




# Asynchronous Javascript



Merupakan sebuah istilah, atau teknik yang dapat memberikan kemampuan program kita untuk memulai sebuah task yang kemungkinan akan memerlukan waktu lebih lama, tetapi masih tetap dapat melakukan task yang lain. Contohnya adalah;

- Ketika melakukan fetching data(mengambil data dari sebuah network).  
Maksudnya adalah, sembari menunggu data di fetching, web tetap dapat menampilkan tampilan yang lain terlebih dahulu, tanpa harus menunggu data selesai di fetching terlebih dahulu.

\*jangan panik jika anda belum paham, kita akan pelajari satu persatu hingga ujung slide.



# Synchronous programming

Merupakan hal yang kita biasa lakukan, yaitu dengan menulis program yang akan dieksekusi baris per baris.

Coba run program disamping, perhatikan, bahwa code melakukan code itu baris per baris, ini dibuktikan dengan hasil yang menampilkan angka 1, 2, dan 3 secara berurutan.

```
const a = 1;  
const b = 2;  
const c = 3;  
  
console.log(a);  
console.log(b);  
console.log(c);
```

1  
2  
3



# Asynchronous Programming

Merupakan sebuah cara untuk mengeksekusi code tanpa harus menunggu code yang lain selesai dilakukan. Disini setTimeout membuat variable b di-print belakangan karena diberi delay selama 1 detik.

```
const a = 1;
const b = 2;
const c = 3;

console.log(a);
setTimeout(() => console.log(b), 1000);
console.log(c);
```

1  
3  
2




# Promise

Merupakan sebuah cara pada JavaScript, yang bekerja seperti sebuah janji.

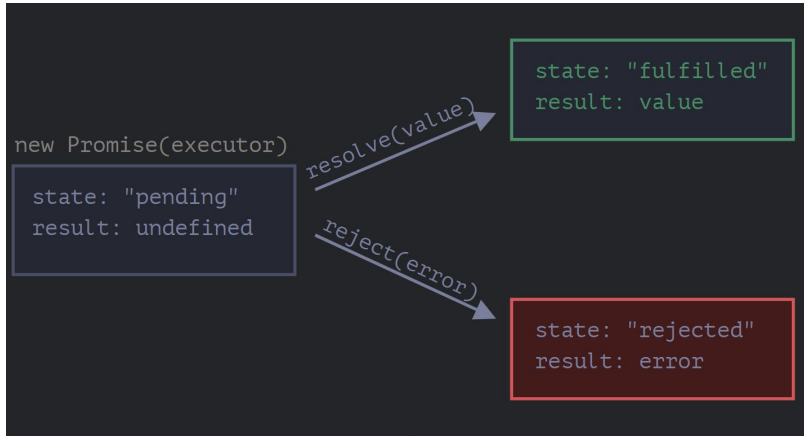
Contoh :

- Kita membuat code yang meminta data dari sebuah server(tidak tahu berapa lama waktu yang dibutuhkan untuk meminta data), code ini kita sebut sebagai Promise.
- Namun, kita harus memberitahu web, agar tahu apa yang terjadi, apakah code berhasil, atau tidak. Jika berhasil, apa yang dilakukan(kita sebut sebagai resolve) ? Begitu juga jika gagal, code apa yang dilakukan(kita sebut sebagai reject).



```
let janji = new Promise(function (resolve, reject) {  
  // executor  
});
```

- Pembuatan Promise, dilakukan dengan cara menuliskan **new Promise**.
- **function(resolve, reject){ //executor }** merupakan sebuah callback yang dilempar sebagai argumen, ketika **new Promise** dibuat, **executor** akan langsung jalan secara otomatis.
- Argumen **resolve**, dan **reject** juga merupakan sebuah **callback** yang disediakan oleh **JavaScript** secara otomatis.
- Ketika executor selesai dijalankan, pada akhirnya akan memanggil;
  - **resolve(value)** jika code selesai dijalankan, dengan result **value**.
  - **reject(error)** jika terjadi error, dengan result **error**.



Sumber: javascript.info

Ketika dibuat, new Promise memiliki built-in properties:

- **State:**  
Secara default, akan memiliki value "pending". Lalu akan berubah menjadi "fulfilled" ketika resolve dipanggil, dan "rejected" ketika reject dipanggil.
- **Result:**  
Secara default, akan memiliki value undefined. Lalu akan berubah menjadi value, ketika reject dipanggil, dan berubah menjadi error, ketika reject dipanggil.



## Consumers: then, catch

Untuk menggunakan promise, kita perlu memanggil;

- `then()`  
Sebuah function yang dijalankan ketika promise di resolved.
- `catch()`  
Sebuah function yang dijalankan ketika promise di reject






```
let janji = new Promise(function (resolve, reject) {  
  // executor  
  setTimeout(() => resolve('Done'), 1000);  
});  
  
janji.then((hasil) => {  
  console.log('[INI HASIL]', hasil);  
}).catch((error) => {  
  console.log('[INI ERROR]', error);  
});
```

[INI HASIL] Done

Jalankan code, dan harusnya akan menampilkan output seperti pada gambar yang di kanan. Kenapa `console.log('[INI ERROR]', error)` tidak dijalankan? Karena kita hanya memanggil **resolve** pada **new Promise**.



```
let janji = new Promise(function (resolve, reject) {  
  // executor  
  // setTimeout(() => resolve('Done'), 1000);  
  setTimeout(() => reject(new Error('Ops')), 1000);  
});  
  
janji.then((hasil) => {  
  console.log('[INI HASIL]', hasil);  
}).catch((error) => {  
  console.log('[INI ERROR]', error);  
});
```


```
[INI ERROR] Error: Ops  
    at Timeout._onTimeout (d:\Dimata\dimata-module\javascript\asynchronous\codes\index.js:12:26)  
    at listOnTimeout (node:internal/timers:573:17)  
    at process.processTimers (node:internal/timers:514:7)
```

Sekarang, coba comment code resolve, dan tambahkan code reject, lalu jalankan codenya. Maka akan menampilkan code seperti pada gambar yang di kanan.



# Finally

Finally merupakan function yang akan dijalankan ketika promise selesai, baik dalam kondisi resolve maupun reject.



```
let janji = new Promise(function (resolve, reject) {
  // executor
  // setTimeout(() => resolve('Done'), 1000);
  setTimeout(() => reject(new Error('Ops')), 1000);
});

janji.then((hasil) => {
  console.log('[INI HASIL]', hasil);
}).catch((error) => {
  console.log('[INI ERROR]', error);
}).finally(() => {
  console.log('Resolve maupun Reject tetap dijalankan');
});
```

```
[INI ERROR] Error: Ops
    at Timeout._onTimeout (d:\Dimata\dimata-module\javascript\asynchronous\codes\index.js:12:26)
    at listOnTimeout (node:internal/timers:573:17)
    at process.processTimers (node:internal/timers:514:7)
Resolve maupun Reject tetap dijalankan
```

Coba jalankan ketika resolve di comment dan reject di comment.



## Jadi, kapan promise dipakai?

Sabar, materi promise ini akan lebih mudah dipahami ketika kita telah memahami fetch. Namun, kita harus tahu terlebih dahulu promise ini. Materi fetch akan segera dimuat pada [repository dimata-module](#).