Modul praktikum - Minggu 11 - *Functions* bagian 2

Dosen pengampu: **Henokh Lugo Hariyanto**
Asisten mata kuliah: **Jein Ananda - (10221031)**; **Muhammad Aulia Rahman - (10221055)**

Tujuan:

- Mampu memahami *parameters* dan *arguments* dari suatu fungsi
- Mampu memahami nilai yang dihasilkan fungsi, maupun *side effect*
- Mampu memahami dasar *functional programming* di JavaScript

> Tips belajar bahasa pemrograman adalah mengetik ulang perintah yang kita temukan di buku atau di internet, lalu kita ubah-ubah untuk menguji pemahaman kita sudah tepat atau belum. Faktor bermain-main dan eksplorasi sangat diperlukan untuk memahami setiap perintah bahasa pemrograman yang kita pelajari. Setiap potongan kode di bawah dapat ditulis dalam berkas `.js` lalu dapat di-*running* dengan Node.js.

Pada sesi praktikum kali ini, kita akan melanjutkan pembahasan tentang fungsi. Sebelumnya kita telah mengenai berbagai istilah dalam fungsi JavaScript:

- function declaration
- function invocation
- function parameters and arguments

Kita juga telah mempelajari pendeklarasian fungsi dan pemanggilan fungsi dengan argument. Berikutnya yang akan kita jelaskan adalah beberapa trik untuk memanggil fungsi dengan kaidah argument tertentu dan keluaran fungsi dengan struktur tertentu.

Di bagian terakhir, kita akan mencoba untuk mempelajari paradigma pemrograman yang populer digunakan dalam JavaScript, yaitu *functional programming*.

Function arguments and parameters

Di bagian ini kita akan mempelajari bagaimana suatu arguments digunakan dalam suatu fungsi.

- *Optional parameters* dan *default parameters*
  Di bagian ini kita akan melihat bagaimana *optional parameter* dan *default parameters*. diimplementasikan ke dalam suatu fungsi. *Optional parameters* disini berarti kita bisa inputkan argument atau tidak ke *optional parameters* tersebut

  optional-params.js

```
// Fungsi berikut akan mengambil property name dari parameter obj
// dan menambahkan setiap property name ke dalam parameter arr.
function getPropertyNames(obj, arr) {
  // Apabila parameter arr tidak digunakan, maka didefinisikan dengan
  // nilai arr = []  (empty array)
  if (arr === undefined) {
    arr = [];
```

```
  }

  // Menambahkan setiap property name dari parameter obj ke parameter arr
  for (let property in obj) {
    arr.push(property);
  }

  return arr;
}

// Object berikut menggambarkan konfigurasi laman website yang diambil
// dari berkas .css
let obj = {
  'max-width': '36rem',
  'padding': ['0', '1rem'],
  'margin': ['3rem', 'auto', '6rem']
};
let arr = ['display'];      // konfigurasi lain yang hanya memuat property
name


// Pemanggilan fungsi pengumpulan property names dari parameters obj
// namun tanpa parameters arr (parameter arr disini opsional)
let objPropNames = getPropertyNames(obj);
console.log(objPropNames);    // => ['max-width', 'padding', 'margin']

// Pemanggilan fungsi pengumpulan property names dari parameter obj
// dengan optional parameter arr yang telah terisi element 'display'
let objPropNamesWithOption = getPropertyNames(obj, arr);
console.log(objPropNamesWithOption);
```

Menurut contoh di atas, kita bisa input-kan `arr` sebagai *optional parameter*. Jika kita tidak berikan input `arr` ke *optional parameter* (parameter urutan kedua), maka secara otomatis fungsi akan mendeklarasikan parameter `arr` sebagai *empty array*.

