**JS LANJUTAN 2.1 Execution Context, Hoisting & Scope**

Kita akan membahas topik mengenai Closures tapi sebelum kita masuk ke topik nya kita perlu pengantarnya dulu, pengantarnya ada 3 buah topik yg disebut dengan, Execution Context, Hoisting dan scope jadi konsep” itu merupakan konsep pada javascript agak aneh untuk sebuah bahasa pemrograma.

Dan ke3 hal itu cukup penting untuk dipelajarin agar kita nanti tidak kebingungan ketika kita masuk ke topik” yg lebih lanjut lagi, dan begitu kalian menyimak penjelasan ini mngkin akan sedikit membingungan kalo bingung bisa pahamin pelan” dan telusuri.

**2 Fase pada Execution Context**

1. Creation
2. Execution

Jadi yg pertama kalian harus ingat di javascript itu ada 2 fase pada execution context, execution context itu saat program kalian dijalankan nanti didalam nya ada yg disebut dengan fase Creation dan ke-2 ada fase Execution.

**Lets Code!**

Tadi saya udah bilang ya kalo javascript itu aneh, contoh nya begini deh tulis console.log(nama); kalo saya tulis gitu hasil yg ada di console nya kira” apa, hasilnya itu eror ya, itu karena variabel nama itu tidak pernah kita definisikan sebelumnya. Nah tapi coba kalo dibawah nya saya tulis begini :

console.log(nama);

var nama = 'Keyla';

Ternyata tampil di consolenya undefined padahalkan sebelumnya kita console.log nama itu dia eror tapi setelah kita tambah variabel nama yg udah di isi string hasilnya malah undefined. Tapi udah jeas lah ya si console.log nya ga munculin isi variabel nama nya karena diduluan kan oleh console.log dulu baru kita isi variabel namanya.

Nah itu sebenernya yg terjadi itu saat program dijalankan yg si console sama var nama nya nah yg terjadi adalah ada sesuatu yg disebut dengan creation phase atau pase kreasi atau pembentukan, pase kreasi ini terjadi di context global, context global itu seluruh isi kode kita isi codingan kita 1file gitu. Terjadi di context global itu karena disini kita langsung didalam file javascript nya tidak didalam function.

Nah nanti misalkan yg didalam function itu beda lagi, kita namakan aja creation phase pada Global Context, jadi pada pase kreasi ini javascriptnya akan mengecek apakah ada variabel atau function didalam codingan nya, itu dulu yg dia cek dia tidak memperdulikan perintah console.log(nama); itu, misalkan kalian punya 10 baris codingan gitu dari 10 baris itu dicari ada kywerd var ga, ada keyword function ga itu dulu yg dia cari, Nah kalo ada dia akan membuat nama variabel apapun nama variabel nya, kebetulan disitu nama variabel nya nama nah itu akan di isi dengan undefined.

Jadi semua Variabel akan diset dulu isinya dengan undefined jika ada keywerd var kalo gaada ya ngga, makannya ketika kita tulis console.log(nama); sebelum var, variabel nya itu udah ada tapi isi nya undefined, lalu kalo ada function itu akan di isi function itu sendiri atau di isi kode function nya jadi maksudnya itu si function nya isi function itu sendiri jadi belum di eksekusi atau belum di jalankan perintah nya karena kita ngecek ada ga si var dan function dulu didalam sebuh codingan javascript gitu.

Nah itu konsep itu disebut dengan hoisting, kalo hoisting kalian terjemahkan kedalam bahasa indonesia itu anggap aja seperti kalian mengerek bendera jadi bendera nya dinaikan ke atas, dan itu juga sama meskipun var nama nya ditulis dibawah atau setelah console.log dia akan otomatis dinaikan ke atas berlaku juga untuk function, jadi yg dilakukan pengecekan keyword function dan variabel ada atau tidak nya nama tersebut atau gini :

*// nama var = undefined*

*// nama function = fn()*

*// itu disebutnya Hoisting*

Yg tadi itu var di isi undefined function di isi kode nya atau isi function itu sendiri. Itu yg terjadi pada execution context di phase kreasi atau creation phase.

