, jurusan yang diassignment keobject mhs langsung

secara otomatis menjadi variable terpisah yang isinya sudah mengambil dari

objectnya, jadi agak unik nih, sederhananya seperti ini

kalo kita mau lihat lebih detil lagi kita coba  penggunaan sederhanya

A. distructuring assignment pada array

misalnya saya punya variable perkenalan yang isinya adalah array string

// const perkenalan = ['halo', 'nama', 'saya ', 'muiz'];

nah sebelum menggunakan destructuring saya mau mengambil tiap tiap nilai

pada array sebagai variable terpisah, caranya kan gampang saja ya

// const salam = perkenalan[0],

// nama = perkenalan[3]

// console.log(salam)

// console.log(nama)

kalo saya munculkan diconsole salam maka hasilnya halo begitu pula kalo

saya muculkan nama diconsole ya ini tidak ada masalah kalo kita menggunakanteknik biasa

sekarang kalo kita mau pakai destruturing kita cukup bikin variable yang

bentuknya array untuk membongkar array yang didalamnya terdapat nama

nama variable yang namanya bisa bebas

// const perkenalan = ['halo', 'nama', 'saya ', 'muiz'];

// const [salam,satu, dua, nama] = perkenalan

// console.log(salam)

yang diassignment kedalam array perkenalan jadi nama variable index

ke 0 diisi dengan value dari array perkenalan index ke 0 juga begitu pula

seterusnya jadi harus urut ya untuk array ini dan saya tulis console.log salam

maka akan muncul halo console log dua maka muncul nama dan

seterusnya gitu ya

nah sekarang kita juga bisa melompati element element pada array

misalnya saya tidak mau memasukkan semua kedalam variable

saya ingin salam dan nama saja, satu dan dua tidak usah

1.skipping values

nah kita bisa skip caranya kita tinggal hapus satu dan duanya saja komanya

jangan

// const perkenalan = ['halo', 'nama', 'saya ', 'muiz'];

// const [salam, , , nama] = perkenalan

// console.log(nama)

jadi salam skip skip nama dan sekarang string nama dan string saya tidak

masuk kedalam variable kalo saya tulis nama muncul ya muiz

2.swap values (menukar isi dari array)

nah selajutnya kita bisa melakuakn swap values dan ini sudah pernah kita coba

dimateri materi sebelumnya ya jadi kalo saya punya dua variable seperti ini

// let a = 1;

// let b = 2;

// [a, b] = [b, a];

// console.log(a); // 2

// console.log(b); // 1

nah kita langsung saja gunakan [a,b] yang diassignment dengan [b,a] maka

kalo saya console harusnya hasilnya 2 dan 1, gitu ya jadi bisa saja sintax

destructuring ini untuk menukar isi pada variable

nah kita juga bisa gunakan retrun value pada function

3.return value pada function

jadi functionnya bisa mengembalikan array dan arraynya langsung ditangkap

oleh sintax destructuring itu contohnya begini

jadi saya punya function coba yang mengembalikan array 1 dan 2

// function coba(){

//     return [1,2]

// }

// const [a,b] = coba()

// console.log(a)

// console.log(b)

nah dengan menggunakan destructuring ini kita bisa langsung masukkan

1 dimaukkan kedalam variable a dan 2 dimasukkan kedalam variable b

4. default value

nah selanjutanya kita bisa atur nilai default pada sebuah variable

// const angka = [5,4]

// let [a = 1, b = 2, c = 3] = angka

// console.log(a)

// console.log(b)

// console.log(c)

jadi misalkan yang pertama a saya isi dengan 1 dan b saya isi dengan 2 dan c

saya isi dengan 3 lalu saya assignment dengan array angka yang isinya 5 dan 4

maka yang terjadi adalah a diisi dengan 5 dan b diisi dengan 4 karena diarray

angka itu hanya memiliki 2 value maka untuk yang c nilainya adalah 3 karena

3 adalah nilai default dari c kecuali kalo array angka itu memiliki 3 value maka c akan berisi value ke 3 dari array angka

5.rest parameter

dan yang terakhir kalo misalkan kalian mau membongkarnya banyak nilai arraynya tapi nantinya kalian tidak tahu ada berapa element nah disini kalian bisa gunakan seperti dimateri sebelumnya, kita bisa gunakan rest parameter

misalkan gini kalo saya punya variable a dan b ya

// const [a,b] = [1,2]

// console.log(a)

kalo begini tidak ada masalah ya tapi gimana kalo ada tiga nilai pada arraynya

kalo empat gimana kalo 10 gimana ?nah kita bisa menangkap nilai sisanya

menggunakan rest parameter

// const [a, ...values] = [1,2,3,4,5]

// console.log(values)

jadi saya hanya menangkap 1 value yaitu 1 dimasukkan ke a dan sisanya masukkedalam values yang mengembalikan array jadi nilai 2 3 4 5 akan menjadi [2,3,4,5]

gitu ya temen temen untuk array, nah sekarang kita masuk keobject

B.destructuring object

misalkan kita punya object namanya mhs

// const mhs = {

//     nama : 'muiz',

//     umur : 18

// }

nah sekarang gimana caranya kita mau menangkap tiap tiap propertinya

sebagai variable nah untuk object cukup gunakan kurung kurawal buka

dan tutup lalu tuliskan nama variablenya sesuai dengan nama propertinya

jadi ini harus sama

// const {nama, umur} = mhs

// console.log(nama)

kita tulis nama dan umur lalu diisi dengan mhs

kalo saya console nama maka akan muncul muiz jadi untuk nama variablenya

tidak boleh sembarang ya harus sesuai dengan nama propertinya

1. assignment tanpa declaration

nah dengan menggunakan object kita tidak perlu melakukan declaration dulu

contohnya seperti ini

// ({nama, umur} = {

//     nama : 'muiz',

//     umur : 18

// })

