Modul praktikum - Minggu 04 - *Types, Values, and Variables*

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

Tujuan:

- Mampu memahami *(data) types*, *values*, dan *variables* dalam JavaScript
- Mampu menggunakan berbagai macam tipe dan nilai *variables*
- Mampu membedakan *immutable primitive types* dan *mutable reference types*
- Mampu melakukan konversi antara satu *type* ke *type* yang lain.
- Mampu mendeklarasikan variable, inisialisasi konstanta.

> 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 pertemuan ini kita akan membahas *types*, *values*, dan *variables*. Istilah *types* merupakan wakilan dari nilai (*values*) yang dapat kita manipulasi (dijumlahkan, digabungkan, dsb.). Jika dalam bahasa pemrograman Python biasanya disebut *data types*, tapi dalam bahasa pemrograman JavaScript lebih sering disebut *types*. Mengetahui *types* yang tepat untuk nilai yang ingin kita wakilkan merupakan syarat untuk membangun suatu program yang fleksibel.

Maksud fleksibel disini adalah kita dapat memanipulasi nilai, mengubah satu *type* ke *type* yang lain.

Proses pengubahan nilai (*value*) dan *type* ini didukung dengan kemampuan *variables* di dalam JavaScript. *Variable* adalah suatu nama yang menunjuk kepada suatu nilai yang nilainya dapat kita ubah-ubah. Pada pertemuan sebelumnya kita telah belajar mengenai bagaimana penggunaan *identifier* untuk menamakan *variable*. Proses modifikasi yang fleksibel ini (yang hampir dimiliki oleh berbagai macam bahasa pemrograman) yanng dimiliki oleh *variable* membuat JavaScript dapat digunakan hampir di segala permasalahan komputasi.

Beberapa topik yang akan kita kaji dalam sesi praktikum ini adalah:

- Beberapa istilah penting dalam *types*
- *Type*: `Numbers`
- *Type*: `Text`
- *Type*: `Boolean`
- *Special type*: `null` and `undefined`
- *Type*: `Symbols`
- *Value*: Global object
- *Type conversions*
- *Variable declaration and assignment*

Beberapa istilah penting dalam *types*

Type dalam JavaScript dapat dibagi menjadi dua kategori:

- *primitive types*: numbers, strings of text, boolean,
- *object types*: selain primitive types

Demikian pula untuk *values* dalam JavaScript terdapat dua kategori:

- *primitive values*: numbers, string, boolean, symbol, `null`, `undefined`
- *object values*: (biasanya disebut object) selain primitive values

Contoh object: `Object`, `Array`, `Set`, `Map`, `RegExp`, `Date`, `Error`, `Function`

Object type memiliki sifat *mutable* artinya nilai dari tipe tersebut dapat kita ubah-ubah. Berbeda dengan primitive type yang memiliki sifat *immutable* (tidak dapat kita ubah-ubah)

Berikut potongan kode untuk mengetahui *types* beberapa *primitive type* dan *object type*

several-types.js

```
console.log(2, typeof(2));
console.log('"apple"', typeof("apple"));
console.log(true, typeof(true));
console.log("Symbol()", typeof(Symbol()));

console.log();
console.log("{}", typeof({}));
console.log("[]", typeof([]));
console.log("new Array()", typeof(Array()));
console.log("new Set()", typeof(new Set()));
console.log("new Map()", typeof(new Map()));
console.log("new RegExp()", typeof(RegExp()));
console.log("new Date()", typeof(new Date()));
console.log("new Error()", typeof(Error()));
```

Ada alasan mengapa digunakan reserved keywords `new` untuk objek di atas adalah objek harus diinisiasi terlebih dahulu sebelum digunakan (meskipun beberapa objek seperti `Array()`, `RegExp()`, dan `Erro()` tetap menghasilkan keluaran yang sama).

Types: Number

Merupakan tipe numerik yang sering paling digunakan dalam bahasa pemrograman JavaScript.