Berikutnya adalah contoh fungsi dengan *default parameter*

default-params.js

```
function getPropertyNames02(obj, arr=[]) {
  for (let property in obj) {
    arr.push(property);
  }

  return arr;
}

// Object berikut menggambarkan konfigurasi laman website yang diambil
// dari berkas .css
let obj = {
  'max-width': '36rem',
```

```
  'padding': ['0', '1rem'],
  'margin': ['3rem', 'auto', '6rem']
};
let arr = ['display'];      // konfigurasi lain yang hanya memuat property
name


// Pemanggilan fungsi pengumpulan property names dari parameters obj
// namun tanpa parameters arr (parameter arr disini akan diset secara
// otomatis sebagai array kosong)
let objPropNames = getPropertyNames(obj);
console.log(objPropNames);    // => ['max-width', 'padding', 'margin']

// Pemanggilan fungsi pengumpulan property names dari parameter obj
// dengan default parameter arr yang telah terisi element 'display'
let objPropNamesWithOption = getPropertyNames(obj, arr);
console.log(objPropNamesWithOption);
```

Terlihat dengan menggunakan *default parameter*, deklarasi fungsi lebih ringkas dan mudah dimengerti bahwa ketika *parameters* `arr` tidak diberikan sebagai argument dalam pemanggilan fungsi, maka secara *default* akan diberikan nilai awalan *empty array*.

- Rest paramaters
  Ketika jumlah argument yang harus kita berikan dalam suatu fungsi saat fungsi tersebut dipanggil ((*invocation*)) sangat banyak, maka *rest parameters* dapat membantu kita tanpa perlu meng-inputkan satu-satu. Bayangkan jika jumlah argument suatu fungsi yang dipanggil sejumlah satu juta arguments.

  Berikut contoh penggunaan *rest parameters* untuk fungsi pencarian nilai maksimum;

  rest-params-max-func.js

```
// Fungsi untuk menghitung nilai maksimumn. Terlihat disini digunakan
// rest parameters yang diawali oleh tiga titik.
// Output dari fungsi ini adalah nilai maksimum dari inputan yang diberikan
function max(...rest) {
  let maxValue = -Infinity;
  for (let n of rest) {
    if (n > maxValue) {
      maxValue = n;
    }
  }

  // Return the biggest
  return maxValue
}

// Disini kita memberikan input arguments lebih dari satu, namun
// di dalam pendeklarasian fungsi kita hanya cukup menggunakan rest
// parameter menggunakan ...rest
let result = max(1, 10, 100, 2, 3, 1000, 4, 5, 6);
console.log(result);
```

- Spread operator untuk *argument* fungsi yang dipanggil/*invoked*
  Ketika input argument yang disisipkan lebih dari satu ataupun banyak, misalnya 20 arguments, cara yang paling efisien adalah menggunakan spread operator di arguments (ingat kembali perbedaan arguments dan parameters). Berikut ini adalah pemanggilan fungsi menggunakan spread operator untuk input arguments lebih dari satu. Contoh fungsi yang digunakan tetap sama yaitu menghitung nilai maksimum

  spread-op-max-func.js

```
function max(...rest) {
  let maxValue = -Infinity;
  for (let n of rest) {
    if (n > maxValue) {
      maxValue = n;
    }
  }

  // Return the biggest
  return maxValue
}

let inputArr = [1, 10, 100, 2, 3, 1000, 4, 5, 6];
let result = max(...inputArr)   // Disini kita menggunakan spread operator
                                // pada argument inputArr
console.log(result);
```

Function as values

Karena fleksibiltas fungsi, kita dapat menggunakan fungsi sebagai suatu nilai.

- Pengunaan fungsi sebagai nilai suatu variable
  Pada pertemuan sebelumnya di bagian pendeklarasian fungsi, kita telah mengetahui bahwa fungsi dapat di-*assign* ke suatu variable. Jika yang kita *assign* adalah nama fungsinya saja, tanpa *arguments* maka yang terjadi adalah kita membuat referensi baru ke fungsi itu dengan nama variable tersebut.

  func-as-vals-variable-assign.js

```
// Berikut adalah fungsi untuk melakukan pengkuadratan
function square(x) { return x*x; }

// Kita assign fungsi tersebut (tanpa argument) ke variable s
let s = square;

// Kita dapat memanggil fungsi tersebut (invocation) dengan nama fungsi
square
// atau dengan variable s
console.log(square(16));
console.log(s(16));
```

- Penggunaan fungsi sebagai nilai dari suatu *property*
  Sedikit berbeda pada pertemuan sebelumnya yang menggunakan fungsi sebagai *method* suatu *object. Di bagian ini kita dapat memberikan nilai suatu *property* dengan suatu fungsi. *Property* dari object tersebut akan beralih fungsi sebagai nama fungsi atau bisa juga dilihat sebagai *method* dari object tersebut.

