



# Basic Javascript Algorithm

---



## Gambaran Umum

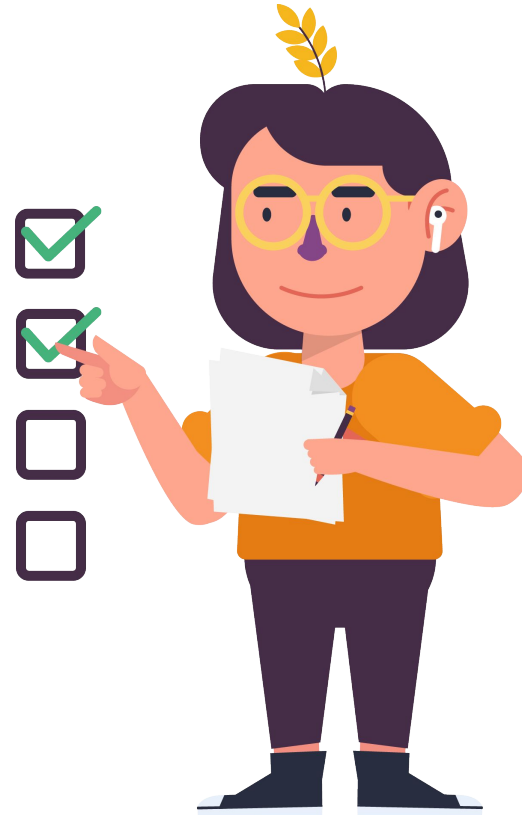
Materi tersebut menjelaskan tentang apa itu **algoritma**, bagaimana **representasi algoritma** dengan **flowchart** dan **pseudocode**, dan juga seperti apa **alur pengambilan keputusan** dan **perulangan**.





## Tujuan Pembelajaran

- a. Siswa dapat menjelaskan **algoritma**.
- b. Siswa dapat menggambarkan **representasi algoritma** yang tepat dan sesuai fungsinya.
- c. Siswa dapat membuat sebuah dokumen alur suatu algoritma dengan **flowchart**.
- d. Siswa dapat membuat representasi algoritma suatu algoritma dengan **pseudocode**.
- e. Siswa dapat menerapkan **alur pengambilan keputusan**.
- f. Siswa dapat menerapkan **alur perulangan**.



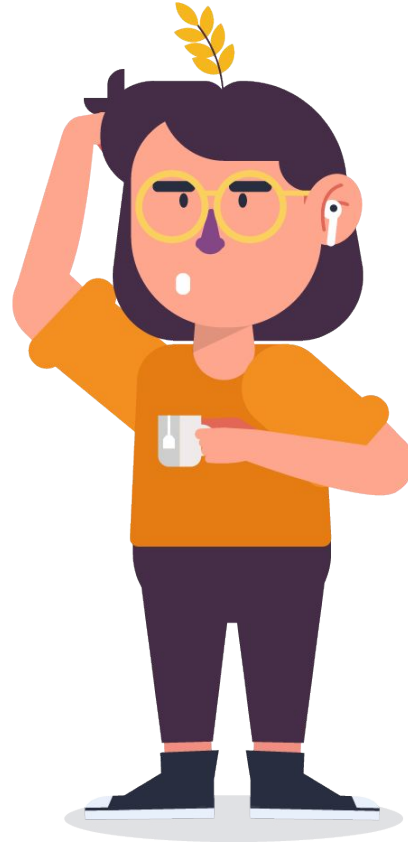
**ALGORITMA**



Algoritma adalah suatu metode *step by step* untuk menyelesaikan suatu masalah



PENGEN TOPUP  
PULSA EUY..!



Setiap hari selalu saja ada masalah ya..!?

Ketika kita **berpikir** untuk menyelesaikan masalah selangkah demi selangkah, maka sebenarnya kita sedang menerapkan **algoritma**.

Contohnya, proses **top-up pulsa** pakai **voucher gosok**.

