**JAVASCRIPT LANJUTAN 2.2 Closure**

Kita harus paham dulu mengenai apa itu Closure definisinya seperti apa dan konsep nya seperti apa, konsep closure ini bukan hal yg gampang di pahami kita liat ada beberapa definisi nya :

* **Menurut Mozila (MDN)**

*“Closure merupakan kombinasi antara function dan lingkungan leksikal (lexical scope) di dalam function tersebut.”* Nanti lexical scope ini ada kaitan nya dengan topik yg kalian pelajari di video sebelum ini.

* **Menurut w3school**

*“Closure adalah sebuah function ketika memiliki akses ke parent scope-nya, meskipun parent scope-nya sudah selesai dieksekusi.”* Nah ini juga udah kita liat contoh kasus nya di video sebelum nya, jadi kita tau sebuah function bisa memiliki function didalam nya tapi kasus tersebut bukan merupakan sebuah Closure, akan menjadi Closure ketika function didalam nya memiliki akses ke parent scope nya.

* **Code Fellow**

*“Closure adalah sebuah function dikembalikan oleh function yang lain, yang memiliki akses ke lingkungan saat ia diciptakan.”* Nah ini balik lagi ke lexical scope jadi kita bisa bikin function yg mengembalikan atau mereturn function gitu, jadi bukan function didalam function tapi function mereturn function.

* **Techsith**

*“Closure adalah sebuah function yang sebelumnya sudah memiliki data, hasil dari function yang lain.”*

Silahkan dipahami definisinya dengan perlahan karena setiap orang mempunya interprentasi yg beda terhadap Closure.

**LEXICAL SCOPE**

Dari tadi sudah disebut di beberapa definisi, nah misalnya saya punya kode sederhana seperti ini :

function init() {

    let nama = 'Keyla';

    function tampilNama() {

        console.log(nama);

    }

    tampilNama();

}

init();

Kita punya sebuah function yg diluar itu namanya init() lalu didalam nya ada sebuah variabel nama dan ada function lagi yg namanya tampilNama yg didalamnya menjalankan console.log(nama); lalu function tampilNama itu di panggil dan diluar function init() si function init() juga di panggil.

Kasus ini sama hal nya ada function didalam function yg kasus di video sebelum nya yg function a(), b(), dan c() itu sama” memanggil function didalam function bedanya function didalam nya mengambil variabel yg ada didalam function parent nya.

function init() {

    let nama = 'Keyla'; *// local variable*

    function tampilNama() { *// inner function (closure)*

        console.log(nama); *// akses ke parent variabel*

    }

    tampilNama();

}

init();

Nah disini kalian udah tau kita punya sebuah Local variable yg dibuat didalam function init(), nah didalam function init() juga kita punya sebuah function lagi yg kita sebut dengan inner function atau (closure), Nah disini ini beda dengan bahasa pemrograman lain kalo inner function itu bisa memiliki akses ke parent variable nya.

Jadi ketika kita panggil console.log(nama); kalo kalian inget execution context dia nyari dulu ada ga si nama didalam fungsi tampilNama kalo gaada dia cari ke atas ada ga di function init() kalo di function init() gaada dia terus ke atas sampai ketemu global object nya yaitu window nah ini yg disebut dengan Lexical Scope.

Jadi function tampilNama punya akses ke variable nama padahal ada di luar function nya, begitu kasus itu terjadi inner function membutuhkan variabel yg ada di parent scope nya maka inner function ini disebut dengan closure.

**Lets Code CONTOH**

Kita impelemtasikan langsung ke codingan nya ya :

function init() {

    let nama = 'Keyla';

    function tampilNama() {

        console.log(nama);

    }

    tampilNama();

}

init();

Hasilnya akan menampilkan nama ‘Keyla’, jadi yg terjadi adalah ketika init() kita jalankan kalo kita telusuri execution context nya itu hoisting terjadi ya, function init() dipanggil teus variabel nama dibuat, terus function tampilNama() di hoisting juga nah baru variabel nama di isi string ‘Keyla’.

