Nama : Lusi Lesmana Tamba
Nim : 119140072
Kelas : RB

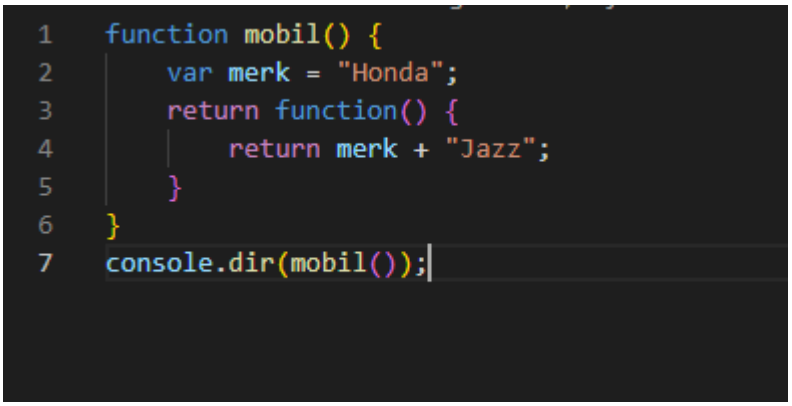
Buatlah sebuah rangkuman materi javascript lanjutan dari kata kunci di bawah dan berikan contoh sendiri:

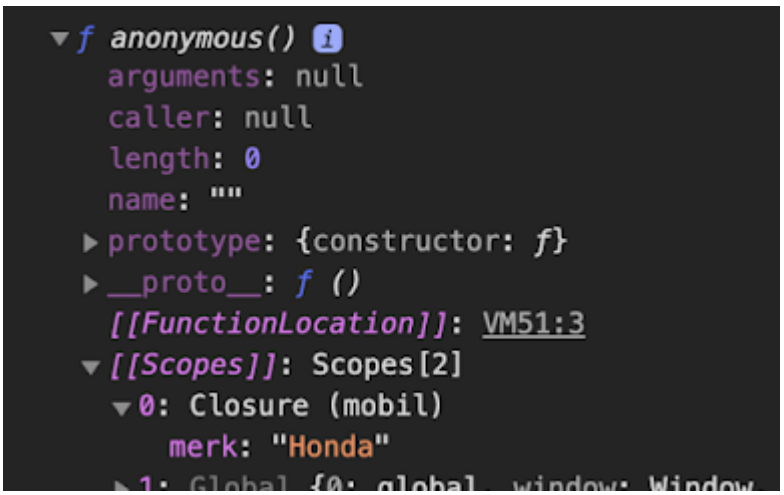
1.Closure
2.Immediately Invoked Function Expression
3.First-class function
4.Higher-order function
5.Execution Context
6.Execution Stack
7.Event Loop
8.Callbacks
9.Promises dan Async/Await

Jelaskan secara sederhana (boleh menggunakan gambar atau model), sertakan sumber dan boleh dari blog perorangan atau website individu.

file pdf berisikan:
1. Identitas Mahasiswa (NIM, NAMA, KELAS)
2. Lampirkan github dengan memuat tugas 2 individu
3. Screenshot program

1. **Closure**
   Closure adalah sebuah fungsi yang dieksekusi oleh fungsi lainnya (nested functions) sehingga fungsi tersebut dapat mengakses/merekam variable di dalam lexical scope pada fungsi induk (parent function).
   Contoh:





Ketika kode dijalankan fungsi mobil*()* seketika itu argumennya di destroy dari memory(variabel *merk* pada kode diatas). Dan variable *merk* sekarang berisi function definition hasil return dari fungsi *mobil()*. Jadi *closure* kali ini adalah variabel merk dari fungsi mobil().

Source :
https://www.shinta.dev/2020/10/apa-itu-closure.html#:~:text=Closure%20pada%20JavaScript&text=Di%20JavaScript%20sendiri%2C%20closure%20merupakan,fungsi%20induk(parent%20function).

2. **Immediately Invoked Function Expression**

   Dengan menggunakan pola IIFE ini maka kita dapat memanggil suatu fungsi tanpa harus menyimpannya, dan dapat pula menyembunyikan kode yang kompleks beserta variabel variabel di dalamnya yang berpotensi untuk mengotori *global scope* javascript aplikasi jika diekspos.