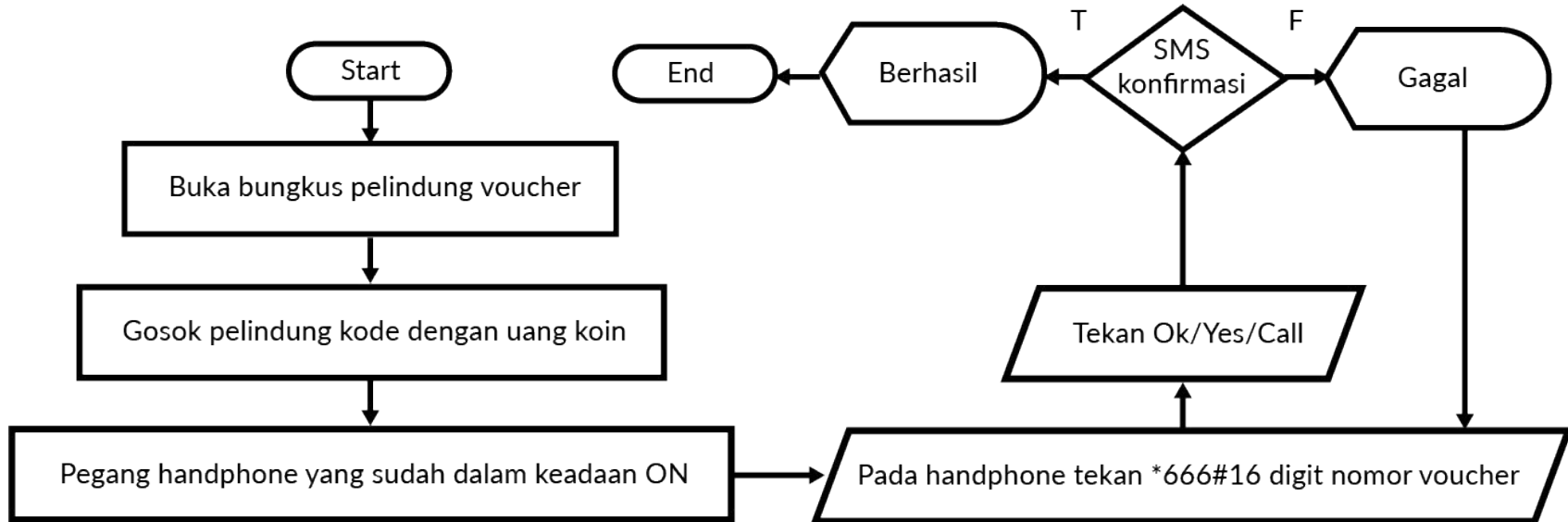
Setelah beli **voucher**, kita harus menggosok bagian kartu agar **kode voucher** dapat digunakan.

Jika *input* kodenya **benar**, proses *top-up* akan berhasil. Tapi, pulsa tidak akan terisi jika kita **salah** input kode.



