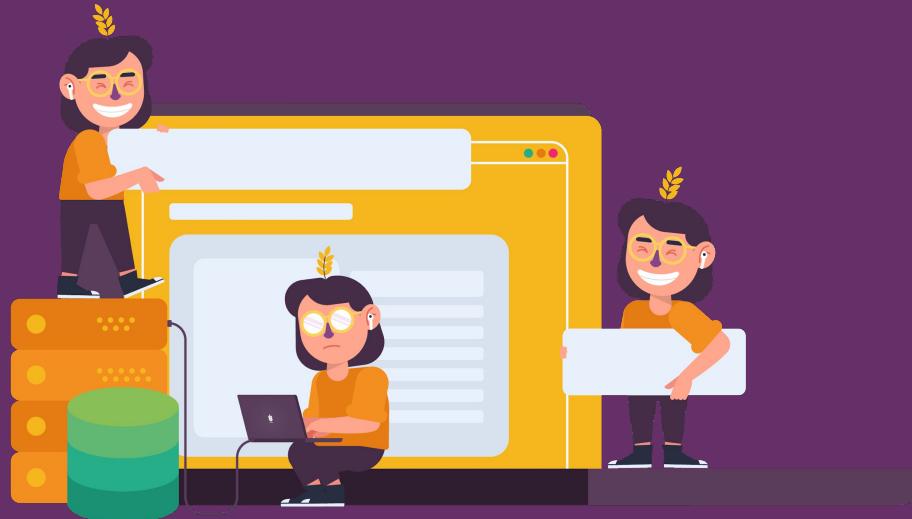




# **Synchronous & Asynchronous JavaScript**

## **Silver- Chapter 3 - Topic 2**

**Selamat datang di Chapter 3 Topic 2 pada course React Native dari Binar Academy!**





Halo 

Di Topic 1 kemarin kamu sudah belajar tentang Object Oriented Programming. Sekarang kita lanjut yuk ke **Chapter 3 Topic 2 React Native** tentang **Synchronous & Asynchronous JavaScript** yukk.

Kenapa kita perlu belajar materi ini? Karena materi ini bakalan penting untuk eksekusi kode-kode dalam JavaScript.

Yuk, langsung aja kita kepoin materi ini!





**Detailnya, kita bakal bahas hal-hal  
berikut ini:**

- Memahami Synchronous vs Asynchronous
- Cara implementasi Callback
- Cara implementasi Promise
- Cara implementasi Async Await