Tapi sebetulnya selain 2 hal ini :

*// nama var = undefined*

*// nama function = fn()*

Ada lagi yg didefinisikan yaitu javascript mendefinisikan object window sebagai global object dan javascript mendefiniskan this sebagai object window. :

*// object.window atau window = object global*

*// this = object.window atau window*

Nah itu yg dilakukan meskipun kalian ga nulisin apapun, meski pun file javascript nya itu bener” koson gaada isi nya kita masih bisa ke console terus tulis window dia bakal muncul hasilnya atau object nya begitu pula dengan this tapi selain dari itu gaada.

Jadi ada 2 object yg dibuat dan kalo ada variabel di isi undefined dan kalo ada function di isi tulisan function nya ga dijalananin function nya, nah itu semua dinamakan creation phase. Setelah ini baru execution phase jadi mengeksekusi programnya baris perbaris dari atas kebawah nah baru tu dijalanin console.log nya kalo kita udah ngelewatin creation phase dan hoisting inget ya hoisting itu mengecek keyword variable dan function yg ada.

Begitu dia jalanin console.log(nama); nama itu isi nya apa didalam memori nama isinya undefined setelah itu baru deh eksekusi yg var nama = ‘Keyla’; nah sekarang baru dimemori nama itu isinya Keyla tapi dia udah terlanjur di console.log jadi nya undefined. Kecuali kalo console.log nya itu setelah var nama = ‘Keyla’; baru deh yg di console nanti hasilnya Keyla.

Kita coba contoh yang lain ya

Misalnya kita punya codingan nya gini :

var nama = 'Keyla';

var umur = 16;

function sayHello() {

    console.log(`Halo, nama saya ${nama}, saya ${umur} tahun.`);

}

Ngga tampil apa” karena memang fungsi nya tidak dijalankan, tapi harusnya sekarang kalian udah tau dibelakang layar yg terjadi adalah creation phase dulu lalu kita lakukan hoisting untuk 2 variabel itu dan kita isi function dengan tulisan function nya.

Kalo kita mau cobain mau liat tinggal tulis console.log(sayHello); gausah pake tanda kurung buka karena kalo pake itu artinya manggil si function nya ginikan sayHello() kalo ini gausah karena gamau manggil function nya cukup tulis nama function nya aja. Gini :

ƒ sayHello() {

    console.log(`Halo, nama saya ${nama}, saya ${umur} tahun.`);

}

Tu jadi isi nya yg ditulis kalo variabel mah kan undefined ya, beda dengan kalo kita manggil function itu misalnya gini :

console.log(sayHello());

var nama = 'Keyla';

var umur = 16;

function sayHello() {

    console.log(`Halo, nama saya ${nama}, saya ${umur} tahun.`);

}

Tapi tampil hasilnya di console itu malah gini :

Halo, nama saya undefined, saya undefined tahun.

Nah ini karena tadi ya hoisting itu namanya, kita kan nulis variabelnya dibawah console.log itu begitu. Nah di console itu ada tulisan undefined itu karena harusnya sebuah function itu mengembalikan nilai jadi kita ganti aja jadi return, gini :

var nama = 'Keyla';

var umur = 16;

console.log(sayHello());

