Modul praktikum - Minggu 05 - *Statements*

Dosen pengampu: **Vika Fitratunanny Insanittaqwa S.Kom., M.Kom**
Asisten mata kuliah: **Bayu Adhitya Wibisana - (10191015)**

Tujuan:

- Mampu memahami *statements* dasar yang dimiliki oleh JavaScript.
- Mampu menggunakan *statements* untuk program sederhana.

> 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.

Statement atau pernyataan merupakan suatu kalimat atau perintah untuk berkomunikasi dengan JavaScript. Sama halnya dengan kalimat yang dipisahkan oleh tanda titik, *statement* di dalam JavaScript juga dipisahkan oleh tanda titik koma.

Expression dievaluasi untuk menghasilkan nilai, sedangkan *statement* dieksekusi/dijalankan untuk membuat sesuatu hal terjadi (yaitu berubah *state* atau keadaan). Contoh seperti penentuan suatu tahun apakah tahun kabisat atau bukan.

Secara garis besar, suatu program JavaScript hanyalah kumpulan berbagai *statements* yang dijalankan dengan urutan tertentu. Urutan ini pun dapat dikendalikan jalannya dengan *statement* khusus yang disebut *control structures*.

Control structures inilah yang merupakan dasar dari pemrograman terstruktur dan yang memberikan struktur penyusunan *statements* sehingga membentuk struktur program yang jelas, ringkas dan efisien.

Ada dua macam *statements* dalam JavaScript:

- Expression statements
  Merupakan ekspresi-ekspresi yang dapat dijalankan tanpa tambahan ekspresi yang lain dan memiliki *side-effect* (ingat pertemuan sebelumnya tentang *side-effect*).

  Contohnya: pemberian nilai pada suatu variable (*assignment*), pemanggilan fungsi (akan kita bahas di pertemuan tentang fungsi).

- Declaration statements

  Contohnya: mendeklarasikan suatu variabel baru (`let a = 2`), mendefinisikan suatu fungsi baru (akan kita bahas di pertemuan tentang fungsi)

Pada kategori di atas *control structure* boleh diikutsertakan apabila kita menganggap *control structures* merupakan *statements* khusus.

Berikutnya akan dibahas *expression statements*, *control structures* (*conditionals*, *loops*, and *jumps*), dan ditutup dengan *statement* tambahan yang masih berguna dalam penyusunan program JavaScript.

Expression Statements

Berikut merupakan *expression statements* yang dibentuk dari *expression*

expression-statement.js

```
let greeting, name, i = 1;

greeting = "Hello " + name;   // statement yang tersusun dari assignment
                              // expression
i *= 3;                       // statement yang tersusun dari compound operator
                              // (assignment operator dan arithmetic operator)
console.log(greeting, i);

cx = Math.cos(3.14);          // statement yang merupakan hasil dari
                              // pemanggilan fungssi
console.log(cx);
```

Compound and Empty Statements

Kumpulan dari beberapa *expression statements* dapat disatukan oleh *statement block* (kurung kurawal) sehingga membentuk satu *statement* baru yang disebut *compound statement*.

compound-statement.js

```
{
  x = Math.PI;
  cx = Math.cos(x);
  console.log("cos(pi) = " + cx);
}
```

Disamping kita bisa menyusun beberapa *expression statements* menjadi satu kesatuan, kita bisa mendefinisikan *empty statement* yang hanya terdiri dari satu karakter titik koma (`;`). Salah satu penggunaannya adalah dalam kasus perulangan

empty-statement-in-forloops.js

```
// program berikut akan mengubah semua nilai array di a menjadi 0
let a = [1, 2, 3, 4, 5];
for (let i = 0; i < a.length; a[i++] = 0) {
  ;   // empty statement
}
console.log(a);
```

Pengkondisian (*conditional*)

- `if` and `if-else`
  Merupakan *control structure* untuk mengendalikan statement sehingga program JavaScript dapat melakukan pemilihan keputusan (*decision*).

  conditional.js

```
let isNextWeekExam = true;

// Hanya `if`, ketika kondisi bernilai salah maka akan dilewati
if (isNextWeekExam) {
  console.log("Remember to study for the exam.");
}

// `if` dengan pasangan `else`
if (isNextWeekExam) {
  console.log("You need to study now.");
} else {
  console.log("No need to study.");
}
```

- `else if`
  Jika kondisi yang tidak terpenuhi (*false* condition) masih memiliki kondisi yang harus dipenuhi maka kita bisa menggunakan `else if` untuk menggambarkan kondisi tersebut. Kita dapat terus melakukan penambahan `else if` sebanyak kondisi yang ingin kita wakili

  many-else-if.js

```
let exam = "programming";
if (exam === "calculus") {
  console.log("Practice calculation of integral");
} else if (exam === "statistics") {
  console.log("Review the lecture material");
} else if (exam === "programming") {
  console.log("Rewrite and understand the code");
} else {
  console.log("You don't have any exam, yeeey");
}
```

