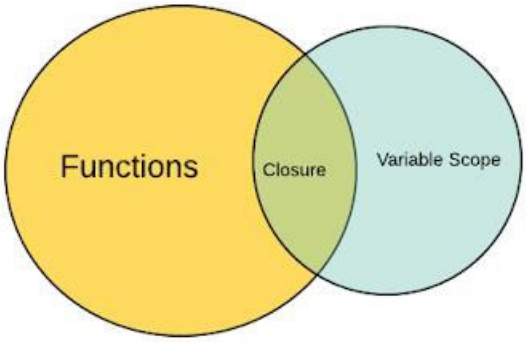
Nama : Husliana Pratiwi
NIM : 119140042
Kelas : RB

Link github : https://github.com/huslianapratiwi/Tugas-2-PAM.git

1. **Closure**
   Closure adalah fungsi yang dijalankan oleh fungsi lain (nested function) sehingga fungsi tersebut dapat mengakses atau merekam variabel dalam lexical scope pada fungsi induk (parent function). Closure adalah kombinasi dari fungsi dan lexical environtment dimana fungsi itu dideklarasikan. Closure dibuat ketika suatu fungsi me-return atau mengembalikan nilai fungsi lain. Nested function atau child function dapat mengakses variabel yang dideklarasikan di luar cakupannya (parent function).



Contoh :

```
function mobil(merk) {
    var tahun = "2002";
    return function(type) {
        return merk +" "+type+" "+tahun;
    }
}

var sedan = mobil("Honda");
var minivan = mobil("Mitsubisi");
sedan("Civic"); //mencetak "Honda Civic 2002"
minivan("Pajero"); //mencetak "Mitsubisi Mobilio 2002"
console.dir(sedan);
```

Kegunaan Closure : Membuat sebuah fungsi yang dinamis, kita bisa menyusun komposisi fungsi serta kode dapat digunakan berulang kali (reusable).

2. **Immediately Invoked Function Expression**
   Immediately-Invoked Function Expressions (IIFE) adalah pola JavaScript umum yang mengeksekusi fungsi secara instan setelah didefinisikan. Kasus penggunaan yang paling umum untuk IIFE adalah Aliasing variabel global, Membuat variabel dan fungsi pribadi, Fungsi asinkron dalam loop.

Fungsi asinkron dalam loop

```
for (var i = 1; i <= 5; i++) {
    setTimeout(function () {
        console.log('I reached step ' + i);
    }, 1000 * i);
}
```

3. First-Class Function
   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. Sebuah objek disebut sebagai first-class function apabila memenuhi syarat berikut :
   - Mendukung **anonymous function** atau function yang tidak memiliki definisi nama.

```
    (function (x) {
        alert(x);
        return x;
    })
```

   - Mendukung pengiriman **function sebagai argumen** dari function lain.

```
myArray.forEach( function(item, index){
document.write('Isi dari array pada posisi ke : ' + index + ' adalah : ' + item + "<BR />");
});
```

   - **Function sebagai nilai kembalian** dari function lain.

```
    function toggle() {
        document.write("Ini function toggle! <BR />");
        return function() {
            document.write("Ini returning anonymous function dari toogle! <BR />");
            return toggle();
        };
    };
```

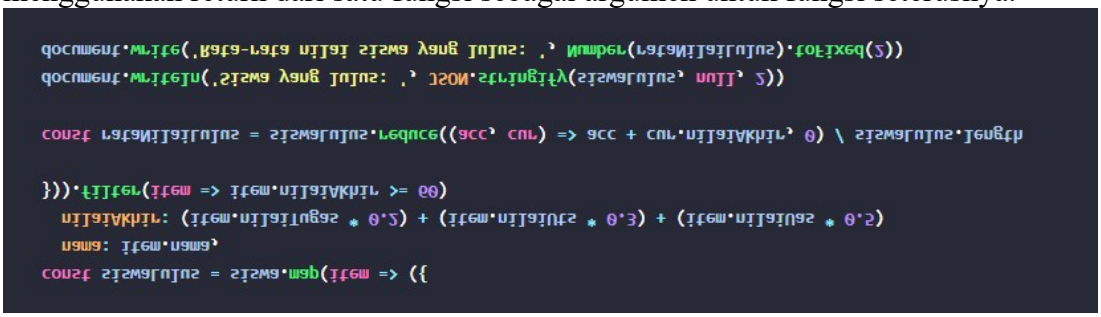
   - Menyimpan **function sebagai variable** atau dalam struktur data lainnya.

```
 function kuadrat(x) {
x = x * x;
return x;
}
var fungsikuadrat = kuadrat;
```

```
 var fungsikuadrat = function (x) {
x = x * x;
return x;
};
```