Nah sekarang ketika kita jalankan tampilNama() console.log dijalankan nah dia membutuhkan data nama sedangkan di dalam local function nya gaada klo itu terjadi dia akan naik keatas cari di parent atau pembungkus nya ada ga, kalo saya buat nama di dalam function tampilNama() :

function init() {

    let nama = 'Keyla';

    function tampilNama() {

        let nama = 'Galih';

        console.log(nama);

    }

    tampilNama();

}

init();

Maka yang akan dipake nama yg di local function tersebut hasilnya ‘Galih’, itu jadi nya bukan closure karena tidak menggunakan lexical scope, tapi kalo kode yg sebelum nya itu adalah closure.

Nah untuk memastikan nya ini ada cara nya, sekarang saya gaakan menjalankan function tampilNama() tapi saya akan simpan kedalam console.log, gini :

function init() {

    let nama = 'Keyla';

    function tampilNama() {

        console.log(nama);

    }

    console.log(tampilNama());

}

init();

kalo saya lakukan itu hasilnya sama tapi di console nya ada undefined itu grgr hoisting ya, kalo misalkan saya hapus kurung buka tutup nya gini :

    console.log(tampilNama);

Maka jadinya function nya ga di eksekusi jadi harusnya yg tampil hanya isi function nya saja, ga di eksekusi dia hanya nampilin isi nya aja, nah tapi kalo saya tulis console.dir, gini :

    console.dir(tampilNama);

dia akan menampilkan object nya, jadi object nya bisa kita liat dalem nya kalo di klik nanti kita menemukan scopes, yg kalo saya klik lagi disitu ada keterangan klo ada closure, jadi function init() itu di dalamnya ada closure.

Kenapa ada closure karena membutuhkan variable nama dari parent atau pembungkus nya jadi muncul closure, kalo misalkan saya punya variable lain misal umur, gini :

function init() {

    let nama = 'Keyla';

    let umur = 15;

    function tampilNama() {

        console.log(nama);

    }

    console.dir(tampilNama);

}

init();

Kalo saya jalankan let umur itu ga akan masuk kedalam closure, cek nya di scopes nya ya itu pake console.dir, tapi ketika saya butuh kan let umur untuk fungsi tampilNama() :

function init() {

    let nama = 'Keyla';

    let umur = 15;

    function tampilNama() {

        console.log(nama);

        console.log(umur);

    }

    console.dir(tampilNama);

}

init();

Sekarang fungsi tampilNama() butuh 2 data dari luar makannya itu akan masuk ke closure scope juga, nah liat :

[[Scopes]]: Scopes[2]

0: Closure (init)

nama: "Keyla"

umur: 15

Karena membutuhkan data dari luar maka sekarang ada 2 variable yg masuk kedalam closure scope gitu, mungkin sekarang udah dapet gambaran kenapa disebutnya gabungan antara function dengan lexical cope nya, sekarang kode nya saya kembalikan lagi gini :

function init() {

    let nama = 'Keyla';

    function tampilNama() {

        console.log(nama);

    }

    tampilNama();

}

init();

pada kasus ini function tampilNama sebagai inner function nya itu kan langsung di jalankan yg di bagian manggil fungsi tampilNama(); gitu.

Nah sekarang ini menarik nih, kita bisa lakukan ini kalo saya return tampil nama tapi ga saya jalankan atau ga saya panggil fungsi nya, gini :

function init() {

    let nama = 'Keyla';

    function tampilNama() {

        console.log(nama);

    }

    return tampilNama;

}

init();

Jadi saya mengembalikan fungction tampilNama itu tapi saya tidak menjalankan nya yg artinya tidak memanggil nya berarti ga jalan console.log didalam function nya itu, kalo di liat di console itu ga tampil apa”.

Itu kan saya menjalankan function init(), itu semua isi function init() dijalanin terus didalam nya kita punya function tampilNama() terus kita kembalikan bukan kita jalankan, nah sampai situ istilahnya function nya baru jalan sebagian.