**Guys, bukan cuma manusia yang harus antre untuk bisa dilayani. Ternyata kode-kode pada Javascript juga lho~**

**Di materi ini kita akan bahas mengenai Synchronous dan Asynchronous pada Javascript**

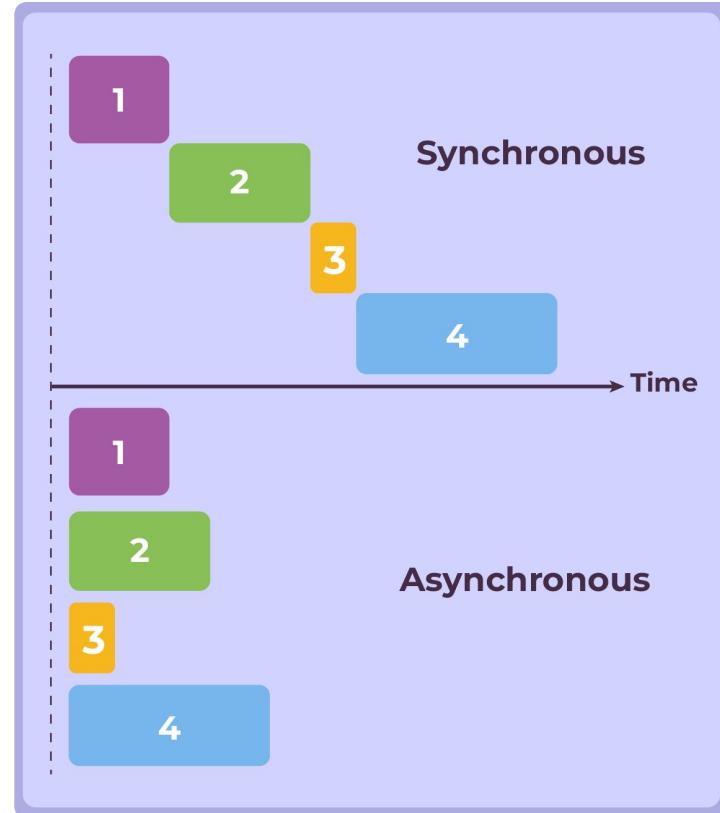




### Pertama-tama, kita kenalan dulu dengan istilah **Synchronous vs Asynchronous** yuk~

Dalam dunia programming, kedua istilah ini dipakai untuk membedakan tentang cara urutan dalam pengeksekusian perintah-perintah atau kode-kode pada Javascript.

Mari kita bedah satu per satu~





## Synchronous

Synchronous adalah konsep yang paling umum dan mudah dimengerti. Setiap **perintah atau kode dieksekusi satu persatu sesuai urutan kode** yang kamu tuliskan.

Output dari kode di samping dijamin akan sesuai urutan, karena setiap perintah/kode harus menunggu perintah/kode sebelumnya selesai.

Proses seperti ini disebut '**blocking**' yang artinya saling menunggu pengeksekusian kode sebelumnya selesai lebih dulu, baru kode pada baris selanjutnya akan dieksekusi.



```
1 console.log('Halo 1');
2 console.log('Haloo 2');
3 console.log('Halooo 3');
4
5
6 // Output console pertama => Halo 1
7 // Output console kedua => Haloo 2
8 // Output console ketiga => Halooo 3
```



**Mari kita perjelas konsep synchronous ini dengan sebuah analogi~**

Konsep synchronous bisa dianalogikan seperti cara restoran mengatur alur pelayanan ketika menerima sebuah pesanan, mulai dari pelanggan datang hingga pesanan sampai ke meja pelanggan tersebut.





## Step 1

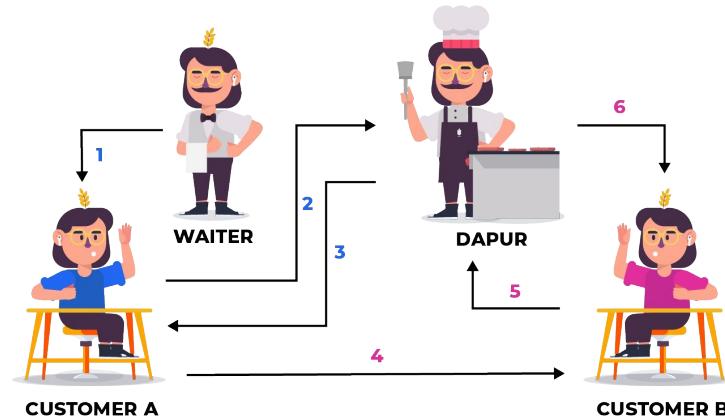
Ketika pelanggan datang ke suatu restoran, waiter akan langsung menghampiri dan membawa mereka ke tempat duduk yang masih kosong .

## Step 2

Setelah pelanggan memesan makanan, waiter akan menyampaikan informasi menu pesanan ke dapur.

## Step 3

Kemudian, menu pesanan pelanggan yang sudah jadi akan dibawa waiter dari dapur menuju meja pelanggan..



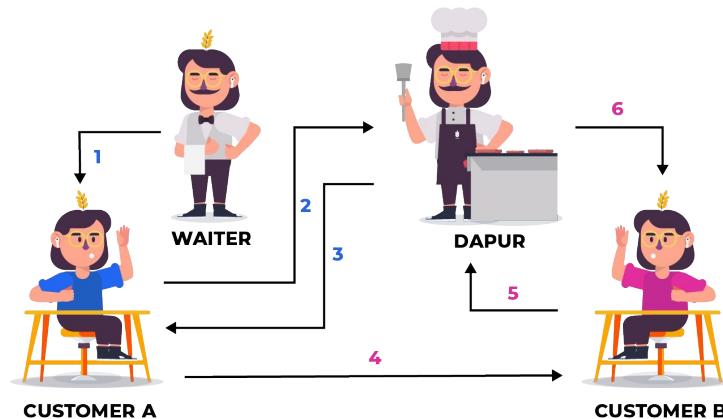


### Coba perhatikan ilustrasi di samping!

Kamu bisa lihat dari step 1 sampai step 3, waiter hanya melayani pesanan satu orang pelanggan saja, yaitu customer A. Setelah proses itu selesai, barulah waitress melayani customer berikutnya, yaitu customer B.