// console.log(nama)

dan kita tidak perlu menggunakan tanda const tapi syaratnya kita harus kasih

kurung buka dan kurung tutup disini dan ini hasilnya sama ya sekarang

gimana caranya kita mau ngasih nama baru yang tidak sesuai dengan

nama propertinya

2. assignment kevariable baru

nah kalo mau beda caranya kalian cukup tambahkan titik dua setelah nama

variablenya lalu spasi nama barunya

// const mhs = {

//     nama : 'muiz',

//     umur : 18

// }

// const {nama: n, umur: u} = mhs

// console.log(u)

jadi sekarang kalo mau memanggil nama jangan nama tapi n

nah selanjutnya kita juga bisa memberikan default value

3.memberikan default value

misalnya saya punya object yang sederhana

// const mhs = {

//     nama : 'muiz',

//     umur : 18

// }

// const {nama, umur, email} = mhs

// console.log(email)

misalkan saya tambahkan variable baru yaitu email kalo saya console.log

menurut kalian akan muncul apa? karena kita check didalam objectnya tidak

ada emailnya kalo ada langsung tampil kalo tidak ada isinya undefined

nah kita bisa kasih nilai disini sebagai nilai default

// const mhs = {

//     nama : 'muiz',

//     umur : 18

// }

// const {nama, umur, email = 'muizzuddin332@gmail.com'} = mhs

// console.log(email)

misalkan saya isi defaultnya untuk email yaitu muizzuddin332@gmail.com

ya jadi dia ngecheck dulu ada email gak? kalo tidak ada dia akan pakai

defaultnya tapi kalo misalkan kalian kasih didalam objectnya email maka yang

akan tampil adalah isi email didalam objectnya

4.memberikan nilai default dan juga assignment kevariable baru

ini juga bisa digabung kita bisa memberikan nilai default dan juga assignment

kevariable baru

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin222@gamil.com'

// }

// const {nama: n, umur: u, email: e = 'muizzuddin332@gmail.com'} = mhs

// console.log(e)

5.rest parameter

dan kalian juga bisa menggunakan rest parameter sama sepreti array

tetapi kalo didalam object dia akan mengembalikan object

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin222@gamil.com'

// }

// const {nama: n, ...value} = mhs

// console.log(value)

jadi nama masuk kedalam variable nama dan sisanya masuk kedalam value

dan dia akan menjadi object kalo saya console value maka isinya adalah object

{umur: 18, email: "muizzuddin222@gamil.com"}

6.mengambil suatu properti tertentu

misalkan saya punya object yang sama contohnya seperti ini jika kalian ingin

mengambil suatu properti tertentu kalian tinggal tuliskan sesuai dengan nama propertinya, misalkan saya hanya ingin mengambil properti email

// const mhs = {

//     id : 123,

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin222@gamil.com'

// }

// const {email} = mhs

// console.log(email)

jadi saya tinggal tuliskan email kalo saya jalankan akan muncul emailnya ya

7.mengambil fild pada object setelah dikirim sebagai parameter untuk function

nah maksutnya gimana kita lihat ya contohnya saya punya object yang sama

const mhs = {

    id : 123,

    nama : 'muiz',

    umur : 18,

    email : 'muizzuddin222@gamil.com'

}

function getid({id}){

    return id

}

console.log(getid(mhs))

jadi kita bikin function untuk mengambil id dari mhs ya dengan menggunakan destructuring ini kita bisa simpan parameternya itu tidak object melainkan

hanya satu nilai tertentu

jadi misalnya saya kirimkan parameter mhs lalu ditangkap diparameternya

hanya yang idnya saja jadi bisa sepeti ini kalo menggunakan destructuring

assignment pada object

oke jadi itu ya beberapa contoh  penggunaan dari destructuring assignment

mudah mudahan kalian paham dengan penjelasannya dan dimateri

selanjutnya kita akan implementasikan pengetahuan kita tadi kedalam study

kasus yang lebih real lagi

jadi mungkin itu saja untuk materi kali ini

kita akan ketemu lagi dimateri berikutnya

dan seperti biasa jangan lupa titik koma ;

**15.Javascript lanjutan: destructring function**

kali ini kita akan lanjutakan kembali materi destructuring ini kita akan lihat

bagaimana destructring ini berpengaruh pada function

langsung saja kita lihat contoh kasus dari code berikut ini

misalnya saya punya function sederhana yang namanya penjumlahanPerkalian

dimana saya nanti punya dua buah parameter yang isinya dia akan

mengembalikan penjumlahan dari dua buah nilai dan juga perkaliannya dan

nilainya saya kembalikan dalam bentuk array

// function penjumlahanPerkalian(a,b){

//     return [a + b, a \* b];

// }

nah kalo tiap tiap nilainya mau saya masukkan kedalam variable saya bisa

lakukan seperti ini

// const jumlah = penjumlahanPerkalian(2,3)

// console.log(jumlah)

kalo saya console log jumlah maka isinya akan ada array, 5 itu penjumlahan

dan 6 itu perkalian kalo saya mau mengambil penjumlahannya saja maka saya

bisa kasih indexnya

// function penjumlahanPerkalian(a,b){

//     return [a + b, a \* b];

// }

// const jumlah = penjumlahanPerkalian(2,3)[0]

// console.log(jumlah)

pada saat dikembalikan langsung saja saya ambil index ke 0 karena mau

mengambil yang 5 kalo saya jalankan maka hasilnya 5 begitu pula dengan

perkalian ya agak ribet kita harus menambakan index di akhir

nah dengan menggunakan destructuring kita bisa langsung saja membuat

sebuah array yang isinya variable kita bongkar nih arraynya

// function penjumlahanPerkalian(a,b){

//     return [a + b, a \* b];

// }

// const [jumlah, kali] = penjumlahanPerkalian(2,3)