Nah gimana supaya tampilNama() nya jalan, nah itu kitaa bisa simpan ke dalam variabel, misalnya gini :

function init() {

    let nama = 'Keyla';

    function tampilNama() {

        console.log(nama);

    }

    return tampilNama;

}

let panggilNama = init();

Jadi sekarang function init() yg sudah di jalankan sebagian masuk ke variabel panggilNama, baru di bawah nya kita jalankan si panggilNama();, untuk menjalankan tampilNama(), gini :

function init() {

    let nama = 'Keyla';

    function tampilNama() {

        console.log(nama);

    }

    return tampilNama;

}

let panggilNama = init();

panggilNama();

Nah dengan begitu function tampilNama jadi jalan lalu console.log didalam nya pun dijalankan jadi ada hasil nya di console liat aja.

Mungkin ini agak aneh ya, tapi ini memungkin kan kita untuk bisa melakukan yg nanti kita sebut dengan function Factories, contoh nya gini. **Function** Factories

Misalnya saya pengen nama nya itu bukan “keyla” nanti tergantung user masukin nya apa gitu ya, kita kasih parameter di tampilNama(), gini :

function init() {

*// let nama = 'Keyla';*

    function tampilNama(*nama*) {

        console.log(*nama*);

    }

    return tampilNama;

}

let panggilNama = init();

panggilNama();

Jadi dengan logika yg tadi kalo misalkan kita jalanin init() aja itu kan function init() nya jalan lalu di kembalikan tampilNama() tapi belum di jalankan, jadi sekarang kalo saya mau jalankan fungsi panggilNama() saya butuh parameter, gini :

function init() {

*// let nama = 'Keyla';*

    function tampilNama(*nama*) {

        console.log(*nama*);

    }

    return tampilNama;

}

let panggilNama = init();

panggilNama('Icawkwkwk');

Nah jadinya nanti yg tampil itu parameter yg di kirimkan sama function panggilNama(), jadi ini salah satu penggunaan dari closure, dan sebetulnya kalo kalian kembali liat ini :

    function tampilNama(*nama*) {

        console.log(*nama*);

    }

    return tampilNama;

itu kan kita mengembalikan function tampilNama() itu kan, bahkan itu bisa kita ringkas lagi ga perlu pake nama function kita bisa langsung return aja, jadinya anonymous function atau fungsi tanpa nama seperti ini :

function init() {

*// let nama = 'Keyla';*

    return function (*nama*) {

        console.log(*nama*);

    }

}

let panggilNama = init();

panggilNama('Icawkwkwk');

panggilNama('Lala');

panggilNama('uhutt');

Kalo saya jalakan itu sama aja, lalu kalo saya pangiil dengan nama yg berbeda, panggil nya lebih dari satu kek di atas gitu bisa ya. Jadi itu pun salah satu contoh dari closure.

**Kenapa menggunakan Closure?**

Alasan nya itu ada beberapa ya, yg pertama untuk membuat function Factories seperti tadi contoh nya, ke-2 untuk membuat seolah-olah kita punya sebuah private method nanti kita lait masing” dari contoh nya ya.

CONTOH MEMBUAT FUNCTION FACTORIES

Misalkan saya punya function misalnya ucapkanSalam() nah bedanya ini author function nya ya atau function luar nya, function luar nya punya parameter juga kalo contoh kasus sebelum nya init() ga ada parameter ya, kita kasih parameter waktu buat ngasih tau nanti kita lagi di pagi, siang, sore atau malem.

Nah function ini mengembalikan function lagi, inner function nya untuk ngasih tau nama nya siapa, Nah function ini menjalakan console.log gini :

function ucapkanSalam(*waktu*) {

    return function (*nama*) {

        console.log(`halo ${*nama*}, Selamat ${*waktu*}, semoga harimu menyenangkan`);

    }

}

Dengan menggunakan logika yang tadi kita bisa membuat pesan nya itu di tampilkan sesuai waktu, misalnya gini :

function ucapkanSalam(*waktu*) {

    return function (*nama*) {

        console.log(`halo ${*nama*}, Selamat ${*waktu*}, semoga harimu menyenangkan`);

    }

}

let Malam = ucapkanSalam('Malam');

let Sore = ucapkanSalam('Sore');

Malam('Kokoy');

Sore('Rafkuy');

Jadi saya punya variabel yg di isi dengan function author nya atau function luar nya, makannya di situ function ucapkanSalam() dijalan kan dan di isi dengan parameter sesuai waktu gitu, Jadi kita membuat function hasil dari function yg lain.