Nah, kita bisa simpulkan bahwa seorang waiter akan menunggu sampai satu pesanan selesai, baru dia akan melayani pesanan selanjutnya.

Dan begitulah gambaran konsep eksekusi kode di Javascript secara synchronous~





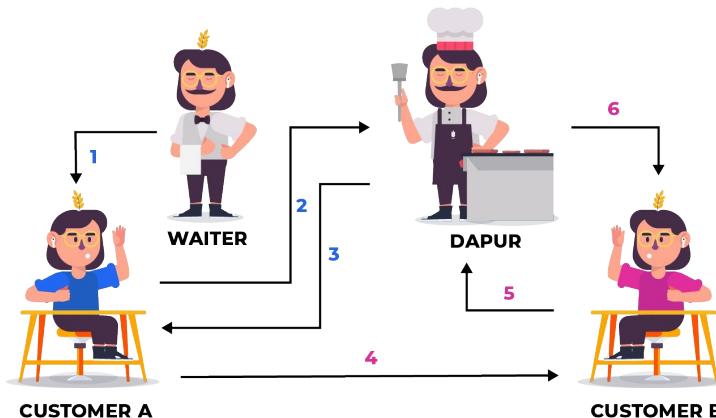
**Synchronous sudah paham kan, guys?**

**Yuk lanjut ke Asynchronous~**





## Analogi proses *Synchronous* pada kehidupan nyata



### Step 1

Ketika *customer* datang ke suatu restoran, *waiter* akan langsung menghampiri dan membawa mereka ke tempat duduk yang masih kosong .

### Step 2

Setelah *customer* memesan makanan, *waiter* akan menyampaikan informasi menu pesanan ke dapur.

### Step 3

Kemudian, menu pesanan *customer* yang sudah jadi akan dibawa *waiter* dari dapur menuju meja *customer*.

Nah, dari diagram disamping ini, bisa kamu lihat bahwa seorang *waiter* akan menunggu sampai satu pesanan selesai, karena dari step 1-3, *waiter* hanya melayani pesanan satu orang *customer* saja, yaitu *customer* A. Setelah proses itu selesai, barulah *waitress* melayani *customer* berikutnya, yaitu *customer* B.



### Asynchronous

Pada asynchronous, hasil eksekusi atau output nggak selalu berdasarkan urutan kode, tetapi berdasarkan waktu proses.

Eksekusi dengan asynchronous nggak akan membloking atau menunggu suatu perintah/kode sampai selesai.

Asynchronous **mengeksekusi perintah/kode selanjutnya, tanpa menunggu proses kode sebelumnya selesai.**





# Kalau dipikir-pikir, cara kerja asynchronous ini lebih efektif, guys!



Coba kamu perhatikan dulu kode di samping. Kelihatan kan kalau outputnya nggak berurutan sesuai input (kode)? Nah, ini karena **cara kerja asynchronous adalah berdasarkan waktu proses.**

Coba lihat lagi bagian output pada kode di samping. Bisa dilihat output 'Haloo 2' muncul pada urutan terakhir, padahal urutan kodenya menempati urutan kedua.



```
1 console.log('Halo 1');
2 setTimeout(() => console.log('Haloo 2'), 1000);
3 console.log('Halooo 3');
4
5 // Output console pertama => Halo 1
6 // Output console kedua => Halooo 3
7 // Output console ketiga => Haloo 2
8
```



Kenapa bisa terjadi ya?

Karena untuk menjalankan perintah/kode 'Haloo 2' membutuhkan waktu proses yang lebih lama dibandingkan perintah/kode 'Haloo 1' dan 'Haloo 2', guys!



```
1 console.log('Haloo 1');
2 setTimeout(() => console.log('Halooo 2'), 1000);
3 console.log('Halooo 3');
4
5 // Output console pertama => Haloo 1
6 // Output console kedua => Halooo 3
7 // Output console ketiga => Haloo 2
8
```



### Jadi, kesimpulannya adalah~

Jika salah satu eksekusi membutuhkan proses yang agak lama, maka sambil menunggu proses tersebut JavaScript akan mengeksekusi perintah selanjutnya.