## Top-up pulsa dengan voucher tadi,

algoritmanya bisa direpresentasikan seperti gambar berikut ini







**Tapi,**  
**menggambarkan**  
**algoritma** itu caranya  
seperti apa ya?

# **REPRESENTASI ALGORITMA**

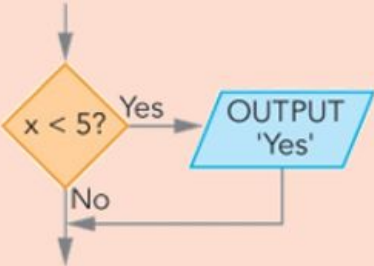


**Sebelum menulis kode**, sebaiknya kita perlu menentukan bagaimana membuat **alur algoritmanya** terlebih dahulu.

## Ada 2 opsi dalam menyusun alur algoritma

Opsi pertama

Opsi kedua

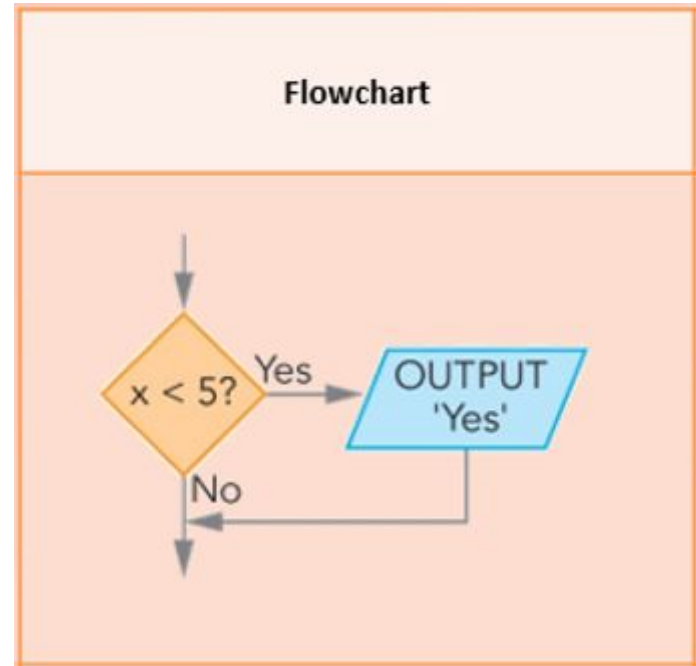
Flowchart	Pseudo-code
	<pre>IF x &lt; 5 THEN     OUTPUT 'Yes' ENDIF</pre>

Opsi Pertama

Bagaimana  
menggambarkan  
*flowchart*?



**Opsi pertama**  
menunjukkan suatu  
***flowchart*** yang  
menggunakan  
**berbagai simbol**  
untuk  
**memvisualisasikan**  
**algoritma.**



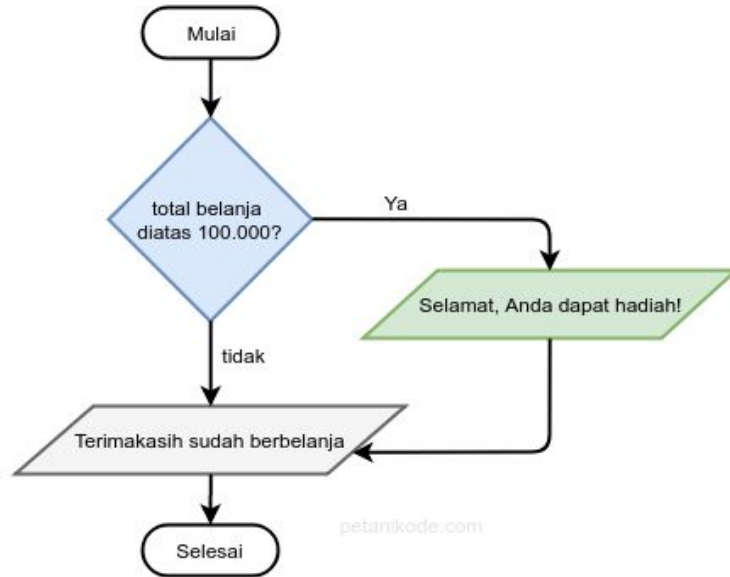


### Jadi, *flowchart* itu adalah ...

**Representasi visual** dari *control flow* suatu algoritma. Representasi ini menggambarkan beberapa hal, yaitu:

- ***statement*** yang perlu **dieksekusi**,
- **keputusan** yang perlu **dibuat**,
- ***flow logika*** (untuk iterasi dan tujuan lain), serta
- **terminal** yang menunjukkan **titik awal dan akhir**.

## Contoh *flowchart* sederhana



*Flowchart* sederhana ini memvisualisasikan sebuah alur untuk mengetahui suatu hal; apakah setelah berbelanja kita berhak mendapatkan hadiah atau tidak. Dalam kasus ini, kita akan mendapatkan hadiah jika total belanja lebih dari 100 ribu rupiah.



## Yuk, kita bahas simbol *flowchart*!

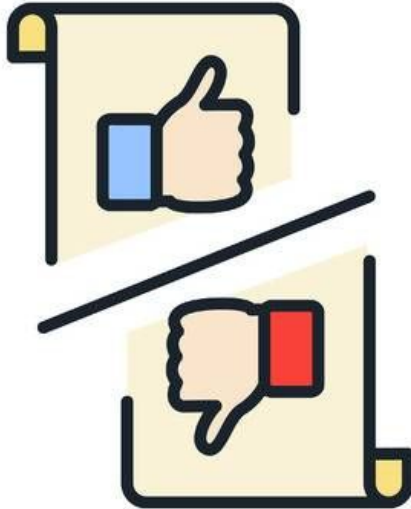
Pada *flowchart* sebelumnya, kamu mungkin sadar kalau ada berbagai simbol yang digunakan. Nah, Itulah keunikannya *flowchart*!