Nah maksud dari membuat function dari function yg lain itu maksudnya yaitu seperti contoh kasus itu kan sebuah variabel yg di isi function setengah jalan blm jalan sepenuh nya, dan variabel tersebut melanjutkan program function yg setengah jalan itu dengan memanggil nama variabel nya seperti menjalankan layak nya function, jadi nya membuat function dari function yg lain.

Nah lanjut lagi yg tadi kan variabel nya udah di isi sama author function ya, nah itu kalo mau saya jalankan saya tinggal tulis seperti contoh di atas yg menjalankan function itu yg Malam(‘Kokoy’); jadi didalam nya diberi parameter untuk nama itu.

Jadi function Malam() dijalan kan parameter nama ngambil dari yg ‘Kokoy’ itu, dan waktu ngambil dari ucapkanSalam(). Nah kalo misalkan kita coba lihat menggunakan console.dir() isi dari Malam(), gini :

function ucapkanSalam(*waktu*) {

    return function (*nama*) {

        console.log(`halo ${*nama*}, Selamat ${*waktu*}, semoga harimu menyenangkan`);

    }

}

let Malam = ucapkanSalam('Malam');

let Sore = ucapkanSalam('Sore');

console.dir(Malam);

Nah nanti kita bisa lihat bahwa inner function itu sudah jalan setengah nya, isi dari function Malam() itu, kalo kita liat di console nya didalam scopes nya itu udah ada Closure, Bararti itu closure dengan menggunakan lexical scope nya waktu yg isi nya ‘Malam’.

Jadi meskipun function Malam() itu belum kita jalankan tapi function ucapkanSalam() nya sudah berjalan dengan mengisi kan data waktu yg berisi “Malam”, jadi udah jalan setengah nya tinggal nunggu data dari parameter nama nya. Jadi itu disebut dengna Factories function, kita bikin function sesuai dengan function yg lain.

CONTOH MEMBUAT FUNCTION PRIVAT METHOD

Dan tadi alasan yg lain adalah agar kita seolah-olah punya private method yakan atau private variabel gitu, Kita kasih contoh clasic nih misalnya saya punya variabel counter yg isi nya 0, ceritanya saya mau bikin sebuah program sederhana untuk menghitung sudah berapa kali sebuah tombol itu di klik.

Ini saya bikin pake function expression aja ya, gini :

let counter = 0;

let add = function () {

    return ++counter;

}

console.log(add());

Jadi function mereturn jadi ditambah satu dulu baru baru di cetak, jadi kalo di console.log harusnya counter nya jadi 1. Lalu kalo saya duplikat pemanggilan nya harus nya jadi 1,2,3 hasilnya bertambah gitu. :

let counter = 0;

let add = function () {

    return ++counter;

}

console.log(add());

console.log(add());

console.log(add());

Ini ga ada masalah dan ini buka Closure, tapi coba bayangkan kalo program kalian udah panjang puluhan bahkan raturan baris terus ada baris yg mengubah isi counter nya misalkan di baris keberapa gitu, :

let counter = 0;

let add = function () {

    return ++counter;

}

counter = 10;

console.log(add());

console.log(add());

console.log(add());

Nah sekarang counter nya jadi 10 dan karena hoisting counter yg ada didalam function pun jadi terpengaruh, yg harus nya 1,2,3 gitu hasil nya jadi 11,12,13 karena ada nya hoisting, walaupun si counter nya kita isi ulang di baris bawah misalkan di baris ke 100an, dan function nya itu di atas nya, nah ini kita gamau gini.

