Nama : Fahriza Yusefa
NIM : 119140005
Kelas : RB

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

1. Closure

   Closures adalah *inner function* yang punya hak akses ke *scope* di dalam *function* di mana dia bernaung, dan *scope-scope* di *function* luar lain nya. *Closures* ini bisa mengakses ke variabel-variabel yang ada di dalam *scope* yang berbeda:

   a. Variabel yang ada di dalam *scope* nya sendiri (ini udah jelas bisa lah ya)
   b. Variabel yang ada di dalam *scope global*;
   c. Dan variabel yang ada di dalam *scope function* di luar nya / *parent function* nya.

   C*losures* itu juga bisa mengakses ke:

   a. Parameter yang dia punya
   b. Parameter yang ada di dalam *function* di luar nya / *parent function* nya.

   Contoh :

```
const variabelGlobal = 'Ini variabel global'

function functionOuter(param1) {
  const variabel1 = 'Ini variabel satu'

  function functionInner(param2) {
    const variabel2 = 'Ini variabel dua'

    console.log(`variabelGlobal: ${variabelGlobal}`)
    console.log(`variabel1: ${variabel1}`)
    console.log(`variabel2: ${variabel2}`)
    console.log(`param1: ${param1}`)
    console.log(`param2: ${param2}`)
  }

  functionInner('Ini param dua')
}

functionOuter('Ini param satu')
```

   Referensi : https://medium.com/@rivayudha/closures-apaan-tuh-alasan-kamu-benci-javascript-8484a3437a40

2. Immediately Invoked Function Expression

   Immediately Invoked : Bagian ini mudah dijelaskan dan didemonstrasikan. Jenis fungsi ini dipanggil segera dipanggil karena fungsi-fungsi ini dieksekusi segera setelah dipasang ke tumpukan, tidak memerlukan panggilan eksplisit untuk memanggil fungsi tersebut. Jika kita melihat sintaks itu sendiri, kita memiliki dua pasang kurung tertutup, yang pertama berisi logika yang akan dieksekusi dan yang kedua umumnya adalah apa yang kita sertakan ketika kita memanggil suatu fungsi, kurung kedua bertanggung jawab untuk memberitahu compiler bahwa ekspresi fungsi harus segera dieksekusi.

Function Expressions : Mudah dipahami bahwa Ekspresi Fungsi adalah Fungsi yang digunakan sebagai ekspresi. JavaScript memungkinkan kita menggunakan Fungsi sebagai Ekspresi Fungsi jika kita Menetapkan Fungsi ke variabel, membungkus Fungsi di dalam tanda kurung, atau meletakkan logika bukan di depan fungsi. Program berikut mengilustrasikan berbagai cara kita dapat membuat Ekspresi Fungsi.
Contoh :

```
// Creating Function Expression by assigning to a variable.
var myFunc = function() { return "GeeksforGeeks"; };

// Creating Function Expression Using Logical Not.
!function() { return "GeeksforGeeks"; };

// Creating Function Expression Using Parentheses.
(function() { return "GeeksforGeeks"; });
```

Referensi
https://stackabuse.com/javascripts-immediately-invoked-function-expressions/ :
https://www.geeksforgeeks.org/javascript-immediately-invoked-function-expressions-iife/

3. First-class function
First-class function adalah sebuah bahasa pemrograman dikatakan memiliki fungsi kelas satu jika fungsi dalam bahasa tersebut diperlakukan seperti variabel lainnya. Jadi fungsi dapat ditugaskan ke variabel lain atau diteruskan sebagai argumen atau dapat 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>
```

Referensi :
https://www.geeksforgeeks.org/what-is-the-first-class-function-in-javascript/

4. Higher-order function

   Higher-order function adalah fungsi yang menerima fungsi sebagai parameter dan/atau mengembalikan fungsi.

   Contoh :

```
const arr1 = [1, 2, 3];
const arr2 = [];

for(let i = 0; i < arr1.length; i++) {
  arr2.push(arr1[i] * 2);
}

// prints [ 2, 4, 6 ]
console.log(arr2);
```

   Referensi :

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

   https://blog.bitsrc.io/understanding-higher-order-functions-in-javascript-75461803bad

5. Execution Context

   Execution Context didefinisikan sebagai lingkungan di mana kode JavaScript dieksekusi. Dengan lingkungan, maksud saya nilai ini, variabel, objek, dan fungsi yang dapat diakses oleh kode JavaScript pada waktu tertentu.

   Contoh :

```
let x = 10;

function timesTen(a){
    return a * 10;
}

let y = timesTen(x);

console.log(y); // 100
```

   Referensi :

   https://medium.com/@happymishra66/execution-context-in-javascript-319dd72e8e2c#:~:text=Execution%20context%20(EC)%20is%20defined,to%20at%20a%20particular%20time.

   https://www.javascripttutorial.net/javascript-execution-context/

6. Execution Stack

   Execution stack dikenal juga sebagai Call Stack, yang berfungsi untuk melacak semua Execution Context yang dibuat selama siklus hidup script.

   JavaScript adalah bahasa utas tunggal, yang berarti bahwa ia hanya mampu menjalankan satu tugas pada satu waktu.

   Contoh :

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

   Referensi :

   https://medium.com/@itIsMadhavan/what-is-the-execution-context-stack-in-javascript-e169812e851a

   https://www.freecodecamp.org/news/execution-context-how-javascript-works-behind-the-scenes/#:~:text=JavaScript%20Execution%20Stack,single%20task%20at%20a%20time.

7. Event Loop

   JavaScript memiliki model runtime berdasarkan event loop, yang bertanggung jawab untuk mengeksekusi kode, mengumpulkan dan memproses peristiwa, dan mengeksekusi sub-tugas yang diantrekan.

   Contoh :

```
function foo(b) {
  const a = 10;
  return a + b + 11;
}

function bar(x) {
  const y = 3;
  return foo(x * y);
}

const baz = bar(7); // assigns 42 to baz
```

   Referensi :

   https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

8. Callbacks

   Callbacks adalah fungsi yang diteruskan ke fungsi lain sebagai argumen, yang kemudian dipanggil di dalam fungsi luar untuk menyelesaikan beberapa jenis rutinitas atau tindakan.

   Contoh :

```
function greeting(name) {
  alert(`Hello, ${name}`);
}

function processUserInput(callback) {
  const name = prompt("Please enter your name.");
  callback(name);
}

processUserInput(greeting);
```

   Referensi :

   https://developer.mozilla.org/en-US/docs/Glossary/Callback_function

9. Promises dan Async/Await

   Promise di dunia javascript dapat diilustrasikan seperti janji di dunia nyata. Ada 2 kemungkinan yang akan terjadi yaitu janji ditepati atau janji tidak ditepati. Ada 2 hal yang akan kita pelajari tentang promise yaitu :

   1. Membuat promise
   2. Menangani promise

   Async — await adalah salah satu fitur baru dari javascript yang digunakan 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 dijalankan, sampai proses asynchronous berhasil.

   Contoh :

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

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

    const data = await getUser()

    console.log(data)

}
```

Referensi :
https://medium.com/codeacademia/belajar-callback-promise-dan-async-await-dalam-5-menit-fa1564956ab0
https://www.matawebsite.com/blog/mengenal-async-await-javascript

Link Github : https://github.com/FahrizaYusefa/Tugas2-PAM