function sayHello() {

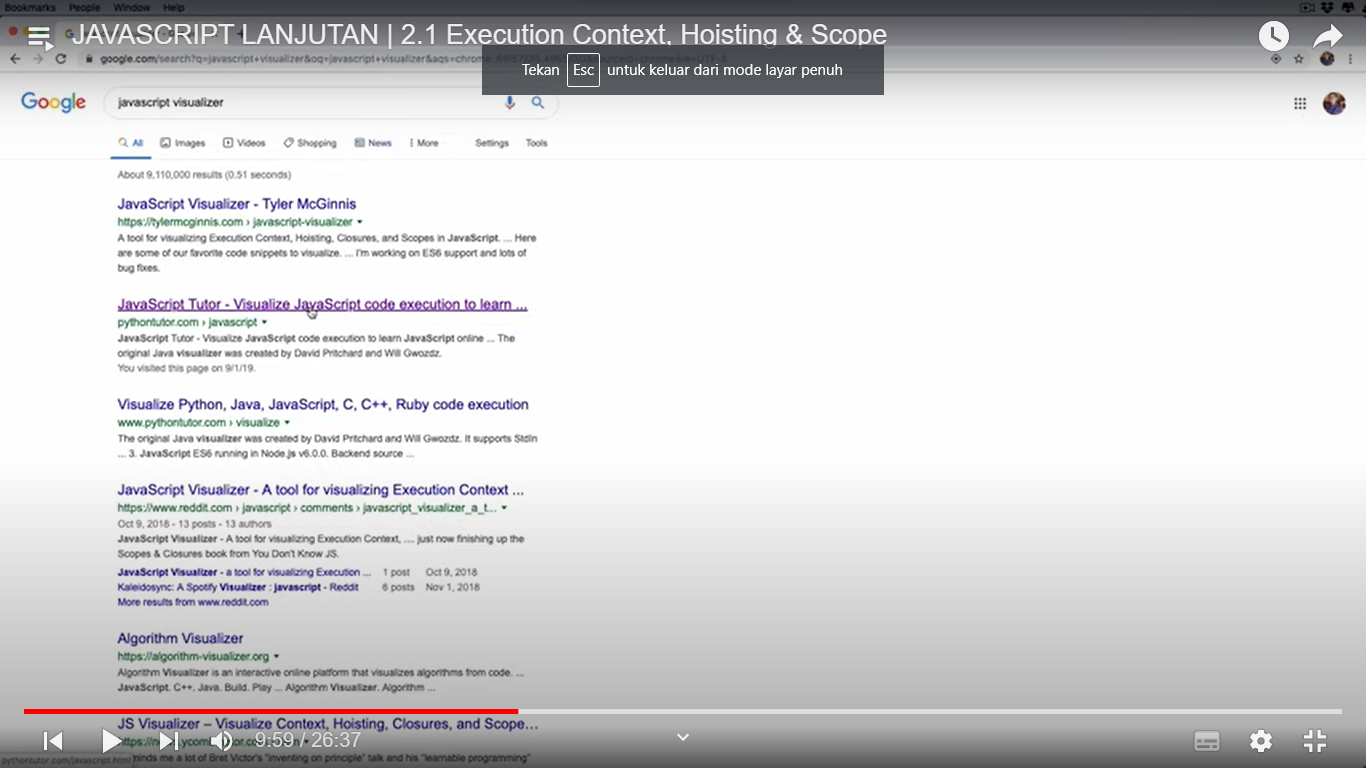
    return `Halo, nama saya ${nama}, saya ${umur} tahun.`;

}

Gitu bukan console.log tapi kita tulis return, baru ilang tulisan undefined nya. Jadi kalo variabel itu di isi dengan udefined kalo function dia hanya di isi dengan isi dari string functionnya, makannya kalo saya tulis atau manggil sebelum function nya ga masalah tapi kalo variabel itu masalah. Jadi itu contoh lain dari Execution Context dan Hoisting.

Nah kalo misalkan kalian mau lihat lebih detail lagi context eksekusi nya itu kalian bisa kunjungi sebuah websaite javascript visualizer, biasanya pak dhika pake link yg udah ungu :

(https://pythontutor.com/visualize.html#mode=display)



Ini untuk mengetahui visualisasi dari bagaimana javascript mengeksekusi kode program kita misalkan kita masukin codingan yg sebelumnya itu ke visualisasinya, klik tombol Visualize Execution di web nya, terus nanti ada visualisasi disebelah kanannya.

Nah ini kita bisa liat step by step gimana javascript menjalankan program nya gitu, disitu ada keterangan panah kalo warna hijau artinya barisnya sudah di eksekusi kalo warna merah akan dieksekusi.

Codingan yg ingin di eksekusi nya :

console.log(sayHello());

var nama = 'Keyla';

var umur = 16;

function sayHello() {

    return `Halo, nama saya ${nama}, saya ${umur} tahun.`;

}

Jadi lah pertama kali javascript ada di console nya mengeksekusi baris satu (yg ada tanda panah warna merah di web nya tapi sebelumnya itu udah terjadi di sebelah kanan ( jalannya program ) :



Jadi ada nama yg isinya undefined, umur isi nya undefined, dan sayHello yg isinya function nya, kalo kita jalanin klik tombol forword, setelah console.log harusnya ke var ya ini dia langsung ngelewatin variabel nya dan ke function nya dulu, karena dia ngehoisting fungsinya dulu.

Kalo sudah bari dia masuk ke lokal function nya juga mengecek function nya, baru dia masuk lagi ke atas baris 1, ini baru phase eksekusi nah abis kebaris 1 baru ke variabel terus kebawah lagi ke function gitu. Jadi visualisasinya loncat” karena ada nya hoisting.

