Nama : Ardhito Saputra

NIM : 120140003

Kelas : RA

Link Github :https://github.com/Ardhito120140003/Tugas-2-Individu-PAM

# Rangkuman

## Closure

Sederhananya Closure pada Javascript adalah function di dalam function. Jadi, di dalam function (disebut dengan outer) ada sebuah function (yang disebut dengan inner). Tapi lebih sejatinya lagi closure adalah inner function yang memiliki akses terhadap variable yang dimiliki oleh outer function.

Ada tiga scopes yang bisa dipenuhi pada sebuah closure, yaitu sebagai berikut:

- Dia dapat mengakses apapun yang dimiliki oleh dirinya sendiri.
- Dia dapat mengakses variable yang dimiliki oleh outer function.
- Dia dapat mengakses variable global.

```
//contoh closure
let outer = () => {
    let b= 10
    let inner = () => {
       let a = 20
       console.log(a+b)
    }

    return inner
 }
 console.log(outer()) // first executed
 console.log(outer()) // second executed
```

Keterangan pada code di atas adalah sebagai berikut:

- Function outer memmilki variable b dan mengembalikan inner function.
- Function inner memiliki variable a dan mengakses variable milik outer.

Kita tahu bahwa function outer tersebut dieksekusi melalui console. Dari situ dapat dijelaskan step by step ketika function outer first executed:

- Variable b dibuat dan memiliki value 10 pada function outer.
- Baris berikutnya adalah function expression, tidak ada yang dieksekusi.

- Baris berikutnya adalah dimana function outer mengembalikan function inner. Function inner tidak akan terkeksekusi kecuali jika yang dikembalikan adalah inner() yaitu memiliki kurung buka dan tutup.
- Kemudian kita lihat bahwa function outer dieksekusi. Pada tahap ini jika proses eksekusi selesai maka semua variable yang ada di scope function tersebut not exist.

## Immediately Invoked Function Expression

Immediately Invoked Function Expression (IIFE) adalah bentuk function yang dieksekusi segera setelah function dideklarasikan. Tujuan penggunaan IIFE adalah menghindari Global NameSpace Pollution yang dapat berakibat terjadinya name collisions, yaitu 'tabrakan' antara nama function satu dengan function yang lain atau bahkan nama function dengan nama variable. Contoh Immediately Invoked Function Expression.

```
//Contoh Immediately Invoked Function Expression

//contoh 1
(function () {
    console.log('Hello Bang');
})();

//contoh 2
function sayHi() {
    console.log('Hi Bang');
}
// kita bisa memanggil function di atas dengan
window.sayHi;

//contoh 3
let word = 'Hi Bang';
console.log(window.word); // 'Hi Bang'
```

## First-class function

Dalam berbagai literature dikatakan bahwa Javascript adalah bahasa pemograman yang mendukung first-class function. Sebenarnya apa sih first-class function itu? Dalam ilmu computer, Bahasa pemograman dapat dikatakan mendukung first-class function apabila memberlakukan function sebagai first-class object. First-class object itu sendiri adalah sebuah entitas (bisa berbentuk function, atau variable) yang dapat dioperasikan dengan cara yang sama seperti entitas lain. Operasi tersebut biasanya mencakup passing entitas sebagai argument, return entitas dari sebuah function, dan assign entitas kedalam variable, object atau array. Apabila dikaitkan dengan javascript, intinya function pada javascript adalah first-class object, Yang berarti function tersebut dapat:

- Di assign ke dalam variable, object, atau array
- Di passing sebagai argument pada function lain
- Di return dari sebuah function

```
//contoh First-class function

// 1. Assign ke dalam variable :
let fn = function doSomething() {
    console.log('Hallo')
}
fn() // Hallo

// 2. Assign ke dalam object :
let obj = {
  doSomething : function doSomething()
  {
    console.log('Hallo')
  }
}
obj.doSomething() // Hallo

//3. Assign ke dalam array :
let arr = []
arr.push(function doSomething()
  {
    console.log('Hallo')
  }
)
arr[0]() // Hallo
```

