Modul praktikum - Minggu 05 - *Expressions and Operators*

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

Tujuan:

- Mampu memahami ekspresi dalam program JavaScript.
- Mampu melakukan evaluasi (perhitungan nilai) dari suatu ekspresi.
- Mampu memahami *side effects* dari suatu ekspresi
- Mampu memahami operator dalam program JavaScript.
- Mampu menyusun gabungan ekspresi dan operator.
- Mampu memahami *short-circuiting behaviour* dari operator logika `&&` dan `||`

> 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.

Ekspresi adalah suatu frasa (kumpulan kata-kata) JavaScript yang dapat **dievaluasi** (program JavaScript dapat diterjemahkan ke dalam bahasa mesin) untuk menghasilkan **nilai dan/atau *side effects***. Kata-kata dalam hal ini adalah sintaks (struktur leksikal) yang telah kita pelajari di pertemuan sebelumnya. Untuk *side effect* akan kita jelaskan pada bagian terakhir.

Operator merupakan karakter atau kata kunci (*keyword*) yang digunakan untuk melakukan operasi antara satu, dua atau tiga *operand*. *Operand* bisa berupa *variables*, *values*, ataupun ekspresi

Operator tidak bisa berdiri sendiri tanpa *operands* (ekspresi yang ingin kita operasikan), namun ekspresi dapat berdiri sendiri tanpa operator.

Contoh ekspresi:

- Nilai-nilai konstan. Contoh: `2`, `3.4`, `"hello"`, `true`
- Nama dari suatu variabel
- Akses array dengan indeks integers (kita akan pelajari di pertemuan berikutnya). Contoh: `a[0]`
- Function invocation (pemanggilan fungsi; kita akan pelajari di pertemuan berikutnya). Contoh: `console.log("hello")`
- Gabungan ekspresi dan operator menjadi ekspresi yang lebih kompleks

Berikut adalah topik yang akan kita pelajari:

- Ekspresi utama (*primary expressions*)
- Operator dalam JavaScript
- Ekspresi aritmatik
- Ekspresi relasional
- Ekspresi logika

- Ekspresi pemberian nilai (*assignment expressions*)
- Operator-operator yang lain

Ekspresi utama

Merupakan ekspresi yang paling sederhana dari semua ekspresi dan hanya memuat dirinya sendiri tanpa pendefinisian variable atau struktur lain.

primary-express-literals.js

```
// Tulis perintah berikut di console browser, atau node.js interactive
1.23        // Sebuah literal bilangan
"hello"     // Sebuah literal string
/pattern/   // Sebuah literal regular expression
```

Ekspresi utama untuk *regular expression* akan kita pelajari di pertemuan tentang JavaScript Standard Library.

primary-express-reserved-words.js

```
// Tulis perintah berikut di console browser, atau node.js interactive
true;       // Akan terevaluasi sebagai boolean bernilai true
false;      // Akan terevaluasi sebagai boolean bernilai false
null;       // Akan terevaluasi sebagai null
this;       // Akan terevaluasi sebagai object yang didefinisikan saat itu
```

reserved word `this` merupakan kata kunci khusus yang mengacu pada object dimana `this` itu dipanggil. Kita akan kembali mempelajari tentang `this` di pertemuan tentang `type`: `object`. Lebih dalam lagi akan dibahas di kuliah Pemrograman Berbasis Objek (SI-201-410).

primary-express-ref-to-var

```
// Tulis perintah berikut di console browser, atau node.js interactive
let i = 0
let sum = 100;

i;
sum;
undefined;
```

Operator dalam JavaScript

Di dalam JavaScript, operator umumnya diwakili oleh karakter tanda baca seperti `+`, `-`, `*`, `/`. Beberapa juga diwakili oleh kata kunci seperti `delete` dan `typeof`. Pada pertemuan sebelumnya kita sudah seringkali menggunakan `typeof` yang dipanggil sebagai fungsi `typeof()`. Sebagai operator, `typeof` juga bisa dipanggil dengan cara: `typeof arg`. `arg` dapat diganti dengan primitive values atau object.

Ekspresi aritmatik

Ekspresi aritmatika yang umum seperti penjumlahan, penguarang, perkalian, dan pembagian diwakili oleh `+`, `-`, `*`, dan `/`, berturut-turut. Dua tambahan lagi adalah pemangkatan `**` dan perhitungan nilai sisa (modulo) `%`