Nah sekarang kita fokus dibagian function nya, tadikan kalian liat di visualizer nya (liat sendiri ke web nya langsung coba sendiri) pada saat masuk menjalankan function itu seolah-olah dia menjalankan execution context nya sendiri jadi kalo ketemu function execution nya udah engga global lagi.

Nah kalo ketemu function dia seolah-olah function membuat local execution context yg didalam nya sama seperti creation phase pada Global Context yang di dalamnya terdapat juga cration dan execution phase, Nah bedanya kalo local execution context itu selain kita bisa akses window kita juga punya akses ke yg namanya arguments, nih ada object argument yang nantinya bisa berisi apapun yg kita masukan di dalam parameter dari function nya dan didalam nya ada hoisting juga, jadi hoisting nya local dia akan ngecek didalam function ada variabel ga kalo ada naikin ke atas, atau didalam function ada function lagi ga nah nanti juga dia melakukan hoisting, dan kalo ada variabel isi dulu dengan undefined pada phase creation nah kalo udah eksekusion baru di isi dengan nilai nya.

Kita coba pake kasus, misalkan saya punya var nama keyla terus saya punya var username ig saya gitu, terus dibawahnya saya punya function, begini :

var nama = 'Keyla';

var username = '@kapzz04';

function cetakURL(*username*) {

    var instagramURL = 'http://instagram.com/';

    return instagramURL + *username*;

}

console.log(cetakURL(username));

harusnya hasilnya itu jadi link menuju instagram saya, keliatan nya ga aneh tapi kalo kita coba telusuri pake virtualize, kalian liat ya sebelum program nya dijalankan didalam memori itu udah ada udah di hoisting, var nama di isi undefined, var username di isi undefined, cetakURL di isi function.

Sekarang kita jalanin klik forward, jadinya nama di isi ‘keyla’, username di isi ‘@kapzz04’, lalu eksekusi function nya di skip langsung ke console.log nya, begitu saya jalanin dia akan memanggil function itu kan si console.log nya ya, Nah itu kita akan masuk kedalam Local Execution context, tuh dia masuk jadinya ada execution context baru ini kita bahas didalam function nya ya.

Username di isi ‘@kapzz04’ itu dapet dari var username diluar function nya ya, lalu var instagramURL di isi undefined awalnya kan gitu ya baru kita jalanin, merah itu belum dijalanin ya tapi itu udah di hoisting duluan.

Kalo dijalanin baru di isi sama string URL nya kita kasih di var instagram, sekarang ke return nya didalam memori nya udah di return tapi belum di cetak, sekarang kalo next sekali lagi atau forward baru dia ke console.log lagi terakhir tampil kelayar.

Jadi intinya setiap function dipanggil maka akan membuat execution context baru yg Local tadi, kalo yg awal itukan untuk global ya yg luar, dan di tambahkan istilahnya itu execution step atau tumpukan eksekusi, dan setelah fungsinya selesai dijalanin kaya tadikan si console.log nya udah menampilkan hasil ke console itu, maka execution context nya akan dihilangkan dari tumpukan nya didalam memori.

Jadi alur nya kalo menurut pendapat saya itu kita hoisting dulu cek semua nama variabel lalu isi dengan undefined dan function nya di isi dengan isi function itu sendiri, setelah itu baru melakukan eksekusi variable nya di isi dengan isi yg sudah kita tuliskan seperti var nama di isi dengan ‘keyla’ gitu, lalu dia langsung ke console.log nya ga ke function nya dulu, pas eksekusi console.log nya ternyata didalam itu ada perintah jalankan function atau memanggil function nya, maka dicari lah nama function tersebut dan melakukan eksekusi yaitu melakukan yg Local Execution Context khusus function nya itu sendiri barulah setelah function dijalankan dia otomatis ilang Local Execution Context nya dari memori baru lah memunculkan hasil console.log nya itu si link menuju instagram saya gitu.

**KASUS BARU LAGI**

Misalnya sekarang saya punya 3 buah function, function a() yg isi nya console.log function b() dan memanggil function b() lalu didalam function b() ada console.log jg function c() dan memanggil function c(), didalam function c() ada console.log aja gini :

function a() {

    console.log('ini a');

    function b() {

        console.log('ini b');

        function c() {

            console.log('ini c');

        }

        c();

    }

    b();

}

a();

Jadi nya ada function didalam function dan memanggil function didalam function wkwkwk. Ini wajib dipahami ini dasar untuk memahami closure.