- `switch`
  Control structure ini mirip seperti `if - else if - else if - else ...` namun lebih ringkas dan berlaku untuk pengujian kondisi yang bernilai string atau integer. Untuk pengujian kondisi yang lebih umum bisa tetap menggunakan `if else`

  switch-demo.js

```
let exam = "programming";
switch (exam) {
```

```
    case "calculus":
      console.log("Practice calculation of integral");
      break;
    case "statistics":
      console.log("Review the lecture material");
      break;
    case "programming":
      console.log("Rewrite and understand the code");
      break;
    default:
      console.log("You don't have any exam, yeeey (or you forget them :D)");
  }
```

Perulangan (*loops*)

Kemampuan repetitif (mengerjakan hal yang berulang-ulang) sangatlah cepat dibanding manusia, itulah yang membuat komputer menjadi berguna. Kemampuan repetitif secara fundamental diwakili oleh *control structure* perulangan (*loops*). Pada bagian ini kita akan mempelajari berbagai macam perulangan yang disediakan oleh JavaScript.

- `while`
  Merupakan *control structure* yang akan menjalankan *statements* berulang-ulang hingga kondisi yang diberikan tidak terpenuhi lagi

  while-demo.js

  ```
  let count = 0;
  while (count < 5) {
    console.log(count);
    count++;
  }
  ```

- `do/while`
  Sama seperti halnya *control structure* `while`, namun *statements* akan dijalan minimal sekali baru setalah itu dilakukan pengujian apakah kondisi perulangan terpenuhi atau tidak

  do-while-demo.js

  ```
  let count = 0;
  do {
    console.log(count);
    count++;
  } while (count < 5);
  ```

- `for`
  Control structure `for` kadang lebih nyaman untuk digunakan karena kita memiliki kendali untuk setiap perulangan yang dijalankan dan variable *counter* yang sudah menjadi bagian dari *control structure* ini

(berbeda dengan `while`, kita masih perlu mendefinisikan variable `count` di luar `while`). Berikut contoh penggunaan `for`

for-demo.js

```
for (let count = 0; count < 5 ; count++) {
  console.log(count);
}
```

- `for/of`
  Merupakan bentuk yang lebih ringkas untuk *control structure* yang khusus digunakan untuk meng-iterasi (melakukan perulangan) elemen-elemen (untuk array) atau *properties* (untuk suatu object). Disini kita hanya menyinggung sedikit. Secara lebih lengkap akan dijelaskan pada pertemuan tentang array

  for-of-demo.js

  ```
  for (let count of  [0, 1, 2, 3, 4]) {
    console.log(count);
  }
  ```

- `for/in`
  Sama seperti `for/of` namun diperuntukkan untuk melakukan perulangan *properties* dari suatu object. Jika dipaksakan untuk digunakan dalam suatu array, maka yang tercetak atau terpanggil sebagai variable counter adalah index disetiap perulangan.

  for-in-demo.js

  ```
  // It will print the indices not the values
  for (let count in ["a", "b", "c", "d", "e"]) {
    process.stdout.write(count + " ");
  }
  console.log();

  // `for-in` is suitable for an object
  let obj = {0: "a", 1: "b", 2: "c", 3: "d", 4: "e"};
  for (let count in obj) {
    process.stdout.write(obj[count] + " ");
  }
  console.log();
  ```

Jumps

Control structure ini merupakan statement yang mampu melakukan lompatan menuju baris-baris kode secara spesifik. Ada banyak cara untuk melakukan lompatan dan setiap cara tersebut berguna untuk kasus-kasus tertentu

- Labeled statements
  Setiap *statement* dapat diberi label sehingga dapat kita jadikan acuan lompatan. Sebagai contoh adalah program berikut:

  labelled-statements-demo.js

```
// We break after i === 1 and j === 1,
// but i continue to proceed for different i
outerloop:
for (let i = 0; i < 3; i++) {
  innerloop:
  for (let j = 0; j < 3; j++) {
    console.log(`i = ${i}, j = ${j}`);
    if (i === 1 && j == 1) {
      break innerloop;
    }
  }
}

console.log()

// We break after i == 1 and j === 1 and also stop the outerloop
outerloop:
for (let i = 0; i < 3; i++) {
  innerloop:
  for (let j = 0; j < 3; j++) {
    console.log(`i = ${i}, j = ${j}`);
    if (i === 1 && j == 1) {
      break outerloop;
    }
  }
}
```

- `break`
  statement `break` digunakan untuk menghentikan perulangan atau poses percabangan seperti di *statement* `switch`. Beberapa contoh sudah diberikan pada bagian sebelumnya.

- `continue`
  statement ini merupakan *statement* yang mirip dengan `break` namun kegunaannya berlawan. `continue` dipakai untuk melanjutkan perulangan. Umumnya digunakan untuk melakukan pengabaian (*skip*) satu atau lebih perulangan.