// console.log(jumlah) // 5

// console.log(kali) // 6

kita tahu isi arraynya ada 2 kita bikin jumlah sama kali yang dia

mendistructuring return valuenya gitu ya, kita juga bisa mendistructring

return value dari function kalo saya console muncul ya hasilnya 5 dan 6

sip ya jadi kita bisa lakukan ini

nah sekarang gimana kalo misalnya return valuenya nambah

jadi tidak hanya ada tambah dan kali saja saya akan masukkan a - b dan a / b

// function cal(a,b){

//     return [a + b, a \* b, a - b, a / b];

// }

// const [jumlah, kali, kurang, bagi] = cal(2,3)

// console.log(bagi)

nah kalo begini gimana? sama saja ya kita tinggal bikin distructringnya lebih

lengkap lagi, kalo saya console bagi hasilnya 0.6 ya

sekarang gimana kalo yang bagi tidak ada? kalo yang bagi didalam arraynya saya hapus begitu saya panggil bagi maka nilainya undefined

ya kalo kalian tidak ingin munculnya undefined kalian bisa kasih nilai default ya

// function cal(a,b){

//     return [a + b, a \* b, a - b,];

// }

// const [jumlah, kali, kurang, bagi = 'tidak ada'] = cal(2,3)

// console.log(bagi)

misalkan nilai defaultnya saya kasih tidak ada kalo saya jalankan maka hasilnya

tidak ada ya tapi ketika direturn valuenya ada array bagi maka dia akan

mengambil nilai yang ada didalam return value baginya

gitu ya jadi kita bisa kasih nilai default juga

nah sekarang probelmnya kalo misalkan kita mereturnnya array adalah ketika

kita melakukan destructuring itu urutannya harus pas harus bener tidak boleh sampai tertukar

misalnya returnnya array index ke 0 yang isinya penjumlahan nah lalu

nama variable index ke 0nya juga harus tambah tidak boleh tertukar ya

jadi untuk array urutan sangat berpengaruh

nah kalo misalkan kalian ingin urutan tidak berpengaruh return valuenya bisa

kita ganti dengan object

// function cal(a,b){

//     return {

//         jumlah : a + b,

//         kurang : a - b,

//         kali : a \* b,

//         bagi : a / b

//     }

// }

// const {bagi, jumlah, kurang, kali} = cal(2,3)

// console.log(kurang)

kita bikin properti untuk yang didalam objectnya ya kalo kita mau distructring

object kita pakai kurang kurawal sekarang ini urutannya tidak berpengaruh

kalian mau bagi dulu kurang jumlah dan kali tidak akan berpengaruh

karena dia mengacu kepada nama properti yang sama dengan nama variablenya jadi urutannya tidak berpengaruh

kalo saya console log kurang, kurang kan diurutan ke 3 ya didalam variablenya

padahal kalo dipropertinya dia urutan ke 2, tapi disini untuk urutan tidak akan terpengaruh kalo saya jalankan hasilnya harusnya -1 ya

jadi kalo kalian ingin urutannya tidak berpengaruh bisa gunakan object

distructuring pada returnnya

nah sekarang kita akan lihat gimana kita melakukan distructur pada argument

contohnya sepert ini, misalnya saya punya object mhs

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin332@gmail.com'

// }

nah ceritanya saya mau bikin sebuah function untuk mencetak string

berdasarkan properti mhs ini untuk emailnya saya biarkan dulu

// function cetak(nama, umur){

//     return `hallo nama saya ${nama}, saya umur ${umur} tahun`

// }

// console.log(cetak(mhs.nama, mhs.umur))

nah jadi kalo saya ingin mengambil properi nama saya bisa lakukan seperti itu

kalo saya console maka hasilnya // hallo nama saya muiz, saya umur 18 tahun

ya ini pakai function declaration biasa objectnya kita pecah sendiri

atau kalo mau cara yang lain kalian bisa kirimin objectnya diargumentnya

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin332@gmail.com'

// }

// function cetak(mhs){

//     return `hallo nama saya ${mhs.nama}, saya umur ${mhs.umur} tahun`

// }

// console.log(cetak(mhs))

jadi kita pecah propertinya didalam functionnya, hasilnya sama saja ya

sekarang kalo kalian mau pakai destructuring supaya dia hanya mengambil

nama dan umurnya saja kalian bisa kirimkannya object lalu kita bisa lakukan   
distructuring didalam parameternya atau ini hamper sama seperti assignment tanpa declaration

jadi pada saat object dikirim diparameternya kita langsung pecah

saya ambil properti nama dan umurnya saja seperti ini

// const mhs = {

//         nama : 'muiz',

//         umur : 18,

//         email : 'muizzuddin332@gmail.com'

//     }

// function cetak({nama, umur}){

//     return `hallo nama saya ${nama}, saya umur ${umur} tahun`

// }

// console.log(cetak(mhs))

jadi object dikirim lalu dipecah didalam parameternya diambil nama dan umurnya saja sehingga didalam functionnya kita bisa tulis nama dan umur saja

kalo kita jalankan hasilnya sama ya

ini untuk memudahkan penulisan kalo misalnya objectnya komplex

nah ini akan kepakek banget ketika objectnya komplex misalnya

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin332@gmail.com',

//     nilai : {

//         tugas : 80,

//         uts : 85,

//         uas : 75

//     }

// }

ceritannya kalian punya properti nilai yang nilai ini ceritanya isinya object jadi

ada object didalam object dan ini bisa lebih komplex lagi ya ini hanya contoh

saja misalkan didalam nilai ini saya punya 3 buah properti dan sekarang saya

mau menambahkan nilai return valuenya

// function cetak({nama, umur, nilai}){

// return `hallo nama saya ${nama}, saya umur ${umur} tahun dan nilai uas

saya adalah ${nilai.uas}`

// }

// console.log(cetak(mhs))

ini harusnya saya tambahkan diparameter nilainya lalu saya tangkap nilai.uas

kalo saya jalanin aman ya

nah sekarang gimana kalo nilainya kita destructuring lagi jadi ini seperti

destructuring bersarang, jadi caranya begini

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin332@gmail.com',

//     nilai : {

//         tugas : 80,

//         uts : 85,

//         uas : 75

//     }

// }

// function cetak({nama, umur, nilai: {tugas,uts, uas}}){

//     return `hallo nama saya ${nama}, saya umur ${umur} tahun dan nilai uas

saya adalah ${uas}`

// }

// console.log(cetak(mhs))

