# PENGEMBANGAN APLIKASI MOBILE

**Link github: https://github.com/jacksih/Tugas2.git**


**JACKY Z.M SIHOMBING**

**120140226**

**RA**

**INSTITUT TEKNOLOGI SUMATRA**

**LAMPUNG SELATAN**

**2022**

1. Closure

Closure adalah adalah salah satu konsep penting dalam JavaScript. Hal ini banyak dibahas dan masih membingungkan konsep. Dengan kata lain, penutupan memberi Anda akses ke ruang lingkup fungsi luar dari fungsi dalam. Dalam JavaScript, penutupan dibuat setiap kali fungsi dibuat, pada waktu pembuatan fungsi.

Source: https://www.tutorialsteacher.com/javascript/closure-in-javascript

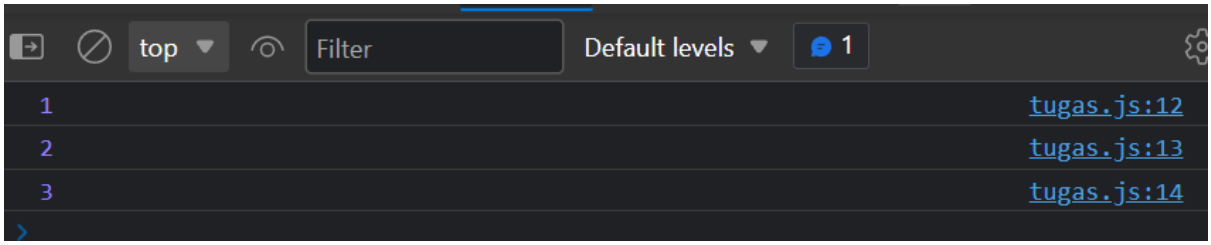
```
function Counter() {
    var counter = 0;

    function IncreaseCounter() {
        return counter += 1;
    };

    return IncreaseCounter;
}

var counter = Counter();
console.log(counter())
console.log(counter())
console.log(counter())
```

hasil




2. Immediately Invoked Function Expression

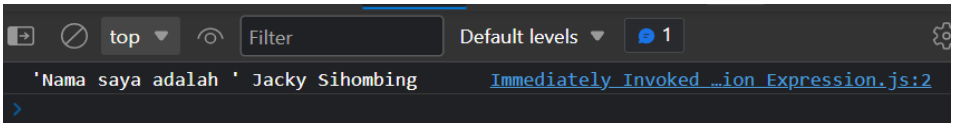
Adalah diucapkan "rapuh", adalah pola JavaScript umum yang mengeksekusi fungsi secara instan setelah didefinisikan. Pengembang terutama menggunakan pola ini untuk memastikan variabel hanya dapat diakses dalam lingkup fungsi yang ditentukan. Deklarasi Fungsi dimulai dengan kata kunci fungsi, diikuti dengan nama fungsi dan argumen apa pun yang mungkin diperlukan. Misalnya, kita dapat membuat fungsi logName menggunakan deklarasi seperti ini:

Source : https://stackabuse.com/javascripts-immediately-invoked-function-expressions/

```
const logUserName = function logName(name) {
    console.log(`'Nama saya adalah ' ${name}`);
};

logUserName("Jacky Sihombing");
```

hasil:




3. First-class function

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. JavaScript memperlakukan fungsi sebagai warga kelas satu. Ini berarti bahwa fungsi hanyalah sebuah nilai dan hanya jenis objek lain.

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

Contoh: Mari kita ambil contoh untuk lebih memahami fungsi kelas satu.

```
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));
```

hasil

```
100 + 100 = 200
100 - 7 = 93
5 * 5 = 25
100 / 5 = 20
```

4. Higher-order function

adalah fungsi reguler yang mengambil satu atau lebih fungsi sebagai argumen dan/atau mengembalikan fungsi sebagai nilai darinya.

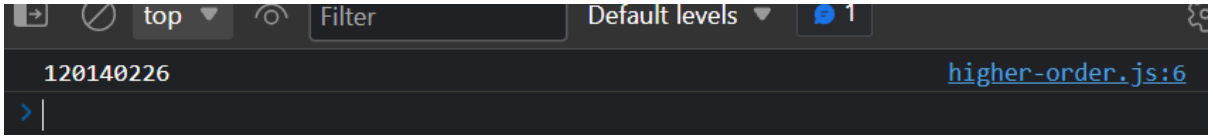
Source: https://blog.greenroots.info/higher-order-functions-in-javascript

Berikut adalah contoh fungsi yang menggunakan fungsi sebagai argumen.

```
function mahasiswa(nim) {
    nim();
}

mahasiswa(function () {
    console.log('120140226');
});
```

Hasil :