Sebelum membuat *flowchart*, kita perlu memahami berbagai simbol dan fungsinya.

Perhatikan dan pahamiilah berbagai jenis simbol serta fungsi dari masing-masing simbol *flowchart* pada tabel.

	<b>Flow Direction symbol</b> Yaitu simbol yang digunakan untuk menghubungkan antara simbol yang satu dengan simbol yang lain. Simbol ini disebut juga connecting line.		<b>Simbol Manual Input</b> Simbol untuk pemasukan data secara manual on-line keyboard
	<b>Terminator Symbol</b> Yaitu simbol untuk permulaan (start) atau akhir (stop) dari suatu kegiatan		<b>Simbol Preparation</b> Simbol untuk mempersiapkan penyimpanan yang akan digunakan sebagai tempat pengolahan di dalam storage.
	<b>Connector Symbol</b> Yaitu simbol untuk keluar - masuk atau penyambungan proses dalam lembar / halaman yang sama.		<b>Simbol Predefine Proses</b> Simbol untuk pelaksanaan suatu bagian (sub-program)/prosedure
	<b>Connector Symbol</b> Yaitu simbol untuk keluar - masuk atau penyambungan proses pada lembar / halaman yang berbeda.		<b>Simbol Display</b> Simbol yang menyatakan peralatan output yang digunakan yaitu layar, plotter, printer dan sebagainya.
	<b>Processing Symbol</b> Simbol yang menunjukkan pengolahan yang dilakukan oleh komputer		<b>Simbol disk and On-line Storage</b> Simbol yang menyatakan input yang berasal dari disk atau disimpan ke disk.
	<b>Simbol Manual Operation</b> Simbol yang menunjukkan pengolahan yang tidak dilakukan oleh komputer		<b>Simbol magnetik tape Unit</b> Simbol yang menyatakan input berasal dari pita magnetik atau output disimpan ke pita magnetik.
	<b>Simbol Decision</b> Simbol pemilihan proses berdasarkan kondisi yang ada.		<b>Simbol Punch Card</b> Simbol yang menyatakan bahwa input berasal dari kartu atau output ditulis ke kartu
	<b>Simbol Input-Output</b> Simbol yang menyatakan proses input dan output tanpa tergantung dengan jenis peralatannya		<b>Simbol Dokumen</b> Simbol yang menyatakan input berasal dari dokumen dalam bentuk kertas atau output dicetak ke kertas.

## Kelebihan dan kekurangan *flowchart*



### Kelebihan

1. *Flowchart* mampu menyajikan *control flow* algoritma secara visual. Jadi, kita lebih mudah untuk mengerti alur penyelesaian suatu masalah atau kondisi.
2. Lebih mudah mendeteksi kesalahan atau ketidakakuratan suatu *flowchart* yang kompleks dan detail.

### Kekurangan

1. Pengerjaannya memakan waktu. Kita harus memposisikan, memberi label, dan menghubungkan simbol dalam *flowchart*.
2. Jika menggunakan *tools* khusus untuk membuat *flowchart*, itu akan menghambat pemahaman kita tentang algoritma.

Opsi Kedua

Bagaimana  
membuat  
***pseudocode?***





**Opsi kedua**, kita bisa merepresentasikan algoritma dengan ***pseudocode***.

Di sini, kita dapat menggunakan **teks** atau **kata-kata** untuk menunjukkan suatu alur.

## Opsi kedua

Pseudo-code

```
IF x < 5 THEN  
    OUTPUT 'Yes'  
ENDIF
```



## Contoh *pseudocode* sederhana



BACA dan SIMPAN "angka pertama" di variabel (A)  
BACA dan SIMPAN "angka kedua" di variabel (B)  
HITUNG "angka pertama" ditambah "angka kedua"  
SIMPAN hasil perhitungan di variabel (SUM)  
TAMPILKAN hasil perhitungan (SUM)



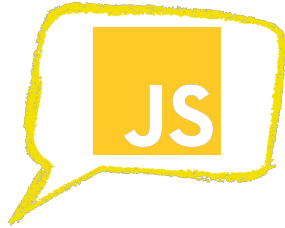
READ "the first number" and SAVE in the variable (A)  
READ "the second number" and SAVE in the variable (B)  
SUM the both numbers  
SAVE the result in the variable (SUM)  
PRINT the result (SUM)

### Catatan:

Karena sintaksnya fleksibel,  
maka tidak ada aturan yang  
tegas untuk menuliskan  
*pseudocode*.