## Higher-order function

Higher-Order function adalah fungsi yang beroperasi didalam fungsi lain, baik menggunakannya sebagai argumen atau mengembalikan nilainya. Secara sederhana, higher-order function adalah fungsi yang menerima fungsi sebagai argumen atau mengembalikan nilai fungsi sebagai output. Contoh Higher-order function.

```
//contoh Higher-order function

//contoh 1 Menggunakan Array.filter() dan Array.map()
let orang = [
    { nama: 'Sutan', umur: 16 },
    { nama: 'Joni', umur: 18 },
    { nama: 'Mark', umur: 27 },
    { nama: 'Back', umur: 14 },
    { nama: 'Toni', umur: 24},
]
orang = orang.filter(orang => orang.umur >= 18)
alert(orang.map(orang => orang.nama)) // ['Joni','Mark','Toni']
```

```
//contoh 2 Menggunakan Array.reduce()
const keranjang = [
    {
      produk: 'Indomie Mie Goreng',
      harga: 2500
    },
    {
      produk: 'Aqua Gelas',
      harga: 500
    },
    {
      produk: 'Sepatu Sport',
      harga: 75000
    }
]
const totalHarga = keranjang.reduce((acc, cur) => acc + cur.harga, 0)
alert('Rp.' + totalHarga) // Hasil: Rp.78000
```

- Array.map() adalah salah satu metode dari higher-order function bawaan yang berfungsi membuat array baru dengan memanggil fungsi callback yang disediakan sebagai argumen pada setiap elemen dalam array. Metode map() akan mengambil setiap nilai yang dikembalikan dari fungsi callback, lalu membuat array baru dengan menggunakan nilai-nilai yang baru.
- Array.filter() adalah salah satu metode dari higher-order function bawaan yang berfungsi untuk membuat array baru dengan mengoperasikan fungsi callback pada setiap anggota dengan mem-filter nilai-nilai didalam array yang sudah ada. Adapun argumen yang ada pada callback filter() adalah: value, index.
- Array.reduce() adalah salah satu metode dari higher-order function bawaan yang berfungsi untuk menjalankan fungsi callback pada setiap anggota dari array yang dipanggil dengan menghasilkan single output. Metode reduce() menerima 2 parameter, yaitu fungsi callback dan initialValue (opsional but recommended). Sedangkan fungsi callback-nya menghasilkan 4 argumen, yaitu: accumulator, currentValue, currentIndex (opsional), sourceArray (opsional).

## Execution Context

Ketika kode javascript di run / dijalankan, environtment (lingkup) dimana kode tersebut dieksekusi sangatlah penting, interpreter akan menentukan kode tersebut termasuk pada environtment yang mana. Kode tersebut akan ditentukan menjadi salah satu dari 3 environtment :

- Kode Global (Global Code) : default environtment, dimana kode pertama kali dieksekusi (executed).
- Kode Fungsi (Function Code) : Ketika kode yang dijalankan masuk ke dalam suatu fungsi.
- Kode Eval (Eval Code) : Teks yang akan dieksekusi didalam internal fungsi Eval.

Agar mudah dipahami, anggap saja Konteks Eksekusi sebagai environtment / scope / lingkup lokasi dimana kode yang sedang di “baca” oleh interpreter. Langsung saja, contoh Kode yang telah di”baca” dan ditentukan global dan function/lokal konteksnya.

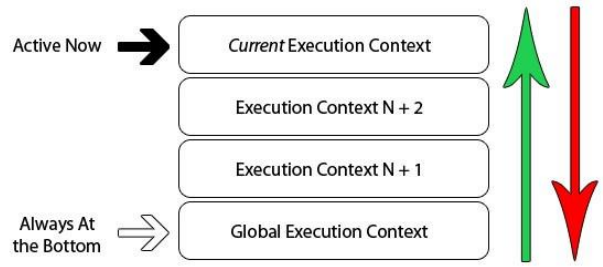
```
// global context

var sayHello = 'Hello';

function person() {          // execution context

    var first = 'David',
        last  = 'Shariff';

    function firstName() {  // execution context
        return first;
    }

    function lastName() {   // execution context
        return last;
    }

    alert(sayHello + firstName() + ' ' + lastName());
}
```

Kita memiliki satu Konteks Global yang berada di dalam border warna ungu, dan 3 Konteks Fungsi / Lokal yang berada di dalam border hijau, biru, dan orange. Konteks Global hanya ada satu, yang mana bisa diakses dari semua konteks-konteks fungsi / lokal yang ada di keseluruhan kode teman-teman.

Setiap kali fungsi dipanggil, maka konteks fungsi akan dibuat, konteks fungsi ini tidak ada batasan jumlahnya, bisa tak terbatas jumlahnya. Setiap konteks fungsi akan membuat private scope, yang berisi deklarasi-deklarasi di dalam fungsi tersebut dan ini akan bersifat lokal, dan tidak bisa diakses dari luar scope fungsi tersebut.