kalian bisa tambahkan disini titik dua lalu tuliskan kurung kurawal lagi isi dari

nilainya jadi kalian bisa tulis tugas, uts, uas atau kalian bisa ambil satu nilai saja yang dibutuhkan yaitu uas jadi didalam kurung kurawalnya kalian bisa tuliskan uas saja itu sama saja

lalu kita bisa langsung tulis saja uas direturnnya

kalo ini kita jalankan sama saja ya

begitu ya jadi ini caranya kalo kalian mau destructuring parameter

mudah mudahan sampai disini kalian paham

ya jadi itu saja untuk materi kali ini kita belajar mengenai distructuring function

karena ini cukup penting dan akan sering dipakai ketika kalian pakai front end

framework seperti react veu dan yang lainnya

jadi itu saja untuk materi kali ini

mudah mudahan kalian paham

kita akan ketemu lagi dimateri berikutnya

dan seperti biasa jangan lupa titik koma ;

**16. Javascript lanjutan:  for of vs for in**

kali ini kita akan mempelajari mengenai dua looping baru yang ada pada

modern pada javascript, sintax looping baru ini dinamakan for..of dan for..in

nah sebelumnya kita tahu kita punya beberapa statement looping ya

kita bisa gunakan for, while kita bisa gunakan forEach untuk melooping pada

array

sekarang kita akan lihat dulu satu satu bagaimana cara kerja dari

for of dan for in ini, yang pertama kita akan bahas mengenai for of

kalo kita lihat defisini dari web developer mozila

*for of is creates a loop iterating over iterable object*

jadi for of ini adalah sebuah looping yang bisa mengulang atau menelusuri

object object yang iterable, lalu apa saja object iterable ini?

-string

-array

-arguments / nodeList

-typedArray

-map

-set

-User defined iterable

ada banyak ya, tapi dimateri kali ini kita hanya akan bahas mengenai string

array dan object yang mirip array yaitu arguments / nodeList

jadi ini adalah yang bisa diloopig oleh for of, nanti kita akan lihat apa

bedanya dengan for in

jadi untuk lebih jelasnya kita langsung lihat saja implementasinya seperti apa

1. for of

kita coba yang for of dulu

- pada array

misalnya saya punya array mhs yang isinya nama nama mahasiswa

// const mhs = ['muiz', 'husain', 'hasan'];

ceritanya saya ingin menampilkan kedalam console nama namanya saja

inikan banyak caranya kalian bisa gunakan for biasa seperti ini

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

//     console.log(mhs[i])

// }

ini pakai for biasa, kalo kalian jalankan akan muncul hasilnya

nah kita juga bisa pakai forEach seperti ini

// const mhs = ['muiz', 'husain', 'hasan'];

// mhs.forEach(nilai => console.log(nilai))

jauh lebih simple dari pada kita pakai for kalo kalian jalankan hasilnya sama

persis, sekarang karena array ini merupakan salah satu dari iterable object

maka kita bisa gunakan for of ya

jadi cara menulisnya begini

// const mhs = ['muiz', 'husain', 'hasan'];

// for(const n of mhs){

//     console.log(n)

// }

kalian tinggal tulis for lalu didalamnya kita representasikan tiap tiap value padaarraynya sebagai variable baru apa misalkan const n lalu of dan nama arraynya jadi cukup menulis seperti ini saja

tiap tiap element kira representasikan sebagai n dari array mhs sehingga

didalamnya kalian bisa tulis console.log n kalo kalian jalankan hasilnya sama

jadi gitu ya ada beberapa cara untuk looping isi dari array

disini saya tidak akan membahas mengenai bagus mana cepet mana atau

optimal mana disini saya hanya kasih tau saja beragam cara untuk

melooping array dan salah satunya karena array ini adalah iterable maka pasti bisa pakai for of

-pada string

string merupakan iterable ini adalah perbedaan dengan menggunakan forEach

forEach tidak bisa digunakan pada string karena dia bukan array

contohnya saya punya variable nama yang isinya muiz

// const nama = 'muiz'

// for(const n of nama){

//     console.log(n)

// }

kalo saya gunakan for of nanti variable n ini akan berisi m u i dan z

kalo saya jalankan maka dia akan melooping tiap tiap karakter pada stringnya

gitu ya

- pada array dan mengambil indexnya

nah sekarang kita gunakan forEach untuk melooping tiap tiap nilai pada array

dan kita akan mengambil indexnya

// const mhs = ['muiz', 'husain', 'hasan'];

// mhs.forEach((a, i) => console.log(`${a} adalah mahasiswa ke ${i + 1}`))

nah forEach punya parameter ke 2 untuk indexnya kalo saya jalankan hasilnya

muiz adalah mahasiswa ke 1

husain adalah mahasiswa ke 2

hasan adalah mahasiswa ke 3

kalo pakai forEach bisa, tapi kalo pakai for of secara default dia tidak punya

index kalo kalian pengen tetep  pakai for of kalian bisa tambahkan method

yang namanya enteries

// const mhs = ['muiz', 'husain', 'hasan'];

// for(const n of mhs.entries()){

//     console.log(n)

// }

kalo kalian lihat sekarang n itu isinya adalah array

[0, "muiz"]

[1, "husain"]

[2, "hasan"]

ada index dan ada namanya jadi kalo kalian mau langsung dapatin dua duanya kalian bisa langsung destructur disini

// const mhs = ['muiz', 'husain', 'hasan'];

// for(const [i, n] of mhs.entries()){

//     console.log(`${n} adalah mahasiswa ke ${i + 1}`)