Setelah membuat **pseudocode** seperti contoh yang tadi,  
Bisa kita **konversikan** ke dalam **bahasa pemrograman**.



```
let no1, no2, no3;  
no1 = prompt("Nomor pertama?");  
no2 = prompt("Nomor kedua?");  
hasil = Number(no1) + Number(no2);  
console.log(hasil);  
alert("Hasil = " + hasil);
```



```
no1 = input("Nomor pertama? ")  
no2 = input("Nomor kedua? ")  
hasil = int(no1) + int(no2)  
  
print("Hasil", hasil)
```

## Catatan:

Jika masalah yang ingin diselesaikan kompleks, dengan membuat *pseudocode* terlebih dahulu akan memudahkan kita *ngoding* nantinya.



Setelah membuat ***pseudocode***,  
gimana ya kalau ingin membuat  
**alur pengambilan keputusan?**

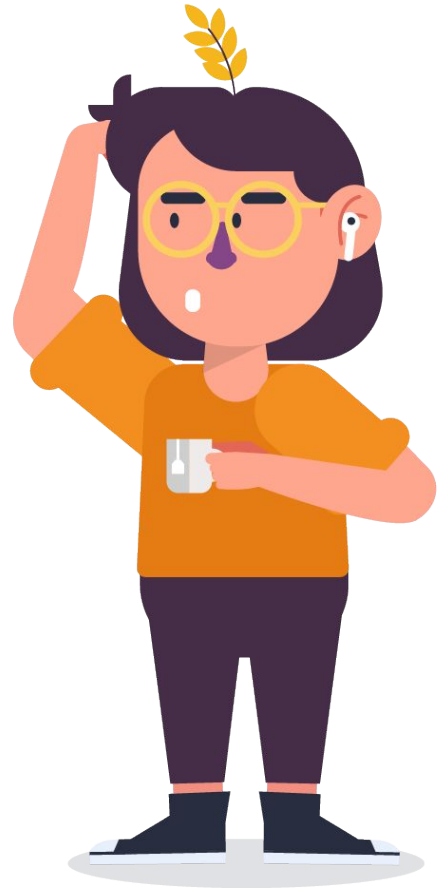
# **ALUR PENGAMBILAN KEPUTUSAN**





Dalam kegiatan sehari-hari, kita perlu melakukan **tindakan yang berbeda** tergantung **kondisinya**.

Misal, kita ingin pergi keluar rumah dengan berkendara. Kita akan memperhitungkan ramalan cuaca untuk menentukan kendaraan apa yang paling nyaman digunakan berdasarkan kondisi cuaca di luar sana.





Karena itulah kita membutuhkan **alur pengambilan keputusan**.

Dengan skema tersebut, kita dapat mengambil langkah yang tepat sesuai dengan kondisi yang ada.

Alur pengambilan keputusan ini dapat menggunakan beberapa cara:

1. **Kondisional *if-else***
2. **Kondisional operator ('?')**
3. ***Switch case***



# Kondisional *if-else*



## 1. Kondisional *if-else*

```
let tahun = prompt('Tahun berapa  
Indonesia merdeka?');  
  
if (tahun == 1945) alert( 'Kamu  
benar!' );
```

```
let tahun = prompt('Tahun berapa  
Indonesia merdeka?');  
  
if (tahun == 1945) {  
    alert( "Mantul!" );  
    alert( "Pinter!" );  
}
```

Pernyataan *if(...)* akan melakukan **mengevaluasian** kondisi di dalam **tanda kurung ()**. Jika hasilnya **sesuai (true)**, maka kode yang terdapat di dalam blok akan **dieksekusi**.

Jika kita ingin menjalankan **lebih dari satu pernyataan**, maka kita harus **bungkus/blok kode** dengan **tanda kurung kurawal {}**.