  func-as-vals-props-vals.js

```
// DIsini obj memiliki property square dengan nilai adalah
// fungsi pengkuadratan. Pemanggilan property *square* secara tidak
// langsung serupa dengan pemanggilan method suatu object
let obj = { square: function(x) { return x*x; } };

console.log(obj.square(16));
```

- Penggunaan fungsi sebagai nilai untuk *element* suatu array
  Di bagian ini hanyalah sedikit kemungkinan penerapan tidak umum dari fungsi yang digunakan sebagai element suatu array. Disini hanya ditambahkan sebagai kemungkinan bahwa kita dapat mengkonstruksi berbagai macam hal dengan fungsi sekalipun terlihat aneh

  func-as-vals-arr-element.js

```
// Fungsi dideklarasikan sebagai element pertama dari arr, lalu nilai
// argumentnya dapat diambil dari element kedua dari arr
let arr = [function(x) => { return x * x; }, 16]
console.log(arr[0](arr[1]));
```

- Penggunaan fungsi sebagai nilai untuk argument fungsi lain
  Penggunaan ini adalah yang paling sering digunakan dalam penyusunan program menggunakan JavaScript. Disini fungsi dijadikan sebagai argument untuk nilai parameter suatu fungsi yang lain. Sebagai contoh kita akan membuat suatu program kalkulator yang mana fungsi-fungsi aritmatik seperti kali, bagi, tambah, kurang akan menjadi input argument ke dalam fungsi kalkulator

  func-as-vals-input-args.js

```
// Kita definisikan fungsi aritmatika dari fungsi kalkulator
function add(x, y) { return x + y; }
function subtract(x, y) { return x - y; }
function multiply(x, y) { return x * y; }
function divide(x, y) { return x / y; }

// Berikut adalah fungsi yang mendefinisikan kalkulator
function calculate(operator, operand1, operand2) {
  return operator(operand1, operand2);
}
```

```
// Menghitung nilai ekspresi  (1 + 2) * (3 - 2) / 3;
let result = calculate(
  divide,
  calculate(
    multiply,
    calculate(
      add, 1, 2),
    calculate(
      subtract, 3, 2)),
  3);
console.log(result);
```

Pada program di atas kita melihat bahwa argument kedua dan ketiga saat proses pembagian (`divide`), merupakan pemanggilan fungsi `calculate`. Sampai pada akhirnya pada level operasi terdalam (yang memiliki kurung, yaitu penjumlahan dan pengurangan). Fungsi aritmatika: `add`, `subtract`, `multiply`, dan `divide` digunakan sebagai input argument fungsi `calculate`. Demikian juga fungsi `calculate` digunakan sebagai input arguments fungsi `calculate`

Functional programming

Dibagian ini kita hanya menyentuh permukaan dari paradigma *functional programming*. Disini kita hanya menerapkan aplikasi *functional programming* untuk memproses *arrays*

Berikut adalah implementasi perhitungan standard deviasi menggunakan *looping*. Secara singkat, standard deviasi mengukur seberapa jauh distribusi data terhadap nilai rata-rata.

compute-stddev-conventional.js

```
const formatStr = require('@stdlib/string-format')

let data = [1, 1, 3, 3, 5, 7, 7, 9, 9];

// Menghitung rata-rata dengan cara menjumlahkan semua element
// dan membaginya dengan banyaknya element
let total = 0;
for (let i = 0; i < data.length; i++) {
  total += data[i];
}
let mean = total / data.length;

// Menghitung standard deviasi, pertama menghitung jumlahan
// kuadrat deviasi setiap element terhadap rata-rata.
// Setelah itu di bagi dengan (jumlah data - 1)
total = 0;
for (let i = 0; i < data.length; i++) {
  let deviation = data[i] - mean;
  total += deviation * deviation;
}
let stddev = Math.sqrt(total/(data.length - 1));
console.log(formatStr('%.2f', stddev));     // => 3.16
```