// }

satu variable sebagai index dan satu variable sebagai namanya kalo saya

jalankan hasilnya sama ya dengan forEach

- nodeList

selanjutnya object iterable adalah nodeList

nodeList ini ketika kita melakukan querySelectorAll pada DOM

coba kalian bikin list yang isinya nama nama mahasiswa dan kalian tangkap tuhsemua li nya

// const li = document.querySelectorAll('ul li');

// console.log(li)

maka kalo saya console.log linya maka hasilnya

NodeList(3) [li, li, li]

linya akan berbentuk nodeList dan nodeList itu bukan array walaupun

kelihatannya sama seperti array ada indexnya dan ada valuenya

kalo kalian menggunakan browser baru untuk melooping linya itu kalian bisa

gunakan forEach walaupun sebetulnya forEach itu khusus untuk array

tapi kalo kalian menggunakan browser yang lama itu untuk menggunakan

forEach kalian harus paksa nodeListnya jadi array murni

ya bisa menggunakan Array.form() tapi kita tidak akan menggunakan

Array.form()  sekarang kita akan gunakan for of

jadi saya bisa tulis begini

// const li = document.querySelectorAll('ul li');

// for(n of li){

//     console.log(n.innerHTML /\* atau kalian bisa gunakan textContent \*/)

// }

ya kalo kalian tidak deklarasikan variablenya javascript otomatis akan

mendeklarasikan variable seperti ini

var n = undefined

jadi dia adalah variable yang dibuat menggunakan keyword var

kalo kalian menggunakan for of ini langsung bisa karena nodeList itu adalah

iterable gitu ya

-arguments

selanjutnya kita juga bisa gunakan untuk arguments

contohnya gini saya punya function yang namanya jumlankanAngka

nanti function ini akan mengembalikan penjumlahan dari argument

// function jumlahkanAngka(){

//     return arguments

// }

// console.log(jumlahkanAngka(1,2,3,4,5))

kalo saya kirimkan keparameter harusnya diparameter itu ada variable untuk menangkap nilainya tapi kita tidak akan gunakan parameter untuk menangkap nilai yang dikirimkan

sekarang kita tangkap menggunakan arguments, secara default kalo kita tidak

tangkap menggunakan parameter maka nilainya akan ketampung diarguments

kalo kita lihat isi dari argumentsnya bentuknya seperti array padahal dia

arguments bukan array

nah sekarang gimana caranya saya ingin menjumlahkan semua nilainya ?

sebetulnya kalo array itu gampang banget kita tinggal reduce saja

// function jumlahkanAngka(){

//     return arguments.reduce((acc, curr) => acc + curr)

// }

// console.log(jumlahkanAngka(1,2,3,4,5))

karena ini bukan array kalo kita paksa menggunakan reduce ya akan eror

begitu pula jika kita pakai forEach dia akan eror

kita bisa ubah arguments menjadi array atau kita langsung saja menggunakan

for of untuk melooping object iterable, jadi saya bisa tulis begini

// function jumlahkanAngka(){

//     let h = 0

//     for(const n of arguments){

//         h += n

//     }

//     return h

// }

// console.log(jumlahkanAngka(1,2,3,4,5))

jadi kita bisa pakai for of untuk melooping isi dari arguments kalo kita jalankan maka hasilnya jadi 15

mudah mudahan sampai disini kalian paham ya

beberapa cara untuk melakukan looping pada iterable object

2. for in

nah kalo tadi kita sudah bahas mengenai for of sekarang kita akan bahas

mengenai for in

*for in is creates a loop only iterating over enumerable*

jadi ini hanya untuk enumerable, enumerable disini adalah properti pada

object kalo tadikan iterable itu dia tidak punya properti

nah untuk object itu kita bisa looping propertinya mengunakan for in

contohnya begini misalkan saya punya variable mhs yang bentuknya object

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin332@gmail.com'

// }

kalo kita mau melooping isinya kita tidak bisa pakai for of kalo kalian tetap

memaksa pakai for of dia akan eror

// for(a of mhs){

//     console.log(a)

// }

Uncaught TypeError: mhs is not iterable

katanya si object ini bukan termasuk bagian dari iterable object

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin332@gmail.com'

// }

// for(a in mhs){

//     console.log(a)

// }

nah kalo kita pakai for in baru jalan kalo saya jalankan maka akan

muncul propertinya, dia akan melooping enumerablenya

nama umur dan email

jadi kalo kalian mau loopingnya properti dari object kalian pakai for in

kalo kalian mau mengulang isi array dan iterable object lainnya kalian bisa

gunakan for of dan terakhir kalo kalian ingin mengulang isinya bukan

propertinya itu kalian bisa tulis begini

// const mhs = {

//     nama : 'muiz',

//     umur : 18,

//     email : 'muizzuddin332@gmail.com'

// }

// for(a in mhs){

//     console.log(mhs[a])

// }

jadi saya tulis mhs index ke a maka yang diambil sekarang valuenya

kalo saya jalankan dia akan muncul valuenya

begitu ya mudah mudahan sampai disini kalian paham

jadi mungkin itu penjelasan mengenai for of dan for in

yang merupakan looping yang baru ya dijavascript modern

kita akan lanjutkan materi javascript lanjutan ini dimateri selanjutanya

dan seperti biasa jangan lupa titik koma ;

17. Javascript lanjutan: spread operator

kali ini kita akan membahas mengenai sesuatu yang dinamakan dengan spread operator dan rest parameter

apa itu spread operator dan rest parameter ini? kedua hal ini direpresentasikan dengan tanda titik tiga kali ...

jadi meskipun notasinya sama tapi kedua hal ini bebeda dan sebetulnya kita

sudah pernah mencoba menggunakan notasi 3 titik ini  dibeberapa materi

sebelumnya tapi nanti kita akan lihat lebih detil lagi maksut dari spread

operator dan rest parameter ini

kita mulai dimateri ini dengan topik yang pertama yaitu spread operator

jadi kita lihat dulu ya definisinya, apa itu spread operator?