Nah, sebenarnya **statement if** akan mengevaluasi ekspresi yang berada di dalam **tanda kurung (...)**, lalu akan mengubah hasilnya menjadi tipe data **boolean**

## Catatan:

- Angka 0, string kosong (" "), tidak terdefinisi (*undefined*), dan NaN akan menghasilkan nilai **false**, sehingga disebut nilai "**falsy**".
- Nilai yang lainnya akan menjadi **true**, sehingga disebut nilai "**truthy**".

## Contoh

Kode dalam kondisi ini tidak akan dijalankan:

```
if (0) { // 0 adalah falsy
  ...
}
```

Sebaliknya, kode dalam kondisi di bawah ini akan dijalankan:

```
if (1) { // 1 adalah truthy
  ...
}
```

Kode dengan nilai boolean yang telah dievaluasi ke if:

```
let cond = (tahun == 2015); // true
atau false
if (cond) {
  ...
}
```



## Contoh *statement else*

Pernyataan ***if(...)*** dapat berisi blok kode lain yaitu ***else***,  
Pernyataan ***else*** ini akan dijalankan hanya ketika **kondisi tidak sesuai atau salah**.

## Contoh *statement if-else if*

Jika kita mau menguji beberapa kondisi. kita dapat menggunakan ***statement if-else if***.

JavaScript akan memeriksa kondisi pertama. Jika kondisinya sesuai, maka akan dieksekusi. Jika tidak sesuai, kondisi kedua akan diperiksa. Jika kondisi kedua juga tidak sesuai, maka kondisi terakhir yang akan dijalankan.

```
let tahun = prompt('Tahun berapa Indonesia merdeka?');  
  
if (tahun == 1945) {  
  alert( 'Kamu benar!' );  
} else {  
  alert( 'Kok lupa sih?' );  
}
```

```
let tahun = prompt('Tahun berapa Indonesia merdeka?');  
  
if (tahun < 1945) {  
  alert( "Masih dijajah.." );  
} else if (tahun > 1945)  
  alert( "Sudah lewat.." );  
} else {  
  alert( "Tepat sekali!.." );  
}
```



## Kondisional *Operator* ('?')



## 2. Kondisional operator '?'

Terkadang kita perlu menetapkan variabel **tergantung pada suatu kondisi**.

Pada contoh ini, sejak awal kita sudah menyiapkan variabel terlebih dahulu dengan nama "akses". Variabel tersebut akan ditetapkan tergantung dari kondisinya.

Misal, jika umur kita lebih dari 18, maka kondisi pertama kita tetapkan agar terpenuhi sehingga kita mendapatkan akses masuk. Jika tidak, maka kita dilarang masuk.

```
let akses;  
let umur = prompt('Kamu umur  
berapa?');  
  
if (umur > 18) {  
    akses = true;  
} else {  
    akses = false;  
}  
  
alert(akses);
```



## 3. Switch case

Pada pembahasan sebelumnya, kita sudah belajar tentang penggunaan If-else. Namun, penggunaan If-else dapat diubah bentuknya menjadi lebih simpel apabila suatu kondisi memiliki banyak pilihan. Hal ini dapat kita lakukan dengan pernyataan **switch case**.

Misalnya seperti pada gambar ini. Dengan statement switch, kita dapat dengan mudah mengganti kostum suatu karakter permainan online dengan banyak opsi tergantung kondisi mood-nya.





## Switch case syntax

```
switch(x) {  
  case 'nilai1': // if (x === 'nilai1')  
    ...  
    [break]  
  
  case 'nilai2': // if (x === 'nilai2')  
    ...  
    [break]  
  
  default:  
    ...  
    [break]  
}
```

- Nilai-nilai diperiksa untuk dilakukan **perbandingan yang ketat**. Pada contoh, 'nilai1', 'nilai2', dan seterusnya akan dibandingkan.
- **Jika persamaan ditemukan**, *switch* akan menjalankan kode di tempat *case* yang sesuai.
- **Jika tidak ada case yang cocok**, maka kode *default* dieksekusi (jika ada).



## Contoh *switch case*

```
let a = 2 + 2;

switch (a) {
  case 3:
    alert( 'Terlalu kecil' );
    break;
  case 4:
    alert( 'Benar sekali' );
    break;
  case 5:
    alert( 'Terlalu besar' );
    break;
  default:
    alert( "Tidak tahu" );
}
```



# **ALUR PERULANGAN (LOOPING)**



Setiap pemrograman memungkinkan kita untuk **mengulang suatu tindakan**.