### Mari kita perjelas konsep asynchronous ini dengan analogi~

Cara restoran mengatur alur pelayanan ketika menerima sebuah pesanan juga bisa kita gunakan untuk menjelaskan konsep asynchronous. Gimana tuh?

Begini..



### Step 1

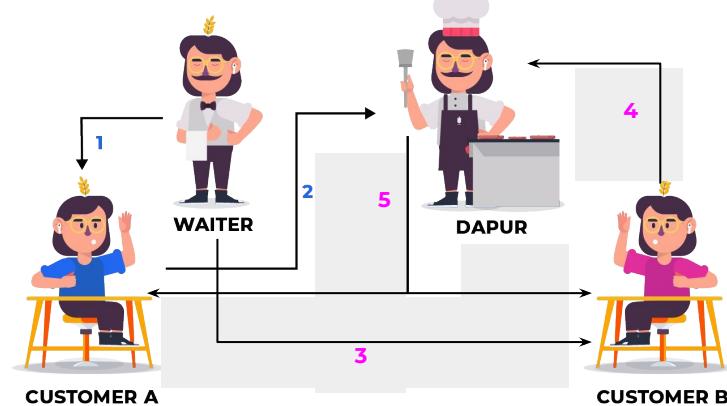
Ketika pelanggan datang ke suatu restoran, waiter akan langsung menghampiri dan membawa mereka ke tempat duduk yang masih kosong .

### Step 2

Setelah pelanggan memesan makanan, waiter akan menyampaikan informasi menu pesanan ke dapur.

### Step 3

Setelah waiter menyampaikan informasi menu pesanan ke dapur, ia nggak menunggu proses tersebut selesai, namun **waiter akan melayani pelanggan lain yang baru datang, yaitu customer B**.





Dari analogi tersebut, kamu tahu nggak di mana perbedaan mendasar antara proses synchronous dan asynchronous?

Yaps, perbedaannya ada di step ketiga!

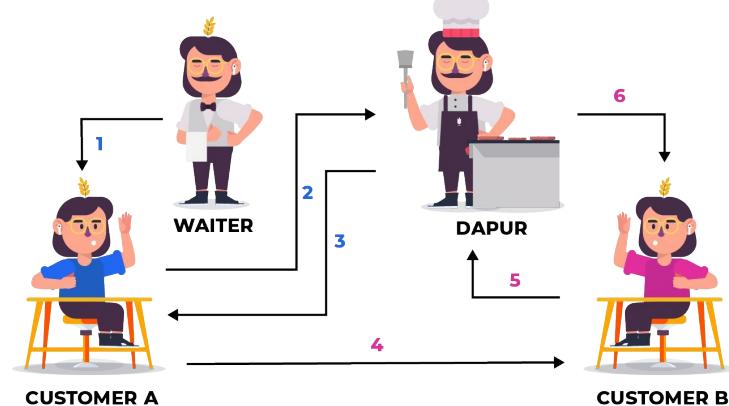




### Coba perhatikan gambar di samping deh..

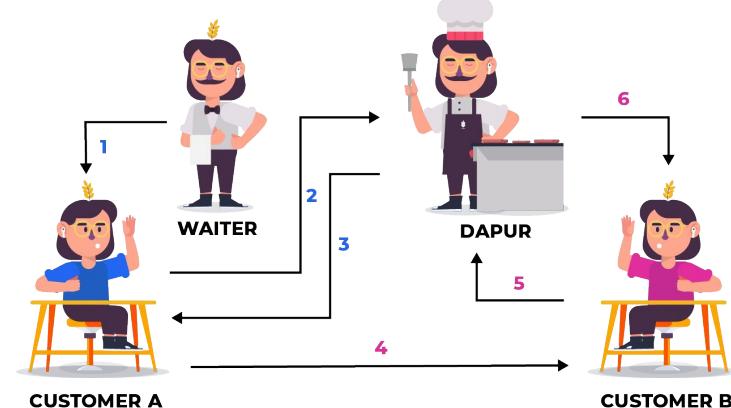
Waiter bisa melayani dua pesanan dari customer yang berbeda tanpa harus menunggu masakan dari customer pertama selesai. **Masakan yang lebih cepat dimasak akan diantar duluan ke customer.**

Dengan begitu, penyampaian informasi ke dapur nggak akan terhambat dan nggak akan terjadi penumpukan pesanan.