5. Execution Context

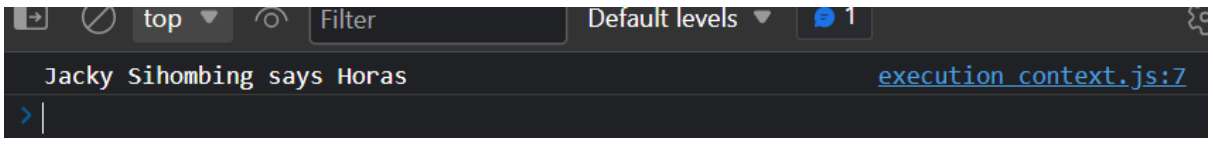
Konsep konteks eksekusi digunakan untuk menggambarkan bagaimana kode bekerja di dalamnya. Konteks eksekusi JavaScript mengacu pada lingkungan yang memungkinkan kode JavaScript dieksekusi. Konteks eksekusi menentukan bagian kode mana yang memiliki akses ke fungsi, variabel, dan objek kode.

Source: https://www.atatus.com/blog/javascript-execution-context/

```
var name = "Jacky Sihombing";
let input = "Horas";

function broadcast(message) {
    return `${name} says ${message}`;
}
console.log(broadcast(input));
```

hasil




6. Execution Stack

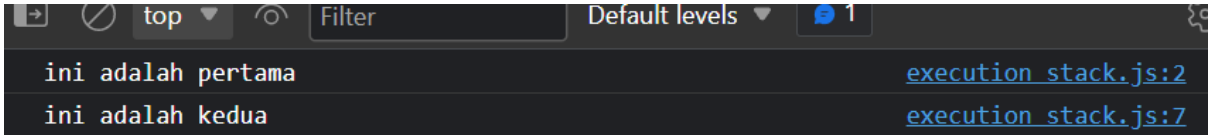
digunakan oleh JavaScript untuk melacak beberapa panggilan fungsi. Ini seperti tumpukan nyata dalam struktur data di mana data dapat didorong dan muncul dan mengikuti prinsip Last In First Out

(LIFO). Kami menggunakan tumpukan panggilan untuk mengingat fungsi mana yang sedang berjalan saat ini. Contoh di bawah ini menunjukkan tumpukan panggilan.

Source: https://www.geeksforgeeks.org/what-is-the-call-stack-in-javascript/

```
function f1() {
    console.log('ini adalah pertama');
}
function f2() {
    f1();
    console.log('ini adalah kedua');
}
f2();
```

hasil:




7. Event Loop

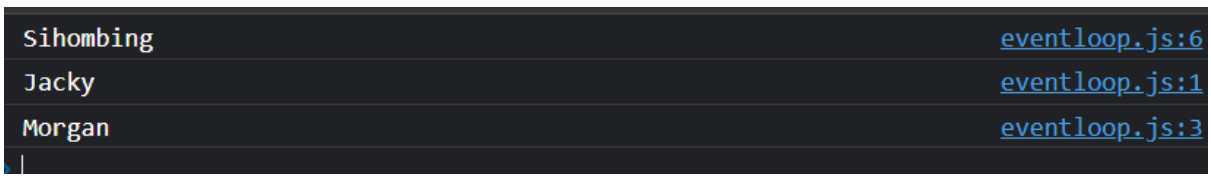
Event Loop adalah salah satu aspek terpenting untuk dipahami tentang JavaScript. Buanyak banget aplikasi yang udah dikerjain, tapi kadang kita gatau processnya dan ngedumel "ngapain sih ini". Makin kesini aku makin pengen belajar lagi nih apasih yang ada dibalik layar si Javascript ini. Nggak cuman asal pake api nya aja.

JavaScript tuh kan single thread ya. Jadi cuman ada satu process yang terjadi pada suatu waktu. Ini adalah batasan yang sebenarnya sangat membantu, karena sangat menyederhanakan cara code tanpa mengkhawatirkan masalah konkurensi

Source : https://id.linkedin.com/pulse/javascript-event-loop-dhimas-yudha-pratama

```
const bar = () => console.log('Jacky')
const baz = () => console.log('Morgan')
const foo = () => {
  console.log('Sihombing')
  bar()
  baz()
}
foo()
```

hasil




8. Callbacks

Fungsi JavaScript dieksekusi dalam urutan yang disebut. Tidak dalam urutan mereka didefinisikan.

Contoh ini akan menampilkan "Selamat tinggal"

Source: https://www.w3schools.com/js/js_callback.asp

```
function myFirst() {
    myDisplayer("Hello");
}
function mySecond() {
    myDisplayer("Goodbye");
}
myFirst()
mySecond()
```

## 9. Promises dan Async/Await

Ada sintaksis spesial untuk bekerja dengan promise dengan cara yang lebih nyaman, dipanggil “async/await”. Ini sangat mudah dipahami dan digunakan.

Source: https://id.javascript.info/async-await

```
async function f() {
    return 1;
}

f().then(alert); // 1
```

hasil