*spread operator adalah sebuah operator yang memecah (exspand / unpack)*

*iterable menjadi single element*

jadi yang tadinya iterable , yang tadinya banyak isinya dipecah menjadi single

element jadi kalo kalian inget iterable itu ada banyak  check aja dimateri

sebelumnya

bagaimana cara memecahnya? kita langsung saja lihat beberapa contoh kasus berikut ini , misalnya saya punya variable mhs yang isinya adalah array

// const mhs = ['muiz', 'husain', 'hasan'];

inget bahwa array itu kita bisa destructor, kalo kita destructur tiap tiap

elementnya itu langsung masuk kedalam variable terpisah tapi kalo

menggunakan spread operator, misalnya gini saya console mhs dan saya

tambahkan ... didepannya

// console.log(...mhs)

maka kalo yang tadinya array itu dipecah pecah menjadi tiap tiap elementnya

kalo saya jalankan hasilnya muiz husain hasan

sekarang kalian lihat bedanya ketika saya masuk kedalam arraynya

misalkan saya ambil index ke 0

// const mhs = ['muiz', 'husain', 'hasan'];

// console.log(...mhs[0])

index ke 0 itu string dan string itu merupakan iterable object sehingga kalo

string kita pecah pakai spread operator hasilnya begini  m u i z

jadi dia akan dipecah  masing masing menjadi huruf terpisah mudah

mudahan kalian paham lalu kapan kita menggunakan spread operator ini

- mengubah menjadi array pada object iterable

jadi kalo misalkan saya punya variable yang isinya string

// const str = 'muiz'

// const arr = [...str]

// console.log(arr)

nah kalo saya tulis  didalam kurung sikunya ...str maka tiap tiap nilai pada str

akan dipisah menjadi m u i z dan tiap tiap nilai tersebut akan masuk kedalam

array arr sehingga kalo saya tulis console.log arr maka hasilnya adalah

array dari tiap tiap huruf pada string str

- menggabungkan dua array

kalo misalkan saya punya array namanya mhs lalu saya punya juga array yang

namanya dosen misalnya

// const mhs = ['muiz', 'husain', 'hasan'];

// const dosen = ['sandhika', 'dody']

// const gabung = [...mhs, ...dosen]

// console.log(gabung)

saya bikin variable gabung dan gimana cara gabungnya? caranya kalian cukup

tuliskan kurung siku buka dan tutup dan didalamnya kita isi dengan spread mhslalu koma dan spread dosen jadi untuk tiap tiap nilai pada mhs dia akan

masuk menjadi array lalu digabung dengan tiap tiap nilai yang ada pada dosen masuk menjadi array

jadi nanti array gabung ini isini nya muiz husain hasan digabung dengan

sandhika dody sehingga kalo saya console log gabung maka isinya adalah

gabungan dari dua array tadi

nah sebetulnya kita juga bisa pakai cara yang lain

dengan menggunakan method array yang namanya concate, kalo pakai

concate kalian bisa tulis begini

// const mhs = ['muiz', 'husain', 'hasan'];

// const dosen = ['sandhika', 'dody']

// const gabung = mhs.concat(dosen)

// console.log(gabung)

nah ini hasilnya sama dengan spread operator, jadi kalo saya jalanin hasilnya

sama tapi dengan menggunakan spread operator itu kita bisa lebih flexible lagi

jika kita ingin menambahkan elementnya ditengah tengah mhs dan dosen

misalnya saya mau gabungin mhs dan dosen tapi ditengah tengahnya saya maukasih value array lain mislakan saya mau tambahkan aji

// const mhs = ['muiz', 'husain', 'hasan'];

// const dosen = ['sandhika', 'dody']

// const gabung = [...mhs, 'aji',...dosen]

// console.log(gabung)

jadi tinggal kita tambahkan ditengah tengah saja kalo pakai concate tidak

semudah itu ya jadi lebih flexible kalo kita pakai spread operator

paham ya jadi kita bisa menggabungkan dua buah array atau lebih

-melakukan copy nilai pada array

selanjutanya kita bisa melakukan copy nilai pada array contohnya saya

punya array lagi mhs ceritanya saya mau bikin array baru yang isinya

sama dengan mhs misalkan caranya begini

// const mhs = ['muiz', 'husain', 'hasan'];

// const mhs1 = mhs

// console.log(gabung)

ketika saya bikin variable mhs1 yang diisi dengan mhs harapannya kan isinya

sama dengan mhs kalo kita jalankan hasilnya sama tapi problemnya kalo

melakukan ini sebenarnya dia tidak benar benar melakukan copy kenapa?

contohnya begini sebelum saya console mhs1 kita ganti mhs1[0] dengan

sandhika

// const mhs = ['muiz', 'husain', 'hasan'];

// const mhs1 = mhs

// mhs1[0] = 'sandhika'

// console.log(mhs1)

nah kan harusnya yang berubah itu hanya array mhs1 index ke 0 jadi kalo saya

console mhs1 isinya ["sandhika ", "husain", "hasan"] tapi kalo saya console

mhs isinya ["sandhika ", "husain", "hasan"] jadi mhs ikut berubah padahal

yang kita ingin ubah hanya array mhs1

jadi kalian tidak bisa mengcopy array seperti ini karena kalo kalian melakukan

ini artinya membuat reverensinya jadi tetap saja mhs1 itu nunjuknya ke array mhs jadi dia tidak benar benar mengcopy jadi caranya kalo ingin mengcopy

kalian harus menggunakan spread operator seperti ini

// const mhs = ['muiz', 'husain', 'hasan'];

// const mhs1 = [...mhs]

// mhs1[0] = 'sanhika'