4. 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. Higher-order function diantaranya adalah map(), filter(), reduce(). Dengan menggunakan higher-order function, kita dapat

melakukan chaining. Chaining sendiri adalah sebuah teknik pemrograman yang menggunakan return dari satu fungsi sebagai argumen untuk fungsi seterusnya.



5. Execution Context
   Ada dua jenis Execution Context dalam JavaScript yaitu :
   - Global Execution Context (GEC)
     GEC adalah Konteks Eksekusi dasar/default di mana semua kode JavaScript yang tidak ada di dalam suatu fungsi akan dieksekusi.

   - Function Execution Context (FEC).
     Setiap kali suatu fungsi dipanggil, mesin JavaScript membuat jenis Konteks Eksekusi yang berbeda yang dikenal sebagai Konteks Eksekusi Fungsi (FEC) di dalam GEC untuk mengevaluasi dan mengeksekusi kode dalam fungsi itu.

   Pembuatan Execution Context (GEC atau FEC) terjadi dalam dua fase yaitu Fase Penciptaan dan Fase Eksekusi.

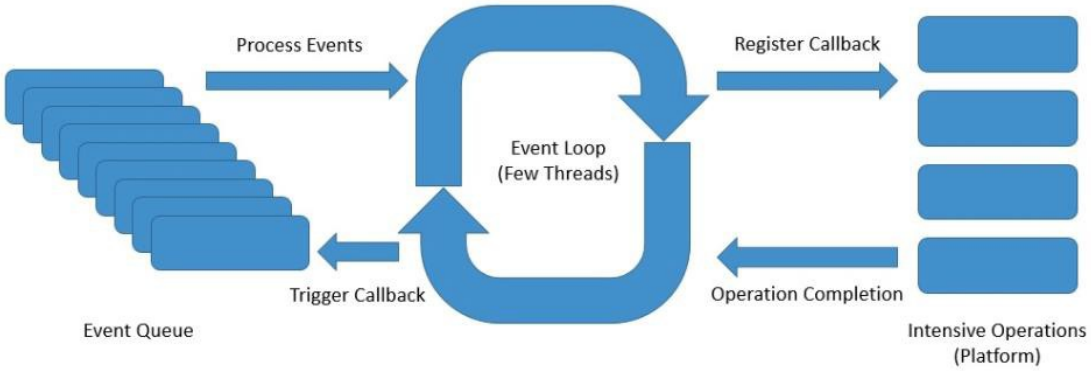
   a. Fase Penciptaan
      Pada fase pembuatan, Execution Context pertama kali dikaitkan dengan Objek Konteks Eksekusi (ECO). Objek Konteks Eksekusi menyimpan banyak data penting yang digunakan kode dalam Konteks Eksekusi selama waktu prosesnya.Fase pembuatan terjadi dalam 3 tahap, di mana properti dari Objek Konteks Eksekusi didefinisikan dan ditetapkan. Tahapan tersebut adalah Pembuatan Objek Variabel (VO), Penciptaan Rantai Lingkup dan Mengatur nilai thiskata kunci.
   b. Fase Eksekusi
      Fase eksekusi adalah tahap di mana eksekusi kode yang sebenarnya dimulai. Pada tahap ini, mesin JavaScript membaca kode dalam Konteks Eksekusi saat ini sekali lagi, kemudian memperbarui VO dengan nilai aktual dari variabel-variabel ini. Kemudian kode diurai oleh parser, diubah menjadi kode byte yang dapat dieksekusi, dan akhirnya dieksekusi.