Sebenernya supaya counter nya itu tidak tergangu sebetulnya kita bisa simpan kedalam gini :

let add = function () {

    let counter = 0;

    return ++counter;

}

counter = 100;

console.log(add());

console.log(add());

console.log(add());

counter yg didalam tidak akan tergangg Cuma sekarang problem nya kalo saya jalankan function add() hasil nya jadi 1 terus terus an karena setiap manggil add() counternya di isi selalu 0 baru di tambah 1 jadinya 1 terus hasilnya.

Nah gimana kalo kita sekarang pake closure, jadi saat kita return yg kita return adalah function, jadi kita punya inner function, lalu kalo saya jalankan fungsi add() tampil nya return an function itu, ini kode nya :

let add = function () {

    let counter = 0;

    return function () {

        ++counter;

    }

}

console.log(add());

console.log(add());

console.log(add());

Nah mengapa hasil console nya itu return an nya itu karena function nya yg jalan baru add(), si inner function nya belum di jalanin baru setengah jalan nya, Nah solusi nya kita harus simpan dulu ke dalam sebuah variabel gitu. :

let add = function () {

    let counter = 0;

    return function () {

      return ++counter;

    }

}

let a = add();

console.log(a());

console.log(a());

console.log(a());

Jadi function add() buat jalanin isi semua nya sedangkan a() hanya untuk jalanin yg bagian return function itu, dan setiap kita jalanin inner function counter akan selalu mengacu ke nilai counter sebelum nya, jadi nya add() itu ke itung dijalanin nya 1x aja jadi awal si counter nya 0, baru yg function a() ngelanjutin jalan nya add() itu jadi nilai nya terus bertambah 1 setiap a() dijalanin gitu, makannya hasil console nya jadi 1,2,3 lagi. Dan jangan lupa selalu return kalo punya function karena bisa aja hasil nya undefined karena ga di return.

Nah jadi seolah-olah counter nya jadi private ga bisa di akses dari luar tapi nilai nya tetap di pertahankan karena dia menjadi closure gitu.

Dan sekarang kalo kita mau kasih atau isi nilai counter nya di baris ke 100an misal nya jadinya itu ga ngaruh sama counter yg lagi berjalan gitu :

let add = function () {

    let counter = 0;

    return function () {

        return ++counter;

    }

}

counter = 100;

let a = add();

console.log(a());

console.log(a());

console.log(a());

Nah tapi kalo kalian liat mungkin agak aneh kita bikin variabel buat nampung function add(), nah sebenernya ada caranya supaya otomatis menjalankan si inner function nya tanpa harus memindahkan ke variabel terus baru dijalankan.

Cara nya kita bisa bungkus dengan immidiately invoked function, jadi cara nya kalian kasih kurung di awal dan akhir function nya yg ini ( ), lalu function itu kita jalankan jadi tambah kan kurung buka kurung tutup di akhiran nya (), gini :

let add = (function () {

    let counter = 0;

    return function () {

        return ++counter;

    }

})();

console.log(add());

console.log(add());

console.log(add());

Jadi pada saat di asaine ke variabel add di langsung menjalankan inner function nya anggap saja karena sebelum nya kan sudah dijalankan terlebih dahulu menggunakan immediately invokad itu.

Jadi function nya itu kaya bayangkan seperti tadi yg let a isi nya function add() nah itu kan manggil function, ini bedanya manggil function nya ga pake nama jadi dia menjejerkan isi program nya bukan nama nya lalu di panggil dengan seperti ini (); bedanya add itu nama function kalo immediately invokad itu pake isi function nya langsung seperti di contoh, semoga kebayang ya.

Jadi kalo saya tambahin pemanggilan nya terus counternya di isi 100 misalkan gini :

let add = (function () {

    let counter = 0;

    return function () {

        return ++counter;

    }

})();

counter = 100;

console.log(add());

console.log(add());

console.log(add());

console.log(add());

console.log(add());

console.log(add());

console.log(add());

Jadi walaupun begitu si counter yg 100 itu ga akan mengganggu function nya ya.