Keliatan dari codingan nya hasil di consolenya itu berurutan ini a, ini b, dan ini c gitu. Keliatan nya gaada masalah ya, jadi kita akan start dari bawah bagian a() tapi di hoisting dulu, hoisting nya function a() isinya semua isi yg dari function a(), baru kita jalanin.

Maka yg dijalanin mulai dari console.log(‘ini a’) tapi sebelum nya di hoisting dulu karena kan didalam function ada Local execution cotext ya, hoisting yg mana hoisting yg function b() nya kan didalam function ada function dijalanin dulu hoisting nya function b() berisi function itu sendiri, kalo sudah di hoisting jalanin maka tampil kelayar ‘ini a’, kalo sudah langsung loncat ke manggil fungsi b().

Begitu b() dijalanin yg terjadi hoisting lagi, yg di hoisting itu isi dari function itu sendiri ada apa aja, baru muncul ke layar ‘ini b’, terus ke jalanin function c() ada yg di hoisting ga, di cek dulu ada variabel oh gaada ada function ga gaada juga jadi gaada hoisting langsung deh cetak kelayar si console.log di function c() itu.

Nah sampe situ belum selesai sampe situ dinamakan execution step atau tumpukan eksekusi, Nah sekarang sampe sini yg selesai baru c(), kalo c() udah selesai local execution context nya hilang di memori dan dia mereturn undefined si fungsi c() nya.

Lalu sekarang kita ke b() selesai dan terakhir a() selesai, Jadi ini maksudnya ada tumpukan eksekusi lalu kalo sudah beres hilang dari tumpukan eksekusi nya.

Jadi kalo menurut kesimpulan saya si function ini itu dia akan mengeksekusi baris 17 atau memanggil function a() itu karena function a() nya dilewatin dia langsung ke yg manggil si function nya dulu baru cek function nya, terus ada console.log ‘ini a’ itu di tampilkan ke console terus dia lewatin function b() nya dia ke pemanggilan function b() baru masuk ke function b() dan munculkan console.log ke console gtu terus ke manggil function c() baru masuk ke function c() dan menampilkan isi console.log ke layar lalu setelah function c() di eksekusi si function c() mereturn undefined karena disitu emang ga ada return ya, Nah kalo udah direturn terus selesai di eksekusi function c() nya dia menghapus execution context Local function c() nya dari dalam memori, setelah itu dia mereturn function b() sama aja, dan sampai di akhir kembali di baris 17 pemanggilan fuction a(). Jadi intinya setelah melakukan return si function itu lalu keluar dari function nya maka hilang Local execution context nya dari memori begitu.

**BALIK KE KASUS SEBELUM NYA**

var nama = 'Keyla';

var username = '@kapzz04';

function cetakURL(*username*) {

    var instagramURL = 'http://instagram.com/';

    return instagramURL + *username*;

}

console.log(cetakURL('@bebek233'));

Nah yg tampil di console.log nya itu dia berubah jadi @bebek233, padahal variabel username di isi @kapzz04, nah ini bisa terjadi karena ada nya function scope ya, begitu kita nulis argument yg @bebek233 itu lalu data nya dikirim kan ke parameter maka username yg ada didalam function itu ngambilnya ke argument nya makannya tulisan nya jadi @bebek233 itu, Jadi username nya dia ngecek dulu argument pa bukan.

Coba kalo argument nya saya apus tapi saya kirim parameter nya yg @bebek233 itu :

var nama = 'Keyla';

var username = '@kapzz04';

function cetakURL() {

    var instagramURL = 'http://instagram.com/';

    return instagramURL + username;

}

console.log(cetakURL('@bebek233'));

Nah ini kalo kita liat di console hasilnya balik lagi jadi @kapzz04 itu karena si @bebek233 di kirimkan ke argument tapi gaada yg nangkep nih nanti dia masuknya ke object argument, nah tapi di function nya ada tulisan username, kalo kalian tulis variabel username yg dia cek itu variabel Local dulu ada ngga oh ternyata gaada, kalo gaada dia cek ke argument nya ada ga.

Jadi cari dulu ke definisi variabel terdekat nah kalo gaada di dalam function nya dia akan liat ke global atau luar function nya oh ternyata ada makannya dia pake yg di global nya, nah kalo misalkan di global gaada nama variabel tersebut gmna, :

var nama = 'Keyla';

*// var username = '@kapzz04';*

function cetakURL() {

    var instagramURL = 'http://instagram.com/';

    return instagramURL + username;

}

console.log(cetakURL('@bebek233'));

Nah kalo gaada nama variabel tersebut dimana pun baru lah menampilkan atau terjadi nya eror, nah ini terjadi ketika javascript nya udah nyari disemua context ngga ada maka tampilkan referenceError.