   Contoh :

```
1  function sayHi() {
2      console.log('Hi Brachio');
3  }
4  // kita bisa memanggil function di atas dengan
5  window.sayHi;
```

   Source :
   https://icalrn.id/immediately-invoked-function-expression/

3. **First-class function**

   Sebuah bahasa pemrograman dikatakan memiliki First-Class Function jika fungsi dalam bahasa tersebut diperlakukan seperti variabel lainnya.Sehingga fungsi tersebut bisa ditugaskan menjadi variabel lain atau juga diteruskan sebagai argumen atau juga dikembalikan oleh fungsi lain.

   Contoh :

```
<script>
    const Arithmetics = {
        add: (a, b) => {
            return `${a} + ${b} = ${a + b}`;
        },
        subtract: (a, b) => {
            return `${a} - ${b} = ${a - b}`
        },
        multiply: (a, b) => {
            return `${a} * ${b} = ${a * b}`
        },
        division: (a, b) => {
            if (b != 0) return `${a} / ${b} = ${a / b}`;
            return `Cannot Divide by Zero!!!`;
        }

    }

    document.write(Arithmetics.add(100, 100) + "<br>");
    document.write(Arithmetics.subtract(100, 7) + "<br>");
    document.write(Arithmetics.multiply(5, 5) + "<br>");
    document.write(Arithmetics.division(100, 5));
</script>
```

Source :
https://www.geeksforgeeks.org/what-is-the-first-class-function-in-javascript/#:~:text=First%2DClass%20Function%3A%20A%20programming,be%20returned%20by%20another%20function.

4. **Higher-order function**

   Adalah sebuah fungsi yang memiliki sebuah fungsi sebagai argumen atau fungsi sebagai nilai return dari fungsi tersebut. Nah fungsi yang menjadi argumen atau nilai return dari Higher Order Function disebut callback function. Beberapa contoh higher order function yaitu penggunaan fungsi array seperti find, filter dl.

   Contoh :

   1. Find

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
const number = numbers.find(number => number >= 5 // 5 adalah nilai yang ingin dicari)
console.log(number) // menampilkan angka 5
```

   2. Filter

```
const numbers = [1, 23, 21, 51, 43, 2, 54, 34]
const newNumbers = numbers.filter((number) => number > 40)
console.log(newNumbers) // [ 51, 43, 54 ]
```

Source :
https://dev.to/iamalinurdin/higher-order-function-di-javascript-4jbg#:~:text=Higher%20Order%20Function%20singkat%20kata,Order%20function%20disebut%20callback%20function.

5. **Execution Context**
   Adalah lingkungan yang mana code javascript tersebut dieksekusi. Ada 3 tipe dari EC yaitu Global Execution Context, Functional Execution Context, dan Eval.
   Contoh :

```
executionContextObj = {
    'scopeChain': {
        // variableObject + semua variableObject dari parent-parent-nya
    },
    'variableObject': {
        // function arguments / parameters, lokal variabel and function declarations
    },
    'this': {}
}
```

   executionContextObj ini dibuat ketika suatu fungsi dipanggil, tetapi tepat “sebelum” fungsinya benar-benar dieksekusi. Jadi interpreternya akan nge scan parameter / argumen yang dipass kedalam fungsi, deklarasi fungsi, dan lokal variabel dari keseluruhan fungsi yang dipanggil tadi. Nah, itu yang dinamakan tahap 1, Tahap Pembuatan.
   Source :
   https://medium.com/@happymishra66/execution-context-in-javascript-319dd72e8e2c#:~:text=Execution%20context%20(EC)%20is%20defined,to%20at%20a%20particular%20time.

6. **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.
   Contoh :

```
(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 konteks eksekusi global.
Source :
https://sekolahkoding.com/forum/apa-itu-execution-context-dan-execution-stack-pada-javascript

7. **Event Loop**
proses yang hanya memiliki SATU THREAD dengan banyak loop atau proses pengulangan yang tidak terhingga atau terus menerus. Pada cara ini kita hanya mengerjakan satu task dan untuk menghandle banyak request kita melakukan seleksi dan prioritas terhadap task yang dikerjakan. Dengan kata lain kita melakukan Thread Safe sehingga dijamin pasti proses ini akan aman terhadap tumpang tindih satu proses dengan proses lainnya.
Contoh :

```
const bar = () => console.log('bar')

const baz = () => console.log('baz')

const foo = () => {
  console.log('foo')
  bar()
  baz()
}

foo()
```

Source :

https://medium.com/the-legend/concurrency-vs-event-loop-5648882ad668#:~:text=Event%20Loop%20adalah%20proses%20yang,prioritas%20terhadap%20task%20yang%20dikerjakan.