- Operator dapat dikategorikan dari jumlah *operands*: *unary*, *binary*, dan *ternary operator*
- Khusus operator unary operator increment (++) dan decrement (--) memiliki side effect terhadap evaluasi kode di baris berikutnya. Side effect merupakan akibat yang terjadi diluar hasil yang diberikan oleh program.
- Setiap operator memiliki level *precedence*. Level ini menunjukkan operator manakah yang akan dievaluasi terlebih dahulu. Seperti perkalian dievaluasi terlebih dari daripada penjumlahan.
- Operator juga memiliki sifat asosiatif. Artinya apabila dua operator yang sama digunakan berturut-turut maka sifat asosiatif menentukan bagian manakah yang dievaluasi terlebih dahulu;

  associative.js

  ```
  let x = 1, y = 2, z = 3;
  let w1 = x - y - z;
  let w2 = ((x - y) - z);
  console.log(w1, w2)
  ```

  arithmetic-js

  ```
  // Perintah berikut dijalankan di browser console atau node.js mode
  interactive
  1 + 2;
  "hello" + " " + "there"
  "1" + "2";
  2**3;
  5 % 2
  -5 % 2
  6.5 % 2.1
  ```

- *Unary arithmetic operators* Merupakan operator aritmatik yang memiliki satu operand (*unary*). **unary-arithmetic.js**

  ```
  // Jalankan program berikut di browser console atau node.js mode interactive
  +2;
  -2;

  let i = 1, j = ++i;
  console.log(i, j);

  let n = 1, m = n++;
  console.log(n, m);
  ```

- *Bitwise operators* Merupakan operator yang melakukan manipulasi bits dari representasi bilangan di komputer. Bilangan yang dikenai operator ini akan berubah menjadi bilangan baru, yang hasilnya bilangan baru ini dapat kita telusuri setelah kita memecah proses *bitwise operator*.

  bitwise-op.js

  ```
  // Jalankan program berikut di browser console atau node.js mode interaktif
  0x1234 & 0x00FF;    // => 0x0034  (Bitwise AND)
  0x1234 | 0x00FF;    // => 0x12FF  (Bitwise OR)
  0xFF00 ^ 0xF0F0;    // => 0x0FF0  (Bitwise XOR)
  ~0x0F;              // => 0xFFFFFFF0   (Bitwise NOT)
  3 << 1;             // => 6: 011 << 1 === 110 = 1*4 + 1*2 + 1*1 = 6
  7 >> 1;             // => 3: 111 >> 1 === 011 = 0*4 + 1*2 + 1*1 = 3
  ```

  Untuk memahami proses Bitwise NOT perlu pembahasan lebih lanjut terkait Two's complement yang merupakan topik di luar jangkauan mata kuliah ini.

Ekspresi relasional

Merupakan ekspresi yang menguji relasi antara dua nilai dan menghasilkan nilai Boolean `true` atau `false`

- *equality* dan *inequality operators* Ada dua operator kesamaan di dalam JavaScript: `==` and `===`. Yang pertama `==` (dua tanda sama dengan) memperbolehkan adanya type conversion. Yang kedua `===` (tiga tanda sama dengan) tidak memperbolehkan adanya type conversion. Demikian pula untuk operator ketidaksamaan: `!=` dan `!==`.

  equality-and-strict-equal.js

  ```
  // Jalankan program berikut di browser console atau node.js mode interaktif
  "1" == true;
  "1" === true;
  ```

- *comparsion operator* Merupakan operator yang yang membandingkan dua buah nilai. Ada 4 operator komparasi: *less than* (`<`), *greater than* (`>`), *less than or equal* (`<=`), dan *greater than or equal* (`>=`)

  comparison-op.js

  ```
  // Jalankan program berikut di browser console atau node.js mode interaktif
  11 < 3;         // => false; perbandingan numerik
  "11" < "3";     // => true; perbandingan string
  "11" < 3;       // => false; perbandingan numerik; "11" -> 11
  "one" < 3;      // => false; perbandingan numerik; "one" -> NaN
  ```

- operator `in` dan `instanceof`

Operator `in` digunakan untuk menguji apakah suatu nilai terdapat pada objek lain.

in-operator.js

```
let point = {x: 1, y: 1};   // Mendefinisikan objek
console.log("x" in point);
console.log("z" in point)

let data = [7, 8, 9];       // Mendefinisikan array
console.log("0" in data);
console.log(1 in data);
console.log(3 in data);
```

Operator `instanceof` merupakan operator untuk menguji apakah suat variable yang memiliki nilai merupakan instance class dari suatu class prototype. Bahasan ini diluar jangkauan mata kuliah ini dan akan dipelajari dalam mata kuliah Pemrograman Berbasis Objek.

Ekspresi logika

Merupakan ekspresi untuk mewakili aljabar Boolean (ingat kembali pembahasan aljabar Boolean di kelas Matematika Diskrit)

- Logika AND (&&)
  Operator logika yang memberikan nilai `true` jika dan hanya jika *operand* pertama **dan** kedua bernilai `true`, selain itu memberikan nilai `false`
- Logika OR (||)
  Operator logika yang memberikan nilai `true` jika dan hanya jika *operand* pertama **atau** kedua bernilai `true` (dan kedua-duanya `true`). Selain itu memberikan nilai `false`.`
- Logika NOT (!)
  Operator logika yang memberikan nilai sebaliknya dari nilai yang dimilikinya. `true` menjadi `false`, `false` menjadi `true`.