// console.log(mhs1)

jadi tiap tiap single element pada array mhs kita jadikan array baru kita

bungkus dengan kurung kusiku buka dan tutup jadi kalo saya console mhs1

isinya ['sandhika', 'husain', 'hasan'];

kalo saya console mhs isinya ['muiz', 'husain', 'hasan'];

jadi array awalnya tetap jadi spread operator bisa juga digunakan untuk

mengcopy array, gitu ya mudah mudahan sampai disini paham

- contoh kasus menggunakan html 1

ceritanya didalam bodynya kalian bikin list yang isinya nama nama mahasiswa

    <ul>

        <li>muiz</li>

        <li>hasan</li>

        <li>husain</li>

    </ul>

sekarang saya ingin mengambil nama namanya menggunakan DOM lalu saya

simpan menggunakan array jadi saya ingin punya array yang isinya muiz,hasan,husain', kalo mau pakai cara looping biasa kalian bisa gini

// const liNama = document.querySelectorAll('ul li')

// const arr = []

// for(const li of liNama){

//     arr.push(li.textContent)

// }

// console.log(arr)

karena ini iterable kalian bisa gunakan for of kalo saya jalankan hasilnya sudah bener ya ["muiz", "hasan", "husain"]

nah sebenarnya ini kita bisa menggunakan higher order function map tapi

kalo pakai map itu kita harus ubah dia menjadi array murni karena ini iterable

object kalian bisa menggunakan spread operator

// const liNama = document.querySelectorAll('ul li')

// const arr = [...liNama].map(value => value.innerHTML)

// console.log(arr)

jadi tiap tiap nilai pada nodeList liNama kita spread lalu kita masukkan kedalam array setelah kita ubah menjadi array murni baru kita bisa menggunakan map

kalo saya jalankan hasilnya seperti ini ["muiz", "hasan", "husain"]

jadi hasilnya sama

- contoh kasus html 2

misalnya didalam htmlnya saya punya h1 yang isinya muiz dan kalian bisa

kasih css sedikit ya sesuai keinginan kalian saja

ceritanya kalo saya hover tiap tiap hurufnya maka akan ada aksi tertentu

misalnya hurufnya menjadi lebih besar kita tangkap dulu h1nya

const h1 = document.querySelector('.nama')

const value = [...h1.textContent].map(n => `<span>${n}</span>`).join('')

console.log(value)

lalu tiap tiap hurufnya saya sisipkan tag span sehingga nanti spannya saya bisa

kasih cssketika saya hover hurufnya akan menjadi besar jadi setelah saya

tangkap h1nya saya bikin variable yang isi dari h1 lalu kita spread  ambil content h1nya

jadi value akan menjadi string dan string itu iterable kita bisa pecah single

elemetnya dan kita masukkan kedalam array lalu kita map kembalikan tiap tiaphurufnya yang dibungkus dengan tag span dan kita join kalo saya console

muncul ya tiap tiap hurufnya akan dibungkus dengan span

<span>M</span><span>u</span><span>i</span><span>z</span>

tinggal kita masukkan value kedalam h1nya

h1.innerHTML = value

dan kita bisa tambahkan css dispannya jangan lupa span itu displaynya inline

kita kasih displaynya inline-block agar animasinya jalan kalo kita jalankan

tiap tiap hurufnya akan tertransform ketika saya hover

oke jadi seperti itu contoh penggunaan dari spread operator dimana kita bisa memisahkan iterable object menjadi single elements mudah mudahan

kalian paham

baik jadi itu untuk materi kali ini pembahasan mengenai spread operator

dan dimateri berikutnya kita akan bahas rest parameter

jadi sampai disini dulu materinya

kita ketemu lagi dimateri berikutnya

dan seperti biasa jangan lupa titik koma ;

**18. Javascript lanjutan: rest parameter**

kali ini kita akan membahas mengenai rest parameter, notasinya sama titik

tiga tapi kali ini kita gunakan untuk hal yang berbeda dan sebetulnya kita sudahpernah mencoba ini pada saat kita membahas mengenai destructuring

untuk lebih jelasnya kita lihat  dulu definisi dari rest parameter ini

*rest parameter adalah parameter yang merepresentasikan argument pada function dengan jumlah yang tidak terbatas menjadi sebuah array*

ya kan kalian tahu didalam function itu bisa kita simpan argument yang

argumentnya kita bisa tangkap menggunakan variable tapi dengan

menggunakan rest parameter  kita bisa tangkap banyak argument sampai tida

terbatas untuk menjadi sebuah array

kita langsung saja lihat contohnya

misalnya saya punya function yang namanya myfunc

// function myfunc(a,b){

//     return a

// }

// console.log(myfunc(1,2,3,4,5))

biasanya kalo kita kirimkan argument seperti ini kita bisa tangkap satu persatu argumentnya menggunakan sebuah variable didalam parameternya maka kalo saya tulis seperti ini, kalo saya returnya a maka munculnya 1 kalo b

munculnya 2

jadi yang diambil adalah nilai pertama dan nilai keduanya sisanya akan

diabaikan, sekarang kalo kita kasih parameter selanjutnya itu dengan

rest parameter jadi saya bisa tulis begini

// function myfunc(a,b, ...myarr){

//     return myarr

// }

// console.log(myfunc(1,2,3,4,5))

jadi saya tulis titik tiga dan nama variablenya misalnya myarr

maka nanti saya bisa ambil sisanya dari parameter yang sudah kita tangkap

jadi kalo saya return myarr maka munculnya adalah [3, 4, 5]

kalo saya ingin memunculkan semua nilai yang ada didalam parameternya sayabisa tulis begini

