Pertemuan 22 - Recap Materi Javascript II

Pendahuluan

Kemarin kita sudah mengulang konsep pemrograman yang sederhana, apa itu Javascript, pendeklarasian `Variable` di Javascript, dan `Control Flow` di Javascript, untuk materi kali ini kita akan fokus kedalam `Function`, `Lamda Function` atau `Anonymous Function` atau lebih sering disebut `Closure`, Konsep `Arrow Function`, `Class`, `Array` di Javascript dan juga `Javascript Object`.

Function

`Function` merupakan block kode yang dapat di eksekusi berkali kali di tempat yang berbeda-beda, `function` ini dapat mengurangi duplikasi kode dan membuat kode lebih rapi.

Cara mendeklarasikan suatu `function` tinggal mengetik `function nama_function ()`, perlu di-ingat `function` tidak boleh ada `-` ataupun spasi.

```
function hello_world() {
    // kode
}
```

`Function` ini dapat menerima banyak parameter, parameter ini digunakan untuk memberi data kepada `function` tersebut agar dapat diolah, parameter ini dapat digunakan seperti variabel tetapi tidak akan mengubah variabel asli dengan kata lain variabel yang diberikan di parameter itu hanya sebuah copy tetapi itu hanya untuk tipe data yang `primitive` kalau `object` akan dikasih referensi jadi apa apa yang di ubah di dalam `function` akan juga mempengaruhi yang diluar contoh :

```
let nama = "Zagar"

function print(teks) {
    teks += " Dangure"
    console.log(teks) // Ini akan menprint "Zagar Dangure"
}

print(nama);
```

Contoh lagi tetapi dengan `object`

```
const array = [];

function push(myarray, val) {
    myarray.push(val);
}
```

```
push(array, 30);
push(array, 20);

console.log(array); // Akan men-print [30, 20]
console.log(typeof array); // Akan men-print object
```

`Function` dapat mengembalikan hasil ke pemanggil dengan `return` didalam `function` tersebut

```
function sum(a, b) {
    return a + b
}

const hasil = sum(1, 2);

console.log(hasil) // Ini akan menprint 3
```

`Function` juga memiliki fitur `this`, fitur ini jarang dipakai didalam `function` tetapi lebih sering di class, sebenarnya semua di Javascript memiliki `this` kalau this di scope global ya global, kalau didalam `function` hanya mengambil di scope `function` tersebut tadi.

```
console.log(this) // this disini

function test() {
    console.log(this); // dan this disini berbeda
}

test();
```

Lamda Function / Anonymous Function / Closure

Merupakan suatu tipe `function` yang langsung dijalankan tanpa ada nama, ini sering digunakan didalam `map` suatu `array` atau di suatu `function` dll.

```
function myFunc(closure) {
    console.log(typeof closure); // Ini akan men-print "function"
    closure() // dan ini untuk menjalankan closure-nya
}

myFunc(function () {
    console.log(1);
})
```

Konsep Arrow Function

`Arrow function` ini sama seperti dengan `function` biasa tetapi sedikit berbeda, jadi normal `function` ketika this muncul meta data `function` tersebut dan variabel lain, kalau di `Arrow Function` yang kena this malah scope atasnya, pendeklarasian `Arrow Function` ini sangat sederhana.

```
let nama = "Alifah";

// Tidak ada parameter
const myFunc = () => {
    console.log('MyFunc');

    //Perintah yang lain
}

// Hanya satu parameter
const myFunc2 = teks => {
    console.log(teks);

    //Perintah yang lain
}

// 2 Parameter dan seterusnya
const myFunc3  = (teks1, teks2) => {
    console.log(`${teks1} ${teks2}`); // ini merupakan string formating agar dapat memangil variabel kedalam string

    //Perintah yang lain
}

myFunc2(teks);
myFunc3("Zagar", "Dangure");

// Jika langsung return atau hanya mengerjakan satu instruksi saja, cara pemakaiannya masih sama
const myFunc4 = () => 1;
const myFunc5 = teks => console.log(teks);
const myFunc6 = (teks1, teks2) => console.log(`${teks1} ${teks2}`);
```

Ini akan berjalan dengan normal tetapi kalau kita memakai `this` kedalam `arrow function` ini akan mengambil scope diatasnya bukan `scope`-nya function tersebut.

```
this.myGlobalTeks = "Hello, World"; // mengisi this di scope global

function myFunc() {
    this.myFuncTeks = "Hi!";
    console.log(this);
}

const myFunc2 = () => {
    this.myFunc2Teks = "Test";
    console.log(this);
```

```
}

myFunc(); // akan mengambil metadata dan data yang tersimpan di `this` scope
function tersebut
myFunc2(); // akan mengambil metadata dan data yang tersimpan di `this` scope
diatasnya dalam konteks ini global
```

Class

`Class` itu merupakan Blueprint atau rancangan dan `class` sering digunakan untuk men-enkapsulasikan suatu data agar mudah di olah dan juga mudah dibaca, karena `Class` hanya suatu rancangan kita harus membuatnya terlebih dahulu setelah dibuat sering dipanggil `Object`, nah `Object` ini memiliki hubungan dengan `Class` yang menjadi basis pembuatan.