Ekspresi pemberian nilai (*assignment expressions*)

Di dalam JavaScript, pemberian nilai ke suatu variable atau *property* (property akan dibahas lebih lanjut di pertemuan tentangn Objek) menggunakan tanda sama dengan (=). Tanda sama dengan ini berbeda makna secara matematis bahwa sama dengan mewakili kesamaan nilai ruas kanan dan ruas kiri. Di dalam *pseudocode* atau bahasa pemrograman lainnya (F#, OCaml, R, S) biasanya diwakili dengan tanda panah ke kiri (`<--`)

assignment-op.js

```
let i = 0;          // Memberi nilai variable i dengan bilangan bulat 0
let j = k = l = 1;  // Memberi nilai ke tiga variables j, k, l dengan
                    // bilangan bulat 1
console.log(i, j, k, l);
```

Kita juga dapat melakukan pengabungan *assignment operator* dengan *strict equality operator*

compound-assign-op.js

```
let a = 1;
let b = 2;
console.log((a = b) === 2);   // (a = b) akan memberikan keluaran akhir 2
                              // sehingga yang tercetak di console adalah `true`
```

Operator pemberian nilai dapat dikombinasikan dengan beberapa operator untuk menghemat penulisan

- `a += b` setara dengan `a = a + b`;
- `a -= b` setara dengan `a = a - b`;
- `a *= b` setara dengan `a = a * b`;
- `a /= b` setara dengan `a = a / b`;
- `a **= b` setara dengan `a = a ** b`;
- `a %=` setara dengan `a = a % b`;

Operator-operator yang lain

Ada beberapa operator-operator yang lain yang perlu kita ketahui.

- *Conditional operator* (`?:`)
  Operator ini termasuk ke dalam kategori `ternary operatory` karena memerlukan tiga *operands*.
  Operator ini adalah operator lain yang mewakili struktur kendali (*control flow*) untuk `if-else`.

  conditional-op.js

  ```
  let username = "Anastashia";    // change to empty string to se the effect

  let greeting1 = "hello " + (username ? username : "there");

  // Coding di atas setara dengan
  let greeting2 = "hello ";
  if (username) {
    greeting2 += username;
  } else {
    greeting2 += "there";
  }
  ```

- *First-Defined operator* (`??`)
  Merupakan `binary operator` yang akan mengevaluasi *operands* manakah yang terdefinisi (memiliki nilai boolean truthy).

  first-defined-op.js

```
let a = 1, b = 2, c = 3, d = 4;
let min = a ?? b ?? c ?? d;
console.log(min);     // => 1

a = undefined;
min = a ?? b ?? c ?? d;
console.log(min);     // => 2; undefined
```

Contoh penggunaan untuk pengaturan ukuran

```
// Jika maxWidth telah didefinisikan, gunakan pengaturan itu
// Jika tidak gunakan pengaturan di objek preferences
// Jika keduanya tidak terpenuhi gunakan pengaturan default
let max = maxWidth ?? preferences.maxWidth ?? 500;
```

- `typeof` dan `delete` operator Operator `typeof` merupakan *unary operatory* dan diletakkan sebelum *operand* dan akan menghasilkan tipe dari *operand*

  Sedangkan operator `delete` digunakan untuk menghapus suatu *property* di objek atau menghapus elemen di suatu array

  typeof-and-delete.js

  ```
  let o = {x: 1, y: 2};
  delete o.x;
  console.log("x" in o);

  let a = [1, 2, 3];
  delete a[2];
  console.log(2 in a);
  console.log(a.length);   // Penghapusan elemen tidak serta merta mengurangi
                           // Ukuran arrat
  ```

  Kita akan mempelajari lebih lanjut terkait objek dan array di pertemuan kemudian.

- operator koma (`,`) Operator koma adalah suatu *binary operator* yang mengevaluasi nilai operand sebelah kiri operator, kemudian mengevaluasi nilai operand sebelah kanan operator, kemudian memberikan hasil dari operand sebelah kanan

  comma-op.js

  ```
  let i, j, k;
  let final = (i = 0, j = 1, k = 2);
  console.log(final);
  ```

Operator koma ini tanpa sadar kita gunakan dalam sintaks perulangan `for-loop`

```
for (let i = 0, j = 0; i < j; i++, j--) {
  console.log(i + j);
}
```

Tugas

Carilah topik *final project* yang akan kalian kerjakan. Paling lambat sebelum UTS sudah mendapat ide untuk mengerjakan apa. Topik atau proyek yang akan kalian kerjakan usahakan sesuai kemampuan kalian. Jika memungkinkan waktu, tenaga, dan *human resources*, maka silahkan untuk membuat proyek yang lebih sulit. Pastikan proyek yang akan kalian kerjakan dapat diselesaikan sebelum minggu ke-16. Ide proyek boleh lebih dari satu. Tautan Google Sheet akan dibagikan untuk diisi. Boleh didiskusikan dengan assiten atau dosen pengampu mata kuliah.