Berikut ini adalah implementasi program di atas menggunakan paradigma fungsional. Disini kita lihat tidak ada *control structure* loop lagi, dan semuanya diganti menjadi fungsi `map` dan `reduce`. Kita akan menjelaskan lebih detail di sesi kelas terkait `map` dan `reduce`.

compute-stddev-func-prog.js

```
const formatStr = require("@stdlib/string-format");

const map = function(a, ...args) { return a.map(...args); };
const reduce = function(a, ...args) { return a.reduce(...args); };

const sum = (x, y) => x + y;
const square = x => x * x;

let data = [1, 1, 3, 3, 5, 7, 7, 9, 9];
let mean = reduce(data, sum)/data.length;
let deviations = map(data, x => x - mean);
let stddev = Math.sqrt(
  reduce(map(deviations, square), sum)/(data.length - 1));
console.log(formatStr("%.2f", stddev));   // => 3.16
```

Tugas (Exercise - 07)

> Laporan harus ditulis dan dikumpulkan dalam bentuk berkas *markdown* atau berkas berekstensi `.md`. Apabila laporan memuat lebih dari satu berkas, misal memuat berkas gambar `.png` atau `.jpg`, maka berkas disatukan menjadi berkas `.zip`.
> **PASTIKAN** berkas `md` sudah dilakukan *preview*, sehingga kode *markdown* bisa di-*preview* dengan benar.
> Format penamaan file: `NIM_NAMA.md` atau `NIM_NAMA.zip` (boleh nama lengkap atau nama panggilan).
>
> **Contoh format laporan atau jawaban (`NIM_NAMA.md`)**
> Nama: [NAMA LENGKAP]
> NIM: [NIM]
>
> 1. (Jawaban nomor 1)
> 2. (Jawaban nomor 2)

1. [30 poin] Ceritakan kembali apa yang sudah dipelajari hingga minggu ke-10. Adakah hal yang terlewat dari *slides* ataupun sesi praktikum. Buat daftar beberapa konsep yang mungkin belum jelas makna, alur dan penggunaannya. Maksimal 10 konsep JavaScript yang masih belum dipahami.

   Lalu coba untuk dipelajari kembali dan dijelaskan menggunakan bahasa sendiri (disertai contoh).

   Maksimal uraian adalah 1000 kata.

2. [70 poin] Buatlah sebuah fungsi untuk menghitung besar pengeluaran Yasuke (lihat slide di pertemuan tentang *Object*) diberikan input *weekly expense* Yasuke selama seminggu.

Berikut data *weekly expense* Yasuke selama seminggu.

```
Monday, March 21:
* Breakfast at a local food stall - 20,000
* Transportation to work - 10,000
* Lunch with colleagues - 50,000
* Snacks from a convenience store - 15,000
* Dinner at home - 30,000

Tuesday, March 22:
* Breakfast at home - 10,000
* Coffee from a cafe - 25,000
* Groceries for the week - 150,000
* Lunch from a food delivery app - 35,000
* Snack from a street vendor - 10,000

Wednesday, March 23:
* Breakfast at home - 10,000
* Transportation to work - 10,000
* Lunch from a food court - 40,000
* Afternoon tea from a bakery - 20,000
* Dinner at home - 40,000

Thursday, March 24:
* Breakfast at home - 10,000
* Coffee from a cafe - 25,000
* Transportation to work - 10,000
* Lunch from a food delivery app - 35,000
* Snacks from a convenience store - 10,000
* Dinner at home - 40,000

Friday, March 25:
* Breakfast at home - 10,000
* Transportation to work - 10,000
* Lunch from a food court - 40,000
* Afternoon snack from a street vendor - 15,000
* Dinner at a friend's house - 0 (free)

Saturday, March 26:
* Brunch with friends at a cafe - 80,000
* Transportation to a shopping mall - 20,000
* Shopping for clothes and accessories - 300,000
* Dinner at home - 50,000

Sunday, March 27:
* Breakfast at home - 10,000
* Transportation to a park - 15,000
* Entrance fee to the park - 5,000
* Lunch from a food stall at the park - 30,000
* Snacks and drinks from a vending machine - 10,000
* Dinner at home - 40,000
```

Fungsi yang dibuat juga harus mampu menghitung rata-rata pengeluaran harian dari Yasuke.