## Execution Stack

Javascript interpreter pada browser dibuat dengan single thread, yang berarti hanya ada satu hal pada browser yang bisa dikerjakan dalam satu waktu. Action-action atau events harus menunggu untuk dieksekusi, dan dikumpulkan di tempat yang namanya Execution Stack / Tumpukan Eksekusi. Stack / Tumpukan itu menggunakan salah satu metode LIFO (Last In first Out), singkatnya yang masuk terakhir yang dieksekusi duluan. Agar lebih mudah, bisa dilihat pada diagram berikut.



Seperti yang kita ketahui, pertama kali browser menjalankan kode, maka langsung masuk ke global execution context / konteks eksekusi global, ini defaultnya, maka dari itu global execution context selalu berada dipaling bawah Execution Stack. Apabila kode globalmu memanggil sebuah fungsi, maka program akan memasuki fungsi yang dipanggil itu, membuat Execution Context baru dan kemudian di push ke bagian paling atas Execution Stack.

Jika teman-teman memanggil fungsi baru didalam fungsi yang tadi dipanggil, maka hal yang sama akan terjadi. Program akan masuk kedalam fungsi baru tersebut, dan membuat Execution Context baru kemudian di push ke atas Execution Stack, ketika fungsi tersebut selesai mengeksekusi Execution Context-nya, maka Execution Context tersebut akan di "pop" dan dibuang dari tumpukan tadi. Kemudian lanjut ke konteks dibawahnya.

```
//contoh Execution Stack
(function foo(i) {
    if (i === 3) {
        return;
    }
    else {
        foo(++i);
    }
}(0));
```

kode diatas, akan memanggil fungsi tersebut sebanyak 3 kali, increment nilai i sebanyak 1. Setiap kali fungsi foo dipanggil, maka konteks eksekusi dibuat dan ketika konteks eksekusi telah selesai dieksekusi maka konteks eksekusi tersebut di "pop" dari tumpukan dan lanjut ke tumpukan dibawahnya, sampai kembali ke Execution Context global.

Hal-hal yang perlu diingat tentang tumpukan eksekusi

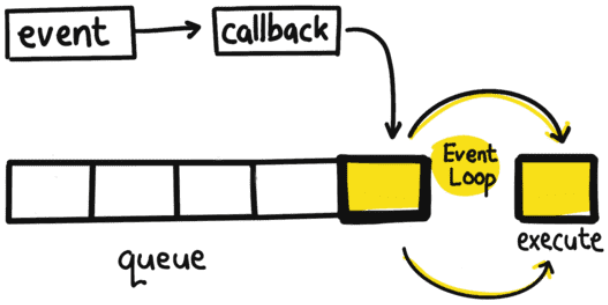
- tumpukan eksekusi terdiri dari satu atau lebih Execution Context (global atau fungsional
- Hanya memiliki satu thread (single threaded)
- eksekusinya syncronous
- 1 konteks global
- konteks fungsi bisa tak terbatas
- setiap pemanggilan fungsi akan membuat satu Execution Context baru.

## Event Loop

Event loop adalah bagian dari JavaScript Runtime yang bertugas untuk menangani Event Callback, Event Callback sendiri adalah bagian dari code yang dieksekusi setelah event tertentu. Contoh Kasus : Klik tombol Download di browser.

- mouse click adalah event
- function yang bertugas untuk mengunduh adalah callback

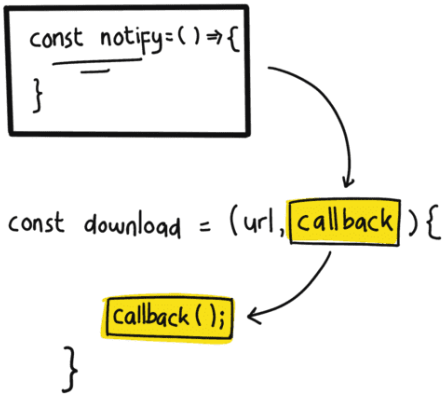
Ketika event terjadi maka callback dari event tersebut akan ditempatkan pada suatu tempat yang disebut Event Handler Queue atau Queue.  Event Loop akan terus memonitor Queue dan akan mengeksekusi callback sesuai urutan siapa yang pertama masuk ke dalam Queue.




```
//contoh event loop
console.log("Before delay");

function delayBySeconds(sec) {
   let start = now = Date.now()
   while(now-start < (sec*1000)) {
     now = Date.now();
   }
}
delayBySeconds(5);
// Executes after delay of 5 seconds
console.log("After delay");
```

## Callbacks

Callback adalah function yang menjadi argument untuk function lain dan akan dieksekusi pada poin tertentu, bisa jadi saat ini atau nanti.



```
//contoh Callbacks
const notify = () => {
    console.log('Download complete!');
};

const download = (url, callback) => {
  console.log(`Downloading from ${url}....`);
  callback();
};

const url = 'https://brachio.site/download';
download(url, notify);
```

Pada code di atas function notify adalah callback function, dipanggil setelah code console.log(Downloading from \${url}....);.