Tadi pertanyaan nya @bebek223 itu kemana dia ngilang ga, Nah ternyata dia masuk kesebuah object yg namanya arguments, kan tadi sebelumnya dibilang ya ada object baru pada saat creation dan execution phase di Local execution context, coba kita console.log arguments nya:

var nama = 'Keyla';

var username = '@kapzz04';

function cetakURL() {

    console.log(**arguments**);

    var instagramURL = 'http://instagram.com/';

    return instagramURL + username;

}

console.log(cetakURL('@bebek233'));

Si @bebek233 ini ada di index ke 0 padahal gaada yg nangkep, jadi kalo saya tulis arguments[0] tampil tuh string nya :

var nama = 'Keyla';

var username = '@kapzz04';

function cetakURL() {

    console.log(**arguments**[0]);

    var instagramURL = 'http://instagram.com/';

    return instagramURL + username;

}

console.log(cetakURL('@bebek233'));

Jadi ga ilang si data @bebek233 nya itu, kalo kalian ngirimin nya 2 parameter, maka didalam arguments akan ada 2 data, gini :

var nama = 'Keyla';

var username = '@kapzz04';

function cetakURL() {

    console.log(**arguments**);

    var instagramURL = 'http://instagram.com/';

    return instagramURL + username;

}

console.log(cetakURL('@bebek233', '@erik'));

Jadi parameter tersebut bisa digunakan meskipun ga kalian tangkep menggunakan argument berbentuk variabel function nya itu, oke jadi yg barusan itu adalah function scope.

**LATIHAN**

function satu() {

    var nama = 'Keyla';

    console.log(nama);

}

function dua() {

    console.log(nama);

}

console.log(nama);

var nama = 'Erik';

satu();

dua('Doddy');

console.log(nama);

Ternyata dari codingan tersebut yg tampil dilayar itu ini :

*// Hasil di console.log*

undefined

Keyla

Erik

Erik

Dengan menggunakan konsep yg sudah kita pelajari execution context, hoisting, dan scope, kita punya 2 buah function pasti yg dilakukan adalah creation phase dulu ya hoisting dulu 3 hal itu, kalo udah di hoisting nama di isi undefined.

Nah kalo sudah start eksekusi nya mulai dari console.log(nama) yg setelah function di atas var nama = ‘Erik” itu, 2 function itu kan di skip kan udah di hoisting, di console.log nama berarti undefined karena kita mulai nya dari cosnole.log ini belum ada yg mendefinisikan isi variabel nama tersebut makannya muncul yg pertama hasilnya undefined.

Baru lah masuk ke var nama = ‘Erik’; sekarang didalam memori variabel nama itu si ‘Erik’, baru masuk ke satu(), jalanin function satu() berarti kita hoisting dulu karena kita memasuki area function atau memasuki Local execution context, jadinya variabel nama di isi undefined dulu terus ada function ga di dalam function satu() gaada barulah kita eksekusi, si variabel nama di isi ‘Keyla’ makannya hasilnya Keyla begitu.

Keluar dari function satu() dan execution step nya menghilang kan execution fungsi satu() di memori atau hilang juga Local execution context nya si function satu(), terus ke function dua() dengan mengirimkan parameter ‘Doddy’.

Berarti masuk ke function dua() ya terus parameternya ga ditangkep karena gaada arument disitu nya, terus hoisting ga karena gaada variabel atau function didalam nya berarti gaada hositing, terus itu console.log(nama), berarti yg pertama di cek itu Local dulu kalo gaada ke argument kalo gaada baru global, berarti yg tampil itu si ‘Erik’, udah beres turun lagi udah hilang dari tumpukan step nya.

Dan terakhir ke console.log(nama); si nama nya ini ambil dari variabel global nya berarti si ‘Erik’ lagi tuh. Kalo misalkan masih bingung liat jalannya program di visualizer nya aja ya.

Jadi kesimpulan dari materi ini menurut saya menjalankan Function itu ga harus berurutan dia bisa di skip kalo ga dipanggil function nya bahkan di abaikan function tersebut kalo gaada yg manggil nama nya.