// function myfunc(a,b, ...myarr){

//     return `a = ${a} b = ${b} myarr = ${myarr}`

// }

// console.log(myfunc(1,2,3,4,5))

jadi kalo saya jalankan munculnya a = 1 b = 2 myarr = 3,4,5

1 ditangkap oleh a 2 ditangkap oleh b dan sisanya ditangkap dan dijadikan

array oleh myarr atau saya mau tangkap semuanya juga bisa seperti ini

// function myfunc(...myarr){

//     return myarr

// }

// console.log(myfunc(1,2,3,4,5))

jadi dia akan berisi semua argumentnya menjadi sebuah array kalo saya

jalankan hasilnya [1, 2, 3, 4, 5]

nah rest parameter ini hanya bisa kita gunakan diakhir dari parameternya

jadi setelah rest parameter saya tidak bisa menambahkan sebuah parameter

lagi dan rest parameter harus berada diakhir dari parameter mudah mudahan

paham penggunaan dasar dari rest parameter

kalo kalian inget sebetulnya didalam function itu kita punya sebuah variable

yang namanya arguments, arguments ini berisi semua argument yang kita

kirim pada saat function dipanggil, jadi kalo saya return arguments

// function myfunc(){

//     return arguments

// }

// console.log(myfunc(1,2,3,4,5))

jadi kalo saya jalankan hasilnya seperti ini

Arguments(5) [1, 2, 3, 4, 5, callee: ƒ, Symbol(Symbol.iterator): ƒ]

kalian lihat arguments ini bukan array

jadi karena dia bukan array kita tidak bisa pakai method method yang ada padaarray kalo yang tadi menggunakan rest parameter saja itu dia akan berbentuk

array murni jadi kita bisa gunakan method method  pada array

nah kalo kalian tetap ingin pakai arguments bisa juga tapi kita harus ubah

arguments ini menjadi array murni kita bisa gunakan Array.from()

// function myfunc(){

//     return Array.from(arguments)

// }

// console.log(myfunc(1,2,3,4,5))

kalo kita jalankan dia akan menjadi array murni atau kita bisa gunakan spread operator

// function myfunc(){

//     return [...arguments]

// }

// console.log(myfunc(1,2,3,4,5))

kalo saya jalankan hasilnya akan sama

sekarang kita coba kasus berikutnya yaitu menjumlahkan seluruh nilai yang adadidalam parameternya misalnya saya punya function namanya jumlahkan

// function jumlahkan(...angka){

//     let nilai = 0

//     for(const n of angka){

//         nilai += n

//     }

//     return nilai

// }

// console.log(jumlahkan(1,2,3,4,5))

karena di iterable object kita bisa pakai for of, kalo saya return hasilnya 15

jadi dengan menggunakan rest parameter bisa sepert itu atau kita mau

ringkas pengulangannya juga bisa

kita bisa pakai higher order function reduce jadi kalian bisa tulis begini

// function jumlahkan(...angka){

//     return angka.reduce((acc,curr) => acc + curr)

// }

// console.log(jumlahkan(1,2,3,4,5))

kalo saya jalanin hasilnya 15, jadi simple kita bisa manfaatkan rest parameter

untuk kasus seperti ini

selanjutnya kita bisa gunakan rest parameter pada saat kita melakukan

arry destructuring misalnya gini saya punya sebuah array kalompok

// const kelompok1 = ['muiz', 'hasan', 'husain', 'doddy', 'wahyu']

misalnya kalian mau atur setiap nama yang muncul pertama dia adalam ketua

setiap nama yang muncul kedua dia adalah waklik ketua dan sisanya adalah

anggota kalian bisa tulis seperti ini

// const [ketua, wakil, ...anggota] = kelompok1

// console.log(ketua)

kalo saya console ketua munculnya muiz kalo wakil munculnya hasan dan

anggota munculnya husain doddy wahyu jadi rest parameter bisa digunakan

pada destucturing assignment atau kita juga bisa menggunakan pada object

distructuring juga

misalnya saya punya object literal namanya team

jadi ini adalah team lengkap untuk project web developmen kalian ya

// const team = {

//     projectManager = 'husain',

//     frontEnd1 : 'Muiz',

//     frontEnd2 : 'wahyu',

//     backEnd : 'hasan',

//     ux : 'riyan',

//     devOps : 'raia'

// }

misalnya seperti ini dan ini adalah team lengkapnya

nah kalo kalian mau pisahkan menggunakan object distructuring

misalkan kalian mau pisahkan antara project manager dan sisanya

kalian bisa tulis seperti ini

// const {projectManager, ...myteam} = team

// console.log(projectManager)

// console.log(myteam)

kalo saya console project manager munculnya husain

kalo myteam munculnya object sisanya jadi bisa juga digunakan untuk object

distructuring seperti ini

dan yang terakhir contohnya kalo kalian mau melakukan filtering pada type

data pada parameter yang dikirim misalnya begini

misalkan saya punya function namanya filterBy

ceritanya saya mau memfilter nilai nilai berdasarkan type datanya

contohnya kalo saya mau mencari angka pada nilai nilai berikut ini

// function filterBy(){

// }

// console.log(filterBy('number', 1,2,'muiz',false,10,true,'husain'))

nah saya mau memilih mana saja dari parameter ini yang bentuknya number

atau nanti kalo saya mau cari string numbernya saya ganti dengan string

nah berarti saya bisa gini, saya akan bagi menjadi dua parameter yang

pertama typenya mau apa sisanya nilainya yang mau difilter jadi disini saya

punya dua buah parameter yang pertama type dan rest parameter value

misalnya

// function filterBy(type, ...values){

//     return values.filter(n => typeof n === type)

// }

// console.log(filterBy('number', 1,2,'muiz',false,10,true,'husain'))

jadi kita bisa pakai filter lalu kita check typeofnya untuk tiap tiap nilainya

kalo typenya adalah number  jadi saya tulis type lalu kembalikan

jadi dia akan mengecheck 1 number bukan? kalo iya return dan seterusnya

kalo kita jalanin tampilnya hanya ada tiga [1,2,10]

kalo typenya saya ganti string maka munculnya ['muiz','husain']

begitu pula dengan type data yang lain

begitu ya teman tema jadi kita bisa gunakan juga untuk kasus yang seperti ini

mudah mudahan sampai disini kalian paham

baik mungkin itu ya penjelasan mengenai rest parameter pada materi kita kali

ini mudah mudahan memberi gambaran antara spread operator dan rest

parameter

jadi sampai disini dulu materinya

kita akan  ketemu lagi dimateri berikutnya

dan seperti biasa jangan lupa titik koma ;