  Dapat juga digunakan untuk mengabaikan elemen suatu array yang tidak terdefinisi, sehingga berguna untuk menghindari error ketika membaca suatu data array

  example-continue.js

```
let data = [1, 2, 3, undefined, 4, 5];
let total = 0;
for (let i = 0; i < data.length; i++) {
```

```
  if (!data[i]) {
    continue;         // tidak memproses `undefined` elemen
  }
  total += data[i];
}

console.log(`total = ${total}`);
```

- `return`
  Statement jump ini digunakan untuk mengakhiri suatu fungsi dan menyatakan apa yang ingin kita keluarkan sebagai hasil dari fungsi tersebut. Secara lebih lengkap akan kita bahas di pertemuan tentang fungsi.

- `yield`
  Statement jump ini digunakan sebagai versi `return` *statement* untuk objek iterator dan generator. Dipilih digunakan `yield` dengan alasan efisiensi eksekusi suatu fungsi hanya akan dieksekusi ketika elemennya dipanggil. Pada kuliah ini kita tidak sampai mebahas objek iterator dan generator.

- `throw`
  Statement jump ini digunakan untuk memberikan petunjuk secara eksplisit bahwa di baris `throw` *statement* dipanggil, terjadi suatu hal yang tidak diinginkan (biasanya disebut *exception*). Umumnya perintah `throw` digunakan secara bersamaan dengan `try/catch/finally` *statements". Contoh akan diberikan pada bagian berikutnya.

- `try/catch/finally`
  Merupakan mekanisme untuk menangkap *exception* yang terjadi selama program dieksekusi (dijalankan)

  Bagian `try` (atau disebut klausa `try`) merupakan bagian tempat indikasi berlangsungnya *exception*.

  Klausa `catch` digunakan untuk menjalankan *statement* lain saat *exception* terjadi di klausa `try`.

  Terakhir klausa `finally` merupakan opsional blok (bisa dipakai bisa tidak) yang akan tetap dijalankan apapun kondisi *exception* dari klausa `try`.

  Berikut contoh yang cukup sederhana dari *exception handling mechanism* menggunakan `try/catch/finally` *statements*.

  try-catch-finally.js

```
// In the textbook, they use alert() and prompt() which are the functions
// that are available in browser not in Node.
const prompt_sync = require('prompt-sync')();

function factorial(x) {
  // If the input argument is invalid, throw an exception!
  if (x < 0) throw new Error("x must not be negative");

  // Otherwise, compute a value and return normally
  let f;
```

```
  for (f = 1; x > 1; f *= x, x--) /* empty */;
  return f;
}

try {
  // Ask the user to enter a number
  let n = Number(prompt_sync("Please enter a positive integer: "));

  // Compute the factorial of the number, assuming the input is valid
  let f = factorial(n);

  // Display the result
  console.log(n + "! = " + f);
} catch(ex) {   // If the user's input was not valid, we end up here
  console.log(ex.name + ": " + ex.message); // Tell the user what the error
is
}
```

Program di atas dijalankan dengan memberi kemungkinan input bilangan bulat positif atau negatif. Bilangan bulat negatif akan memberikan *exception*.

Miscellaneous Statements

- `"use strict"`
  Merupakan *statement* yang secara khusus mengubah mode dari JavaScript ke *strict mode*. *Strict mode* merupakan mode yang mana beberapa statement bawaan JavaScript dapat digunakan atau tidak. Mereka tidak dapat digunakan karena adanya defisiensi (kode tidak efisien). *Strict mode* juga menjamin adanya pengecekan error lebih agresif sehingga program JavaScript yang kita buat akan mengalami peningkatan dalam hal keamanan program tersebut (mengurangi resiko di-*hack*).

  Berikut contoh perbedaan menggunakan `"use strict"` dan tidak.

  non-strict-mode.js

```
// Non-strict mode
let x = 10;
delete x;
console.log(x);   // => undefined
```

  use-strict-mode.js

```
// Strict mode
"use strict";
let x = 10;
delete x;         // => SyntaxError (we cannot perform delete)
console.log(x);
```

Tugas (Exercise - 03)

> 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. (**10 poin**) Ceritakan dalam 200 kata tentang hal yang telah kalian pelajari di sesi praktikum ini.
2. (**90 poin**) Dari contoh di bagian perulangan, buatlah program JavaScript untuk mencetak gambar kartu berikut:
    - Kartu wajik dengan ukuran 1

```
#---#
| * |
#---#
```

    - Kartu wajik dengan ukuran 2

```
#-----#
|  *  |
| *** |
|  *  |
#-----#
```

    - Kartu wajik dengan ukuran 3

```
#-------#
|   *   |
|  ***  |
| ***** |
|  ***  |
|   *   |
#-------#
```

    - Kartu wajik dengan ukuran 4

```
#---------#
|    *    |
|   ***   |
|  *****  |
| ******* |
|  *****  |
|   ***   |
|    *    |
#---------#
```

Petunjuk:

- Gunakan *package* `prompt-sync` sehingga *user* dapat memberikan input ukuran kartu yang ingin ditampilkan (batasi sampai ukuran 6).
- Program tidak perlu terlalu rumit, cukup menggunakan pemahaman yang telah kalian dapatkan selama minggu ke-01 hingga minggu ke-05.