Misal, ketika belanja online, kita ingin **menambahkan 10 item** atau kita ingin **mengurangi 5 item** di keranjang belanja.

Nah, dengan menerapkan **alur perulangan (loops)**, kita jadi lebih mudah untuk menggunakan kode yang sama beberapa kali.

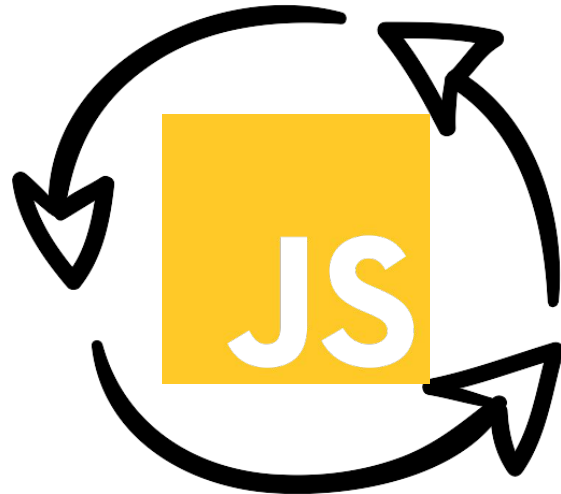




## Jenis alur perulangan

Terdapat **tiga jenis alur perulangan** yang perlu kita ketahui, yaitu:

1. ***“for” loop***
2. ***“while” loop***
3. ***“do...while” loop***





### Contoh sederhana alur perulangan dengan “for”

Alur perulangan *for* dapat digambarkan dalam kegiatan sehari-hari, misalnya ketika petugas kebersihan mengepel lantai di sebuah gedung.

Ia akan mencuci kain pel, memeras pel, lalu mengepel sampai ke lantai lima, kemudian beristirahat.



#### Catatan:

Pengulangan akan terus dilakukan sampai kondisi tertentu tercapai. **Saat kondisi tersebut tercapai, maka perulangan akan dihentikan.**



Nah, jika dituliskan alurnya seperti ini:

1. Ambil alat pel.
2. Cuci kain pel menggunakan air bersih.
3. Peras kain pel.
4. Mengepel lantai secara urut.
5. **Lakukan tahap 2 - 4, naik ke lantai berikutnya sampai ke lantai 5.**
6. Istirahat.



### "for" loop syntax

```
for (inisialisasi; kondisi; pengubah nilai) {  
  // ... pernyataan/perintah ...  
}
```

### Contoh penulisan "for" loop

(dari alur sederhana yang sebelumnya)

```
for (let i = 1; i <= 5; i++) {  
  //mengepel lantai dari lantai 1 sampai 5  
  alert(i);  
}
```

#### Catatan:

- Inisialisasi:  
Untuk menentukan variabel awal untuk pengulangannya.
- Kondisi:  
Untuk menghentikan fungsi pengulangan.
- Pengubah nilai:  
Untuk mengaktifkan pengulangannya, bisa bertambah atau berkurang.





### Contoh sederhana alur perulangan dengan “while”

Nah, sekarang kita coba ubah contoh kondisi alur perulangan “for” menjadi kondisi alur perulangan dengan “while”.

Petugas ingin memeriksa adanya noda pada lantai. Jadi, selama terdapat noda pada lantai, maka ia akan mengulangi tahap kedua sampai kelima secara terus-menerus.



Jika dituliskan alurnya akan seperti ini:

1. Ambil alat pel.
2. **Periksa adanya noda pada lantai.**
3. Cuci kain pel menggunakan air bersih.
4. Peras kain pel.
5. Pel lantai secara urut.
6. Ulangi tahap 2 - 5.

#### Catatan:

Pengulangan akan dilakukan bila suatu kondisi tercapai atau benar. **Selama kondisi itu benar, perulangan akan terus dilakukan.**



### “while” loop syntax

```
while (kondisi) {  
    // ... pernyataan/perintah ...  
}
```

### Contoh penulisan “while” loop

(dari alur sederhana yang sebelumnya)

```
let i = 1  
while (i<=5) {  
    //mengepel lantai dari lantai 1 sampai 5  
    alert(i);  
    i++;  
}
```

#### Catatan:

Pada contoh ini, jika `i++` dihilangkan, maka *loop* akan berulang secara terus menerus atau tidak terhingga. Pada kasus ini, *browser* menyediakan cara untuk menghentikan *loop*. Tapi, kalau JavaScript dari sisi *server* maka kita yang harus menghentikan prosesnya.



