Nama : Ilham Fadhlur Rahman

Nim : 120140125

Program Studi : Teknik Informatika

Kelas : Pengembangan Aplikasi Mobile - RA

Tugas Individu 2 Pengembangan Aplikasi Mobile RA

Closure

Closure adalah sebuah function yang bisa memanggil function induk walaupun function induk telah di tutup, dalam JavaScript function ini memungkinkan kita memiliki variabel private. Closure juga mengadopsi konsep Lambda Calculus sehingga function tersebut dapat di panggil oleh function lainnya (nested function).

```
/* Closure */
function closure() {
  let nama = "Ilham";
  function tampilNama() {
    console.log(nama);
  }
  tampilNama();
}
closure();
```

Immediately-Invoked Function Expression

Immediately-Invoked Function Expression adalah pola yang dapat memanggil suatu fungsi tanpa harus menyimpannya, dan dapat dapat menyembunyikan kode yang kompleks beserta variabel-variabel di dalamnya yang berpotensi mengotori global scope Javascript aplikasi anda jika di ekspos.

```
/* Deklarasi biasa */
const firstName = "Ilham";
const lastName = "Rahman";
const getName = () => {
  const fullName = `${firstName} ${lastName}`;
  return fullName;
};

console.log(firstName);
console.log(getName());

/* Immediately Invoked Function Expression */
```

```
(() => {
  const firstName = "Ilham";
  const lastName = "Rahman";
  const getName = () => {
    const fullName = `${firstName} ${lastName}`;
    return fullName;
  };

  console.log(firstName);
  console.log(getName());
})();
```

First-class Function

First-class Function adalah suatu function yang dapat diteruskan sebagai argumen ke function lainnya, dapat dikembalikan oleh function lain, dan juga dapat ditetapkan sebagai nilai ke variable.

```
/* First-class Function */
const Aritmatika = {
  tambah: (a, b) => {
    return "${a} + ${b} = ${a + b}";
  },
  kurang: (a, b) => {
    return "${a} - ${b} = ${a - b}";
  },
  kali: (a, b) => {
    return "${a} * ${b} = ${a * b}";
  },
  bagi: (a, b) => {
    if (b != 0) return "${a} / ${b} = ${a / b}";
    return "Tidak dapat dibagi nol";
  },
};

console.log(Aritmatika.tambah(100, 100) + "<br>");
console.log(Aritmatika.kurang(100, 7) + "<br>");
console.log(Aritmatika.kali(5, 5) + "<br>");
console.log(Aritmatika.bagi(100, 5));
```

Higher-order Function

Higher-order Function adalah suatu function yang mempunyai function sebagai argumennya, juga dapat dikatan bahwa function sebagai nilai dari function tersebut.

```
/* High-order Function */
const arr1 = [1, 2, 3];
const arr2 = [];

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

console.log(arr2);
```

Execution Context

Execution Context adalah sebuah konsep dalam spesifikasi bahasa pemrograman yang kira-kira dapat disebut sebagai “lingkungan” dimana berfungsi sebagai tempat sebuah fungsi dijalankan, yaitu lingkup variable, argumen fungsi, dan nilai objek.

```
/* Execution Context */
function nama() {
  console.log("Ilham");
  function prodi() {
    console.log("Teknik Informatika");
    function nim() {
      console.log("120140125");
    }
    nim();
  }
  prodi();
}
nama();
```

Execution Stack

Execution Stack adalah suatu function yang dapat memanggil function lain. Ketika function dipanggil, maka function tersebut akan memanggil lagi function yang ada sehingga function yang terakhir dipanggil akan di eksekusi terlebih dahulu.

```
/* Execution Stack */
var str = "Test";

function pertama() {
  var a = 3;
  kedua();
  console.log(`${str} ${a}`);
}
```

```
function kedua() {
  var b = 2;
  ketiga();
  console.log(`${str} ${b}`);
}
function ketiga() {
  var c = 1;
  console.log(`${str} ${c}`);
}

pertama();
```

Event Loop

Event Loop adalah proses yang hanya memiliki satu thread dengan banyak loop atau proses pengulangan yang tidak terhingga. Pada proses ini kita hanya mengerjakan satu task dan untuk menghandle banyak request kita melakukan seleksi dan prioritas terhadap task yang dikerjakan.

```
/* Event Loop */
function pertama() {
  console.log("Ilham");
  setTimeout(function kedua() {
    console.log("120140125");
  }, 1000);
  console.log("Rahman");
}
pertama();
```

Callbacks

Callbacks adalah function yang diteruskan ke function lain sebagai argumen. Function ini kemudian dapat dipanggil selama eksekusi function urutan yang lebih tinggi karena dalam JavaScript, Function adalah objek yang dapat diteruskan sebagai argumen.

```
/* Callbacks */
function helloWord(nama) {
  alert("Hello World " + nama);
}
function tampilkanPesan(callback) {
  const nama = prompt("Masukkan nama anda:");
  callback(nama);
}
tampilkanPesan(helloWord);
```

Promises dan Async/Await

Promises digunakan untuk melacak apakah even itu di tepati atau tidak, ada 3 kondisi yang dapat terjadi 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.

Async adalah suatu method untuk mengeksekusi kode tertentu tanpa menunggu kode sebelumnya di eksekusi.

Await berfungsi untuk menunggu (wait) eksekusi dari sebuah kode.

```
/* Promises */
var promises = new Promises((resolve, reject) => {
  setTimeout(() => {
    resolve("foo");
  }, 1000);
});

/* Async dan Await*/
const getAllUser = async () => {
  try {
    const result = await getUser();
    console.log(result);
  } catch (error) {
    console.log(error);
  }
};
getAllUser;
```

GitHub

Pengembangan-Aplikasi-Mobile/Tugas 2 at main · CaamVilvactDJavu/Pengembangan-Aplikasi-Mobile (github.com)

REFERENSI

Closures - JavaScript | MDN (mozilla.org)

JavaScript Callbacks (w3schools.com)

The event loop - JavaScript | MDN (mozilla.org)

Hoisting,Execution Context Dan Scope – Muhammad Fahri

Mengenal Lebih Dalam Tentang First Class Function Pada Javascript | by Arya Rifqi Pratama | Medium

Understanding Execution Context and Execution Stack in Javascript | by Sukhjinder Arora | Bits and Pieces (bitsrc.io)

Higher Order Functions in JavaScript – Beginner's Guide (freecodecamp.org)

IIFE - MDN Web Docs Glossary: Definitions of Web-related terms | MDN (mozilla.org)

JavaScript Promises (w3schools.com)