## Promises dan Async/Await

Promise bisa dikatakan sebagai object yang menyimpan hasil dari sebuah operasi asynchronous baik itu hasil yang diinginkan (resolved value) atau alasan kenapa operasi itu gagal (failure reason).

Kita ambil contoh seperti saat kita memesan ojek online. Saat kita mencari driver lewat aplikasi, aplikasi akan berjanji(promise) memberi tahu hasil dari pencarian kita.Hasilnya bisa diantara dua, yaitu driver ditemukan (resolved value) atau alasan kenapa driver tidak ditemukan(failure reason).

Sebuah Promise berada di salah satu diantara 3 kondisi(state):

- pending, operasi sedang berlangsung
- fulfilled, operasi selesai dan berhasil
- rejected, operasi selesai namun gagal

Sama seperti pada kasus memesan ojek online, status permintaan kita pada aplikasi online diantara tiga kondisi:

- Mencari driver (pending)
- Menemukan driver (fulfilled)
- Driver tidak ditemukan (rejected)

```
//contoh Promise
let progress = 100;
const download = new Promise((resolve, reject) => {
  if (progress === 100) {
    resolve('Download complete');
  } else {
```

```
    reject('Download failed');
  }
});
```

Function setelah keyword new Promise disebut executor. Dan di dalam executor terdapat dua callback function:

- resolve(value) adalah callback function yang dieksekusi jika operasi yang dijalankan oleh executor berhasil(fulfilled)
- reject(error) adalah callback function yang akan dieksekusi jika operasi gagal (rejected)

Async/Await diperkenalkan di ES8 / ES2017 untuk menghandle operasi asynchronous dengan syntax yang lebih mirip dengan synchronous. Async/Await sendiri dibuat di atas Promise.Kita bisa membuat simulasi proses download menggunakan code dibawah ini.

```
//contoh Async Await
const getStatus = (url) => {
    console.log(`Downloading from ${url}...`);
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        resolve('Download Complete');
      }, 3000);
    });
};
async function download(url) {
    let status = await getStatus(url); // tunggu sampai promise selesai
    console.log(status);
}
const url = 'https://brachio.site/download';
download(url);
```

Pada code di atas keyword await membuat function download harus menunggu sampai function getStatus() selesai dieksekusi.

# Referensi


Faridho. (2020, September 24). *Mengenal Konsep Closure Pada Javascript - JavaScript & TypeScript Upgrade - Medium*. Medium; JavaScript & TypeScript Upgrade. https://medium.com/javascript-typescript-upgrade/mengenal-konsep-closure-pada-javascript-39a6e915b152

*Function pada JavaScript*. (2020). Devsaurus.com. https://devsaurus.com/javascript-function#immediately-invoked-function-expression-iife

Arya Rifqi Pratama. (2019, April 7). *Mengenal Lebih Dalam Tentang First Class Function Pada Javascript*. Medium; Medium. https://medium.com/@aryarifqipratama/mengenal-lebih-dalam-tentang-first-class-function-pada-javascript-9d12cb11febe

Sutan Gading Fadhillah Nasution. (2019, August 19). *Mengenal higher-order function di JavaScript | Gading's Hideout*. Gading's Hideout. https://gading.dev/id/blog/mengenal-higher-order-function-di-javascript

sekolahkoding. (2018, February 9). *Apa itu Execution Context dan Execution Stack pada JavaScript ? - Forum Sekolah Koding*. Sekolahkoding.com. https://sekolahkoding.com/forum/apa-itu-execution-context-dan-execution-stack-pada-javascript

Nagarajan, M. (2019, September 4). *What is the Execution Context & Stack in JavaScript?* Medium; Medium. https://medium.com/@itIsMadhavan/what-is-the-execution-context-stack-in-javascript-e169812e851a

*Panduan Lengkap Asynchronous pada JavaScript*. (2021). Devsaurus.com. https://devsaurus.com/javascript-asynchronous