`Class` juga dapat memiliki `property` dan `method` tersendiri, `method` ini sama seperti `function` tetapi `this`-nya akan berkonteks kedalam class tersebut, dan `property` hanya variabel yang hidup di dalam class tersebut, setiap `class object` memiliki variabel tersendiri jadi kalau ada 2 object salah satu diganti data variabelnya tidak akan mempengaruhi yang lain, pendefinisian variabel dan method ada 2 cara, cara pertama ialah cara normal setiap object bisa memiliki value yang berbeda tetapi nama variabel/method yang sama atau yang global jadi semua yang memiliki `class` yang sama akan terpengaruh.

```
// Melakukan pendefinisian suatu class
class Motor {
    cc = 150;
    bahanBakar = 'bensin';
    tanki = 0;

    // Setiap inisialisasi ini akan dipanggil
    constructor(merk, totalTanki) {
        // Mengisi variabel merk di dalam object dengan data yang diberikan
        this.merk = merk;

        // Mengisi variabel tankiBahanBakar di dalam object dengan data yang
diberikan
        this.totalTanki = totalTanki;
        this.tanki = totalTanki;
    }

    jalan(kilometer) {
        this.tanki -= kilometer; // ini bermaksud this.tanki = this.tanki -
kilometer
    }
}

// Inisialisasi object dengan class motor
const mio = new Motor('Mio', 4);
const honda = new Motor('Honda', 5);

console.log(mio); // { cc: 150, bahanBakar: 'bensin', tanki: 4, merk: 'Mio',
totalTanki: 4}
```

```
console.log(honda); // { cc: 150, bahanbakar: 'bensin', tanki: 5, merk: 'Honda',
totalTanki: 5}

mio.jalan(2);

console.log(mio); // { cc: 150, bahanBakar: 'bensin', tanki: 2, merk: 'Mio',
totalTanki: 4}
console.log(honda); // { cc: 150, bahanbakar: 'bensin', tanki: 5, merk: 'Honda',
totalTanki: 5}
```

Konsep Inheritance

Konsep inheritance merupakan suatu cara untuk suatu class memiliki `property` dan `method` yang di inherit seperti induk dan anak, pasti anak memiliki hal yang sama dengan induknya, dan konsep ini dapat di lakukan berkali kali hingga berantai banyak sekali.

Di induk terkadang memiliki `constructor` sendiri jadi di class anaknya kita harus memanggil method `super` untuk memberikan parameter ke constructor induk agar dapat memiliki behavior yang sama karena di Javascript tidak bisa melakukan `overloading` terhadap `constructor`, `overloading` merupakan suatu cara `method` atau `function` memiliki instruksi yang berbeda ketika inputnya berbeda.

```
// Class sepeda sebagai induk
class Sepeda {
    constructor(merk) {
        this.merk = merk;
    }

    gayuh(kilometer) {
        console.log(`Perjalanan ${kilometer}`);
    }
}

// Class Motor sebagai anaknya
class Motor extends Sepeda {
    constructor(merk, bensinAwal) {
        super(merk);
        this.bensin = bensinAwal;
    }

    jalan(kilometer) {
        this.bensin -= kilometer;
    }

    isiBensin(liter) {
        this.bensin += liter;
    }
}

const motor = new Motor('Honda', 2);

console.log(motor); // Akan menprint { merk: 'Honda', bensin: 2 }
```

```
motor.jalan(2);

console.log(motor); // Akan menprint { merk: 'Honda', bensin: 0 }

motor.gayuh(2); // Akan menprint "Perjalanan 2"
```

Array

`Array` merupakan `Object` di Javascript kegunaannya untuk mengelompokan suatu data dengan bentuk data struktur `List`, `Array` di Javascript dapat menyimpan berbagai tipe data di dalam array yang sama tidak seperti bahasa yang lain yang hanya 1 jenis tipe data yang dapat dimasukan di setiap array.

Karena Array di Javascript merupakan `Object` kita dapat mendeklarasikan variabel `const` tetapi jika kita mau mengganti value keseluruhannya dengan `=` kita harus pakai `let` atau `var`.