8. **Callbacks**

Dalam javascript, fungsi adalah objek sehingga fungsi dapat diteruskan sebagai argumen. Callback adalah fungsi yang diteruskan ke fungsi lain sebagai argumen. Fungsi ini selanjutnya dapat dipanggil selama eksekusi fungsi urutan yang lebih tinggi yang merupakan argumen.

Contoh :

```
const isEven = (n) => {
    return n % 2 == 0;
  }

  let printMsg = (evenFunc, num) => {
    const isNumEven = evenFunc(num);
    console.log(`The number ${num} is an even number: ${isNumEven}.`)
  }

  // Pass in isEven as the callback function
  printMsg(isEven, 4);
  // Prints: The number 4 is an even number: True.
```

Source :
https://www.codecademy.com/learn/game-dev-learn-javascript-higher-order-functions-and-iterators/modules/game-dev-learn-javascript-iterators/cheatsheet

9. **Promises dan Async/Await**

Promises digunakan untuk melacak apakah even yang tidak sinkron telah dieksekusi atau tidak dan menentukan apa yang terjadi setelah hal tersebut.

Ada 3 jenis status object yaitu :

1. Pending yaitu keadaan awal, sebelum event terjadi.
2. Resolved yaitu setelah operasi selesai dan sukses.
3. Rejected yaitu jika operasi mengalami error selama eksekusi dan promise gagal.

Untuk mengatasi error promises, untuk menyukseskan resolved promises digunakan metode .then(), untuk rejected promises digunakan metode .catch().

Contoh :

```
<script>
  const promise = new Promise(function (resolve, reject) {
    const string1 = "geeksforgeeks";
    const string2 = "geeksforgeeks";
    if (string1 === string2) {
      resolve();
    } else {
      reject();
    }
  });

  promise
    .then(function () {
      console.log("Promise resolved successfully");
    })
    .catch(function () {
      console.log("Promise is rejected");
    });
</script>
```

Async/Await merupakan sebuah syntax khusus yang digunakan untuk menangani Promise agar penulisan code lebih efisien dan rapi.Async/Await adalah salah satu cara untuk mengatasi masalah asynchronous pada Javascript selain menggunakan callback dan promise.

1. Pada implementasi Async/Await, kita menggunakan kata kunci async sebelum fungsi. Await sendiri hanya bisa digunakan pada fungsi yang menggunakan async.
2. Untuk menggunakan Async/Await, kembalian dari suatu fungsi harus merupakan suatu Promise. Async/Await tidak dapat berdiri tanpa adanya Promise.
3. Tidak seperti Promise, dengan Async/Await maka suatu baris kode dapat tersusun rapi mirip dengan kode yang sifatnya synchronous.
4. Setiap baris yang menggunakan await, akan ditunggu sampai Asynchronous Promise tersebut di resolve.

Untuk menghandle error Async/Await kita dapat menggunakan try catch di dalam function yang kita buat, sehingga jika terjadi error kita dapat menangkap errornya dalam block catch. Contoh:

```
<script>
  const helperPromise = function () {
    const promise = new Promise(function (resolve, reject) {
      const x = "geeksforgeeks";
      const y = "geeksforgeeks";
      if (x === y) {
        resolve("Strings are same");
      } else {
        reject("Strings are not same");
      }
    });

    return promise;
  };

  async function demoPromise() {
    try {
      let message = await helperPromise();
      console.log(message);
    } catch (error) {
      console.log("Error: " + error);
    }
  }

  demoPromise();
</script>
```

Source :
https://www.geeksforgeeks.org/difference-between-promise-and-async-await-in-node-js/

LINK GITHUB :
https://github.com/LusiLT/Tugas2_PAM