Jadi, **proses asynchronous ini lebih efisien** dibandingkan harus menunggu suatu proses selesai dulu, baru kemudian menerima informasi baru.





### Proses asynchronous akan sering kamu temui, pahami baik-baik ya~

Yupp, proses asynchronous akan sering kamu gunakan saat sudah masuk pada tahap komunikasi data.

Data akan jadi bagian inti dari sebuah aplikasi, maka konsep asynchronous sangat penting untuk kamu kuasai.





# Kalau dari tampilan kode ini bisa kelihatan bedanya kan~

### Synchronous

Pada contoh coding di bawah, dapat terlihat bahwa output console Halo 1, Halo 2 dan Halo 3 **tampil secara berurutan**. Berarti syntax-nya pun **dieksekusi secara berurutan**.



```
1 console.log('Halo 1');
2 console.log('Haloo 2');
3 console.log('Halooo 3');
4
5
6 // Output console pertama => Halo 1
7 // Output console kedua => Haloo 2
8 // Output console ketiga => Halooo 3
```

### Asynchronous

Pada contoh coding di bawah, dapat terlihat bahwa output console Halo 1, Halo 2 dan Halo 3 **tampil secara tidak berurutan**. Berarti syntax-nya pun **dieksekusi secara tidak berurutan**.



```
1 console.log('Halo 1');
2 setTimeout(() => console.log('Haloo 2'), 1000);
3 console.log('Halooo 3');
4
5 // Output console pertama => Halo 1
6 // Output console ketiga => Halooo 3
7 // Output console kedua => Haloo 2
8
```



### Jadi, kayak gini bedanya~

Output dari kode **synchronous process** dijamin akan sesuai urutan, karena setiap perintah harus menunggu perintah sebelumnya selesai. Proses seperti ini sering disebut dengan istilah **blocking**.

Sementara itu, output dari kode **asynchronous process** nggak selalu berdasarkan urutan kode, tetapi berdasarkan waktu proses.

Perintah	Waktu
P1	1
P2	3
P3	2

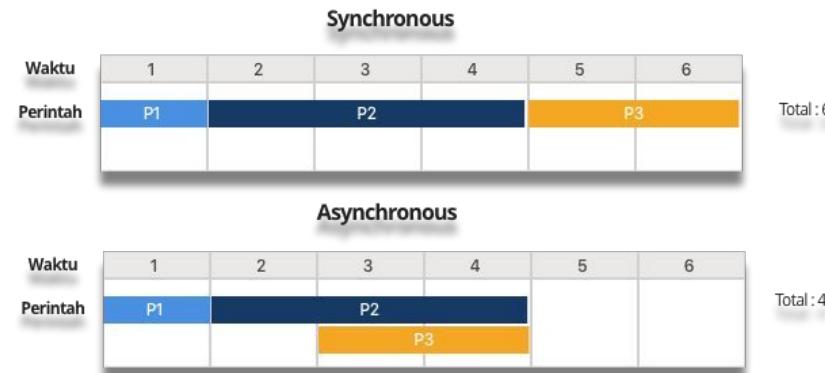


## Jadi semakin kelihatan ya kalau asynchronous ini lebih efisien~

Eksekusi kode dengan asynchronous nggak akan melakukan blocking atau dengan kata lain nggak akan menunggu suatu perintah sampai selesai.

Berdasarkan contoh kode yang tadi dapat kita lihat bahwa performance asynchronous process lebih efisien.

Untuk lebih jelasnya, lihatlah gambar di samping:





Meskipun begitu, ternyata ada lho cara yang bisa kamu lakukan kalau harus tetap menggunakan proses urutan eksekusi kode secara synchronous.

Tiga metode yang bisa digunakan yaitu **Callback, Promise, dan Async/Await**.