## Contoh *infinite loop*

Gimana ya kalau kita ingin menghentikan perulangan yang terus-menerus tanpa henti (*infinite loop*)?

Ibaratnya mau sendiri dulu gitu...  
Hm.. Break dulu aja kali ya..



```
const lantaiBersih = true;
let lantaiGedung = 5;

while (lantaiBersih) {
  alert(`Saya sudah mengepel lantai ${lantaiGedung}.`);
  lantaiGedung--;
}
```

## Solusi *infinite loop*



```
const lantaiBersih = true;
let lantaiGedung = 5;

while (lantaiBersih) {
  alert(`Saya sudah mengepel lantai ${lantaiGedung}.`);
  lantaiGedung--;
  if (lantaiGedung === 0) {
    alert("Semua lantai sudah bersih."); break;
  }
}
```



## Contoh sederhana alur perulangan dengan “do...while”

Sekarang mari kita coba ubah contoh yang kondisi alur perulangan “while” menjadi kondisi alur perulangan dengan “do...while”.

Petugas akan melakukan tahap pertama sampai keempat terlebih dahulu. Kemudian, ia memeriksa kondisi kain pel. Jika kain pel sudah kotor, maka ia akan mengulang langkah kedua hingga kelima.

### Catatan:

Alur perulangan do...while sama dengan while. Tapi, **do...while** akan langsung melakukan loop, lalu cek kondisi. Sedangkan **while loop** harus mengecek kondisi di awal.



Jika kita ubah alurnya akan seperti ini:

1. Ambil alat pel.
2. Cuci kain pel menggunakan air bersih.
3. Peras kain pel.
4. Mengepel lantai secara urut.
5. **Setelah dirasa kain pel sudah kotor.**
6. Lakukan tahap 2 - 5.  
(jika kain pel tidak kotor, maka tidak perlu diulang)

## “do...while” loop syntax

```
do {  
  // ... pernyataan/perintah ...  
} while (kondisi);
```

## Contoh penulisan “do...while” loop

(dari alur sederhana yang sebelumnya)

```
let i = 1  
do {  
  //mengepel lantai dari lantai 1  
  sampai 5  
  alert(i);  
  i++;  
} while (i<=5)
```

### Catatan:

**Loop akan langsung mengeksekusi pernyataan/perintah di awal,** kemudian akan melakukan pengecekan kondisi. Jadi, jika kondisinya masih *true* maka kode akan dijalankan lagi.

# Rangkuman

## Algoritma

Suatu metode *step by step* untuk menyelesaikan suatu masalah.

## Flowchart

Representasi visual dari *control flow* suatu algoritma.

## Pseudocode

Representasi tekstual dari suatu algoritma yang mendekati sebuah flowchart.

## Alur Pengambilan Keputusan

- Kondisional *if-else*
- Kondisional operator ('?')
- *Switch case*

## Alur Perulangan

- "for" loop
- "while" loop
- "do...while" loop

## Buatlah algoritma tentang cara *apply* sebagai *job seeker* di sebuah **situs pencarian kerja!**

- Algoritma tersebut memiliki representasi visual (**flowchart**).
- Algoritma tersebut memiliki representasi tekstual (**pseudocode**).
- Terapkan **alur kondisional** dari algoritma tersebut ke dalam bahasa pemrograman Javascript.
- Terapkan **alur perulangan** dari algoritma tersebut ke dalam bahasa pemrograman Javascript.
- Push jawaban tersebut ke repo group di **Gitlab: Week2**  
Deadline: Hari ini, pukul 23.59 WIB.  
(Kita bahas besok pada pertemuan berikutnya)



## Tugas

# Referensi

<https://www.w3schools.com/js/>

<https://seputarilmu.com/2019/02/flowchart-adalah.html>

<https://medium.com/dot-intern/jenis-flowchart-dan-simbol-simbolnya-ef6553c53d73>



**Terima Kasih**