Contoh :

```
const myArray = [1, 2, 3, 4];

// Menggunakan [] untuk mengakses data didalam array dengan menggunakan angka
sebagai indeks
console.log(myArray[0]); // Akan menprint 1
console.log(myArray[1]); // Akan menprint 2
console.log(myArray[2]); // Akan menprint 3
console.log(myArray[3]); // Akan menprint 4

// Kita juga dapat menganti isi array dengan []
myArray[0] = 10; // menganti isi indeks 0 menjadi 10
console.log(myArray[0]) // ini akan menprint 10
```

Array memiliki beberapa `method` yang nanti sering kita pakai

- `entries()`
  Ini digunakan ketika kita menggunakan `Iterator`.
- `push(value)`
  Ini akan mengambil parameter lalu ditambahkan kedalam `array`.
- `map(closure(value, index))`
  Ini akan mengambil `closure` lalu di eksekusikan, closure ini dapat menerima `value` lalu `index`, closure ini akan mengeksekusikan kode yang sama di setiap data di array hampir mirip dengan loop, closure ini dapat `return` yang nanti bisa ditangkap hasilnya dalam bentuk `array`.
- `forEach(closure(value, index))`
  Sama dengan map tetapi tidak boleh memiliki `return`.
- `fill(value, start, end)`
  Ini menerima parameter value lalu mengisikan ke seluruh indeks dari start sampai end, tetapi hanya bisa kalau indeksnya ada di dalam Array.
- `filter(closure(value, index, array))`
  Hampir mirip dengan loop `method` ini mengambil closure yang menerima `value`, lalu `index`, dan `array`

yang di loop, method ini digunakan untuk melakukan filter dan closure harus return tipe data `Boolean`, method ini akan return data yang memiliki hasil perbandingan didalam closure `true`.

- `reduce(closure(previousValue, currentValue, index, array))`
  Digunakan untuk melakukan operasi terhadap array sampai array itu habis dan method ini akan return 1 data saja tidak dalam bentuk array (Kecuali array didalam array)
- `concat(...value)`
  Ini menerima banyak input atau array untuk melakukan penambahan kedalam array tersebut.
- `join(seperator)`
  Method ini membuat array menjadi string, dapat diberi seperator agar membatasi setiap data didalam array.

Contoh pemakaiannya, beberapa contoh akan meninggalkan beberapa parameter karena tidak terlalu dibutuhkan

- `entries()`

```
const myArray = [1, 2, 3, 4, 5];

for(const [val, index] of myArray.entries()) {
    console.log(`${val} : ${index}`);
}
```

- `push(value)`

```
const myArray = [];

myArray.push(1);
myArray.push(2);
myArray.push(3);
myArray.push(4);

console.log(myArray); // Ini akan menprint "[1, 2, 3, 4]"
```

- `map(value, index)`

```
const myArray = [1, 2, 3, 4, 5];

const result = myArray.map((value, index) => `${index}: ${value}`)

console.log(result); // [ '0: 1', '1: 2', '2: 3', '3: 4', '4: 5' ]
```

- `forEach(value, index)`

```
const myArray = [1, 2, 3, 4, 5];
```

```
// Akan menprint `0: 1` lalu ganti baris `1: 2` dan seterusnya
myArray.forEach((value, index) => console.log(`${index}: ${value}`))
```

- fill(value, start, end)

```
const myArray = [1 ,2 ,3, 4, 5];

myArray.fill(1, 0, 4);

console.log(myArray); // [ 1, 1, 1, 1, 5 ]
```

- filter(closure(value, index))

```
const myArray = [1 ,2 ,3, 4, 5];

console.log(myArray.filter(value => value < 3)); // [1, 2]
```

- reduce(closure(previousValue, currentValue, index, array))

```
const myArray = [1 ,2 ,3, 4, 5];


const result = myArray.reduce((prev, curr) => prev + curr);

console.log(result) // 15
```

- concat(...value)

```
let myArray = [1, 2];

myArray = myArray.concat(4, 5, 6);

console.log(myArray); // [1, 2, 4, 5, 6]

myArray = myArray.concat(2, 3, 4);

console.log(myArray) // [ 1, 2, 4, 5, 6, 2, 3, 4 ]
```

- join(seperator)

```
let myArray = [1, 2];

console.log(myArray.join(' ')); // 1 2
```

Javascript Object

`Javascript Object` mirip dengan `Class` karena dapat men-enkapsulasi variabel dan function, tetapi bedanya kita tidak bisa memakai `this` oleh sebab itu agak sulit kalau membuat tanpa bantuan `Class`, cara menyimpan di `Javascript Object` sering disebut `Key Value Pair` yang berarti selalu ada kunci disetiap isi.

Contoh paling sederhana, kita membuat variabel siswa yang memiliki key `nama`, `no`, `sekolah`, dan function `printNama()` :

```
const siswa = {
    nama: 'Ghaza',
    no: 4,
    sekolah: "SMK Muh 1 Klaten Utara",
    printNama: () => console.log(siswa.nama),
}

console.log(siswa) // { nama: 'Ghaza', no: 4, sekolah: 'SMK Muh 1 Klaten Utara' }
siswa.printNama() // "Ghaza"
```