- *Integer literals*
  Merupakan data yang berbentuk integer yang dapat di-inputkan secara langsung dalam program JavaScript

  integer-literals.js

```
let a = 0;
let b = 3;
let c = 1_000_000;  // tanda garis bawah hanya untuk pemisah
                    // tidak mengubah nilai
console.log(a, b, c);

let d = 0xff;       // => 255: (f*16^1 + f*16^0) = (16*16 + 16*1)
let e = 0x15EC41;   // => 1436737: (1*16^5 + 5*16^4 + E*16^3 + C*16^2 +
4*16^1 + 1*16^0)
                    //             = (1*16^5 + 5*16^4 + 14*16^3 + 12*16^2 +
4*16 + 1*1)
console.log(d, e);

let f = 0b10101;    // => 21: (1*16 + 0*8 + 1*4 + 0*2 + 1*1)
let g = 0o377;      // => 255: (3*64 + 7*8 + 7*1)
console.log(f, g);
```

- *Floating-Point Literals*
  Merupakan data yang berbentuk desimal yang dapat di-inputkan secara langsung dalam program JavaScript

  floating-point-literals.js

```
let a = 3.14;
let b = 2345.6789;
let c = .333_333_333_333_333_333    // tanda garis bawah hanya pemisah
                                    // tidak mengubah nilai
let f = 6.02e23;        // 6.02 × 10²³
let g = 1.473822E-32    // 1.4738223 × 10⁻³²

console.log(a, b, c, f, g);
```

Terdapat *special value* dalam *type: numeric* yaitu `NaN`. `NaN` digunakan dalam JavaScript untuk mewakili hasil pembagian angka dengan nol ,penarikan akar bilangan negatif, dan *parsing* bilangan bulat namun input yang diberikan adalah string.

Type: Text

Merupakan tipe *string* yang digunakan untuk mewakili teks.

- *String literals*
  Merupakan data yang berbentuk string yang dapat di-inputkan secara langsung dalam program JavaScript

  string-literals.js

```
let a = "";         // String kosong dehgan nol karakter
let b = "testing";
```

```
let c = "3.14";           // angka namun dalam bentuk string
let d = 'name="Hasan"';   // ekspresi dalam string

let f = "Wouldn't you prefer to learn JavaScript?";
let g = `"She said 'hi'", he said.`;        // menggunakan backticks

console.log(a);
console.log(b);
console.log(c);
console.log(d);
console.log(f);
console.log(g);
```

- *Escape sequences* di *string literals*
  Merupakan urutan karakter untuk bisa menggunakan karakter yang telah digunakan oleh JavaScript. Sebagai contoh seperti halnya JavaScript telah menggunakan tanda petik dua sebagai awalan dan akhiran suatu string maka dengan *escape sequences*, kita dapat menggunakan secara literal (bukan sebaga penanda awalan dan akhiran string) dalam suatu string

  escape-seq.js

```
let a = "\"She said \'hi\'\", he said.";
let b = "This is first line;\nThis is second line";

console.log(a);
console.log(b);
```

  Jika mengingat pertemuan sebelumnya terkait pembahasan Unicode, kita mengetahui untuk meng-input karakter Unicode, kita menggunakan awalan `\u`. Awalan ini juga termasuk *escape sequences*.

- *Template Literals* Serupa dengan *string literals* namun kita dapat menyediakan *template* dengan cara mensubstitusikan nama variable yang menyimpan suatu nilai. Khusus *template literals*, *delimiters* (tanda awalan dan akhiran suatu string) menggunakan *backticks* (`).

  template-literals.js

```
let name = "Randy";
let greeting = `Selamat datang ${name}.`;

console.log(greeting);
```

  Pada kode di atas, `${name}` akan dieksekusi menjadi "Randy" dan string `Randy` akan ditambahkan pada string sebelumnya (`Selamat datang `)

Type: Boolean

Merupakan *type* yang digunakan untuk mewakili kondisi Boolean (benar atau salah). Untuk *type* ini hanya memiliki dua kemungkinan nilai yaitu (`true` dan `false`).
Di dalam JavaScript dikenal istilah *falsy*, yang artinya untuk suatu nilai dalam JavaScript bernilai `false` jika dikonversi ke *type* Boolean.

Berikut adalah nilai yang akan dikonversi menjadi `false` jika dikonversi ke *type* Boolean.

falsy-values.js

```
let a = undefined;
let b = null;
let c = 0;
let d = -0;
let f = NaN;    // NaN: Not a Number
let g = "";

console.log(
  Boolean(a), Boolean(b), Boolean(c),
  Boolean(d), Boolean(f), Boolean(g));
```

Special type: `null` and `undefined`

`null` umumnya digunakan sebagai wakilan untuk suatu keadaan tanpa nilai. Misal untuk *type* data numerik kita bisa membuat keadaan tanpa nilai adalah 0. Namun untuk *type* yang lebih umum seperti string, atau *object type*, ada banyak pilihan untuk menentukan keadaan nol atau tanpa nilai. Disitulah *null* hadir untuk memperumum keadaan tanpa nilai untuk berbagai *object type*.