6. Execution Stack
   Execution stack, juga dikenal sebagai "calling stack" dalam bahasa pemrograman lain, adalah tumpukan dengan struktur LIFO (Last in, First out), yang digunakan untuk menyimpan semua konteks eksekusi yang dibuat selama eksekusi kode. Mesin menjalankan fungsi yang konteks eksekusinya berada di bagian atas tumpukan. Saat fungsi ini selesai, tumpukan eksekusinya dikeluarkan dari tumpukan, dan kontrol mencapai konteks di bawah tumpukan.

```
let a = 'Hello World!';

function first() {
  console.log('Inside first function');
  second();
  console.log('Again inside first function');
}

function second() {
  console.log('Inside second function');
}

first();
console.log('Inside Global Execution Context');
```

7. Event Loop

   Event Loop adalah proses yang hanya memiliki SATU THREAD dengan banyak loop atau proses pengulangan yang tidak terhingga atau terus menerus. Untuk beberapa kasus event loop ini memberikan performance yang lebih bagus dibandingkan concurrency karena tidak ada blok yang terjadi.



   Event loop memiliki flow atau alur proses sebagai berikut.

   a. Event emiter atau sumber dari event akan memasukan task ke event queue. Task-task ini akan antri untuk dieksekusi pada proses loop selanjutnya.
   b. Event loop akan mengambil task dari event queue dan memprosesnya berdasarkan handler yang ada. Pada prose ini, dilakukan prioritas pengerjaan dengan algoritma tertentu. Proses yang terjadi adalah sequential satu persatu dan loop akan kembali segera setelah meregistrasikan callback pada handler dari proses.
   c. Apabila proses telah selesai, event loop akan men-triger callback dan memberikan informasi proses selesai dan mengirimkan hasil kepada yang memanggil proses ini.

8. Callbacks

   JavaScript menjalankan kode secara berurutan dalam urutan top-down. Namun, ada beberapa kasus kode berjalan (atau harus dijalankan) setelah sesuatu yang lain terjadi dan juga tidak berurutan. Ini disebut pemrograman asinkron. Callback memastikan bahwa suatu fungsi tidak akan berjalan sebelum tugas selesai tetapi akan berjalan tepat setelah tugas selesai. Ini membantu kami mengembangkan kode JavaScript asinkron dan membuat kami aman dari masalah dan kesalahan.

```
const message = function() {
    console.log("This message is shown after 3 seconds");
}

setTimeout(message, 3000);
```

9. Promises dan Async/Await
   Promise di dunia javascript dapat di ilustrasikan seperti janji di dunia nyata. Ada 2 kemungkinan yang akan terjadi yaitu janji di tepati atau janji tidak tepati. Ada 2 hal yang akan kita perlajari tentang promise yaitu Membuat promise Menangani promise.
   - Membuat promise
     Untuk membuat promise cukup dengan memanggil constructornya. Constructor promise akan menerima argument sebuah fungsi dengan 2 parameter yaitu resolve dan reject.

     ```
     var promise1 = new Promise((resolve, reject) => {
       setTimeout(() => {
         resolve("foo")
       }, 3000)
     })
     ```

     3 state yang mungkin terjadi dengan promise :
     oPanding -> sedang dalam proses oResolved -> janji di tepati atau berhasil oRejected -> gagal atau janji di batalkan

   - Menangani promise
     Async — await adalah salah satu fitur baru dari javascript yang di gunakan untuk menangani hasil dari sebuah promise. Caranya adalah dengan menambahkan kata ‘async’ di depan sebuah fungsi untuk mengubahnya menjadi asynchronous. Sedangkan await berfungsi untuk menunda sebuah kode di jalankan, sampai proses asynchronous berhasil.

     ```
     const init = async() => {

       await createPost(newPost)

       getPost()

     }
     ```