**Loop**

Ketika menulis program komputer, akan ada situasi di mana kita perlu melakukan hal yang sama berkali-kali. Misalnya kita ingin menampilkan semua nama pengguna yang terdaftar di aplikasi atau sesederhana menampilkan angka 1 sampai 10. Tentunya tidak praktis jika kita menulis kode seperti berikut:

1. console.log(1);
2. console.log(2);
3. console.log(3);
4. console.log(4);
5. console.log(5);
6. console.log(6);
7. console.log(7);
8. console.log(8);f
9. console.log(9);
10. console.log(10);

Bagaimana jika kita perlu menampilkan angka 1 sampai 100? Sudah terbayang repotnya, bukan? Maka dari itu kita perlu mempelajari teknik yang dapat mengatasi permasalahan tersebut, teknik ini disebut dengan *looping*. JavaScript memiliki banyak opsi untuk melakukan *looping* atau perulangan kode, antara lain:

**For loop**

Dari beberapa cara melakukan proses loop pada JavaScript, “for” merupakan salah satu cara yang banyak digunakan. Struktur dasar dari for tampak seperti berikut:

1. for(inisialisasi variabel; test kondisi; perubahan nilai variabel) {
2. // do something
3. }

Berikut ini contoh penerapannya secara nyata:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13887#run1-editor1)



1

for(let i = 1; i <= 5; i=i+2) {

2

console.log(i);

3

}

4

​

5

/\* output

6

0

7

1

8

2

9

3

10

4

11

\*/

 INPUT  RESET

 JALANKAN

1 3 5

Lebih ringkas, bukan? Mungkin kode tersebut sulit dipahami, jadi mari kita bahas sedikit demi sedikit. Terdapat tiga bagian utama dalam sintaks for di atas:

* Pertama, variabel i sebagai *index* iterasi. Pada variabel ini kita menginisialisasi nilai awal dari perulangan.
* Kedua, operasi perbandingan. Pada bagian ini, JavaScript akan melakukan pengecekan kondisi apakah perulangan masih perlu dilakukan. Jika bernilai true, maka kode di dalam blok for akan dijalankan.
* Ketiga, increment/decrement. Di sini kita melakukan penambahan atau pengurangan variabel iterasi. Jadi, pada contoh di atas variabel i akan ditambah dengan 1 di setiap akhir perulangan. Perubahan nilai ini penting karena jika kita mengubah nilainya, proses perulangan dapat berjalan selamanya karena kondisi akan terus terpenuhi.

JIka diartikan, maka kode di atas bisa dimaknai dengan “Jika i kurang dari 5, maka tampilkan nilai i.”

**For of loop**

Cara lain dalam melakukan looping adalah menggunakan for..of. For of mulai hadir pada ECMAScript 2015 (ES6). Cara ini jauh lebih sederhana dan modern dibanding for loop biasa. Sintaks dasar dari for of loop adalah seperti ini:

1. for(arrayItem of myArray) {
2. // do something
3. }

Yup, for of tidak membutuhkan banyak *statement* untuk melakukan *looping* pada *array*. Penjelasan tentang array akan lebih detail dibahas pada modul berikutnya. Sebagai permulaan, kita bisa menganggap array sebagai kumpulan nilai yang disimpan dalam satu variabel.

Dengan for..of nilai tiap array akan diinisialisasi pada variabel baru yang kita tentukan pada tiap proses *looping*-nya. Jumlah proses *looping*-nya pun akan menyesuaikan dengan ukuran dari array. Sederhananya seperti kita melakukan perintah “Hei JavaScript! Lakukan perulangan pada myArray, akses tiap nilainya, dan simpan pada variabel arrayItem”. Pada proses looping kita gunakan variabel arrayItem untuk mengakses tiap nilai dari item myArray.

Agak sulit memang menjelaskan dengan kata-kata, mari kita terjemahkan dalam kode secara langsung.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13887#run2-editor1)



1

let myArray = ["Luke", "Han", "Chewbacca", "Leia"];

2

​

3

for(const arrayItem of myArray) {

4

console.log(arrayItem)

5

}

6

​

7

/\* output

8

Luke

9

Han

10

Chewbacca

11

Leia

12

\*/

 INPUT  RESET

 JALANKAN

Luke Han Chewbacca Leia

Lebih mudah dan simpel bukan?

**While and do-while**

Metode lain untuk melakukan looping adalah dengan statement *while*. Sama seperti for, instruksi while mengevaluasi ekspresi boolean dan menjalankan kode di dalam blok while ketika bernilai true.

Untuk menampilkan angka 1 sampai 100 dengan while kita bisa menulis kode seperti berikut:

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13887#run3-editor1)



1

let i = 1;

2

​

3

while (i <= 100) {

4

console.log(i);

5

i++;

6

}

 INPUT  RESET

 JALANKAN

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Bisa dilihat pada kode di atas bahwa looping dengan while tidak memiliki ketergantungan dengan variabel iterasi seperti pada for loop. Karena itu, meskipun while dapat melakukan perulangan yang sama dengan for, while lebih cocok digunakan pada kasus di mana kita tidak tahu pasti berapa banyak perulangan yang diperlukan.

Bentuk lain dari while adalah perulangan do-while.

* [**main.js**](https://www.dicoding.com/academies/256/tutorials/13887#run4-editor1)



1

let i = 1;

2

​

3

do {

4

console.log(i);

5

i++;

6

} while (i <= 2);

 INPUT  RESET

 JALANKAN

1 2

Kondisi pada while akan dievaluasi sebelum blok kode di dalamnya dijalankan, sedangkan do-while akan mengevaluasi boolean expression setelah blok kodenya berjalan. Ini artinya kode di dalam do-while akan dijalankan setidaknya satu kali.

**Infinite loops**

Ketika menerapkan perulangan pada program, ada satu kondisi yang perlu kita hindari yaitu *infinite loop*. Infinite loop atau *endless loop* adalah kondisi di mana program kita *stucked* di dalam perulangan. Ia akan berjalan terus hingga menyebabkan *crash* pada aplikasi dan komputer kecuali ada intervensi secara eksternal, seperti mematikan aplikasi.

Kode berikut ini adalah contoh di mana kondisi infinite loop dapat terjadi:

* [**While**](https://www.dicoding.com/academies/256/tutorials/13887#tab1-code1)
* [For](https://www.dicoding.com/academies/256/tutorials/13887#tab1-code2)

1. let i = 1;
3. while (i <= 5) {
4. console.log(i);
5. }

Dapatkah Anda menemukan apa yang salah dari kode di atas sehingga terjadi infinite loop?

Jawabannya adalah karena variabel i selalu bernilai 1. Alhasil, kondisi i <= 5 akan selalu bernilai true yang mengakibatkan aplikasi akan terus mencetak 1 ke konsol sehingga mengalami *crash*.