Sedangkan `undefined` merupakan keadaan saat suatu nama *variable* belum kita tentukan jenis *type*-nya. Atau suatu keadaan yang kita tidak tahu dia harus seperti apa.

Berikut tabel perbedaan antara `null` dan `undefined`

`null`	`undefined`
tidak memiliki nilai, dan **sengaja digunakan sebagai *value***	dideklarasikan, tapi **belum diberikan *value***
jenis *type*-nya adalah `object`	jenis *type*-nya adalah `undefined`
perilaku aritmatik seperti `0`	perilaku aritmatik seperti `NaN`

Pendeklarasian adalah proses memberikan *type* dari suatu variable dan bisa disertai pemberian *value* atau tidak.

Berikut adalah pengujian tabel perbedaan di atas

null-and-undefined.js

```
let a = null;
let b = undefined;
```

```
console.log("typeof(a) : ", typeof(a));
console.log("typeof(b) : ", typeof(b));

console.log("a + 0: ", a + 0);
console.log("a - 0: ", a - 0);
console.log("a + 5: ", a + 5);
console.log("b + 0: ", b + 0);
console.log("b - 0: ", b - 0);
console.log("b + 5: ", b + 5);

console.log("a === 0", a === 0);
console.log("b === NaN", b === NaN);
```

Type: Symbols

Digunakan untuk untuk penamaan *key* atau *property name* dalam *object type*. Kita akan membahas tentanng Symbol di pertemuan tentang *Object type*.

Berikut adalah contoh bahwa `object type` Symbols tidak akan pernah menghasilkan nilai yang sama ketika didefinsikan dengan nilai yang sama

symbols-demo.js

```
let a = Symbol("property_a");
let b = Symbol("property_a");

console.log(a === b);
```

Value: Global object

- Ketika program interpreter JavaScript (contohnya: Node.js atau web browser) mulai berjalan, program tersebut akan membuat suatu objek global dan memberikan sekumpulan *properties* yang mendefinisikan beberapa nilai berikut:

    1. *Global constants*, seperti: `undefined`, `Infinity`, dan `NaN`.
    2. *Global functions*, seperti: `isNaN()`, `parseInt()`, dan `eval()`
    3. Fungsi konstruktor, seperti: `Date()`, `RegExp()`, `String()`, `Object()`, `Array()`.
    4. Objek global, seperti: Math dan JSON.

    Seperti yang telah dibahas dalam sesi kuliah, pendeklarasian variable yang bersifat global sangat susah untuk ditelusuri perubahannya apabila terjadi *bug*. Global object digunakan untuk suatu nilai yang konstan dalam satu program. Kita akan kembali ke pembahasan global object pada pertemuan tentang *Object type* dan *Function*.

Type conversions

- Perlu diketahu JavaScript dapat melakukan pengubahan *type* suatu *literal* dalam suatu operasi aritmatik secara otomatis

autoconversion-type-.js

```
console.log(10 + " apples");    // => "10 appels": Bilangan 10 dikonversi
menjadi suatu string
console.log("7" * "4");         // => 28: kedua string dikonversi ke
bilangan

let n = 1 - "x";                // n === NaN; string "x" tidak dapat
dikonversi ke bilangan
console.log(n + "thing(s)");    // => "NaN thing(s)": NaN dikonversi ke
string "NaN".
```

- Beberapa fungsi konversi eksplisit dalam JavaScript

explictit-conversion-func.js

```
let a = Number("3");
let b = String(false);
let c = Boolean([]);                // Mengkonversi array kosong ke Boolean
*type*
let d = "0b" + (2023).toString(2);  // Argumen toString menunjukkan basis
bilangan

console.log(a, b, c, d);

let f = (2023).toFixed(3)     // Argument toFixed menunjukkan banyaknya
angka di belakang titik
console.log(f);

let g = (0.001).toExponential(3)    // Argument toExponential menunjukkan
banyaknya angka di belakang titik
console.log(g);

let h = (2023).toPrecision(6);    // Argument toPrecision menunjukkan
banyaknya karakter yang ingin ditampilkan
console.log(h);
```

- Untuk melakukan konversi string menuju numbers ada dua fungsi yang sering digunakan (`parseInt` dan `parseFloat`).

number-parsers.js

```
console.log(parseInt("3 blind mice"));      // => 3
console.log(parseFloat(" 3.14 meters"));    // => 3.14
console.log(parseInt("-12.34"));            // => -12
console.log(parseInt("0xFF"));              // => 255
console.log(parseInt("0xff"));              // => 255
console.log(parseInt("-0XFF"));             // => -255
```

```
console.log(parseFloat(".1"));              // => 0.1
console.log(parseInt("0.1"));               // => 0
console.log(parseInt(".1"));                // => NaN: integers can't start
with "."
console.log(parseFloat("$72.47"));          // => NaN: numbers can't start
with $
console.log(parseInt("11", 2));             // => 3: (1*2 + 1)
console.log(parseInt("ff", 16));            // => 255: (15*16 + 15)
console.log(parseInt("zz", 36));            // => 1295: (35*36 + 35)
console.log(parseInt("077", 8));            // => 63: (7*8 + 7)
console.log(parseInt("077", 10));           // => 77: (7*10 + 7)
```

- Beberapa trik konversi menggunakan operator (tentang operator akan dibahas di minggu ke-5);

  conversion-trick.js

```
let a = 5;
console.log(a + "");    // => String(5)

let b = "32";
console.log(+b);        // => Number("32");
console.log(b - 0);     // => Number("32");

console.log(!!a);       // => Boolean(5);
console.log(!!b);       // => Boolean(5);
```

Variable declaration and assignment

- Suatu *identifier* yang dapat berubah-ubah nilainya disebut *variable*. Jika kita tidak dapat mengubahnya maka dinamakan *constant*.
- Umumnya di dalam JavaScript pendeklarasian variable dan konstanta menggunaka `let` dan `const`. Untuk membedakan variable identifier dan constant identifier, umumnya untuk constant identfier menggunakan aturan penamaan huruf besar, dan variable identifier menggunakan huruf kecil.
- Setelah mendeklarasikan kita boleh (tidak harus) memberikan nilai pada variable

  declaration-example.js

```
let i;          // deklarasi tanpa inisialisasi nilai
let j, sum;     // deklarasi dalam satu baris

let message = "hello";    // deklarasi dengan insialisasi nilai
let k = 0, l = 0, m = 1;  // dekarasi dan inisialisasi dalam satu baris
let x = 2, y = x*x;       // Inisialisasi dapat mengunakan variable yang
sudah didekalarsikan

const H0 = 75;            // Hubble constant (km/s/Mpc)
const C = 299792.458;     // Speed of light in a vacuum (km/'s)
```

```
const AU = 1.496E8        // Astronomical Unit: distance from the earth to
the sun (km)
```

- Perbedaan deklarasi variable dengan `var` dan `let` adalah pada perilaku mereka ketika didekalariskan dalam suatu *scope*. Kita akan kembali lagi ke *function-scoped* dan *block-scoped* di pertemuan tentang fungsi. Untuk menghindari *bug*, umumnya orang menggunakan `let` karena dia memiliki sifat `block-scoped`.

- Terdapat suatu teknik untuk melakukan *unpacking* data kedalam variable yang dideklarasikan. Teknik ini dikenal sebagai *destructuring assignment*

  destructuring-assign.js

  ```
  let [x, y] = [1, 2];      // setara dengan  let x = 1, y = 2
  [x, y] = [x + 1, y + 1];  // setara dengan  x = x + 1, y = y + 1;
  [x, y] = [y, x];          // menukar nilai dua variable x dan y
  console.log([x, y]);      // => [3, 2]:
  ```

  Pada contoh di atas kita menggunakan *array literal* yang akan kita bahas dalam pertemuan tentang *Array*.

Tugas (Exercise - 02)

> 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. Jelaskan yang membuat salah atau benar dari dari program berikut?

   ```
   let a = 0.1
   let b = 0.2;
   let large_numbers = 9_007_199_254_740_992;
   console.log(a + b === 0.3);
   console.log(large_numbers + 1 === large_numbers);
   console.log(large_numbers + 2 === 9_007_199_254_740_992);
   ```

2. Jelaskan perbedaan hasil dua baris terakhir dalam program berikut:

```
let a1 = "banana";
let a2 = ("b" + "a" + + "a" + "a").toLowerCase()

console.log(`${a1}`);
console.log(`${a2}`);
```

Bagaimana hasil kedua baris tersebut? Mengapa hasilnya demikian? Jelaskan. Silahkan menggunakan chatGPT namun harus tetap dipahami prosesnya seperti apa.