## Callback

Callback sebenarnya adalah function, bedanya dengan function pada umumnya adalah cara eksekusi/cara panggilnya.

Jika function yang kamu buat biasanya dieksekusi berurutan dari atas ke bawah, maka **function callback dieksekusi pada point tertentu**, itu sebabnya disebut callback.

```
1 // Kode dengan proses Asynchronous
2
3 function p1() {
4   console.log('perintah 1 done')
5 }
6 function p2(callbackFunction) {
7   //setTimeout atau delay digunakan
8   //untuk mensimulasikan kalo proses p2 butuh waktu lebih lama
9   //daripada proses p1 dan p3
10  setTimeout(
11    function() {
12      console.log('perintah 2 done')
13      callbackFunction()
14    },100
15  )
16 }
17 function p3() {
18   console.log('perintah 3 done')
19 }
20 p1()
21 p2(p3)
22
23 /* Output :
24 perintah 1 done
25 perintah 2 done
26 perintah 3 done
27 */
```



Callback disebut juga dengan high-order function. Jika function pada umumnya dieksekusi secara langsung, maka **callback dieksekusi dalam function lain melalui parameter.**

```
1 // Kode dengan proses Asynchronous
2
3 function p1() {
4   console.log('perintah 1 done')
5 }
6 function p2(callbackFunction) {
7   //setTimeout atau delay digunakan
8   //untuk mensimulasikan kalo proses p2 butuh waktu lebih lama
9   //daripada proses p1 dan p3
10  setTimeout(
11    function() {
12      console.log('perintah 2 done')
13      callbackFunction()
14    },100
15  )
16 }
17 function p3() {
18   console.log('perintah 3 done')
19 }
20 p1()
21 p2(p3)
22
23 /* Output :
24 perintah 1 done
25 perintah 2 done
26 perintah 3 done
27 */
```

## Ini bisa kita analogikan kayak pengumpulan tugas kampus lho~

Di mata kuliah Bahasa Indonesia, mahasiswa dikasih tugas membuat cerpen untuk dikerjakan di rumah masing-masing.

Dosen bilang, tugas itu dikumpulkan besok di kampus dan harus sesuai abjad nama.



## Pas ngerjain tugas~

Andi, Bayu dan Clara mulai ngerjain tugas ini di waktu yang bersamaan di rumah mereka masing-masing. Andi adalah orang yang paling cepat selesai di antara mereka bertiga, lalu Clara selesai kedua, sementara Bayu selesai paling akhir.





## Di hari pengumpulan tugas~

Berhubung nama Andi berawalan huruf "A", jadi Andi sudah ngumpulin tugasnya paling awal. Sementara Clara berhalangan hadir ke kampus di hari pengumpulan tugas.

Karena tugas ini harus dikumpulkan sesuai abjad, jadinya Clara menitipkan tugasnya ke Bayu. Sehingga tugas Clara akan dikumpulkan setelah Bayu menyelesaikan tugasnya terlebih dahulu.



## Di mana proses callback pada analogi??

Jadi, Clara adalah function callback, sementara Bayu adalah function tempat mengeksekusi callback.

Nah, proses ketika Clara menitipkan tugasnya ke Bayu itu adalah analogi dari cara kerja callback. Sehingga proses pengumpulan tugas tetap sesuai dengan abjad.

Kalau pada pemrograman, proses eksekusi function tetap sesuai dengan urutannya.



Function Eksekusi

Function Call Back



**Nah, sekarang mari kita coba  
praktekkan callback untuk bikin proses  
asynchronous jadi synchronous yuk!  
Biar kamu makin jago~**





## Kita mulai dari karakteristik asynchronous dan synchronous dulu nih

Proses **asynchronous identik dengan delay**, di mana hasil dari proses tersebut membutuhkan selang waktu tertentu untuk menghasilkan output.

Sementara **pada synchronous, output diproses berdasarkan urutan kode**.

Seperti gambar di samping, urutan output yang keluar sesuai dengan urutan perintah/kode yang dieksekusi, karena pada proses ini setiap perintah/kode memiliki waktu pemrosesan yang sama.

```
 1 // Kode dengan proses Synchronous
 2
 3 function p1() {
 4   console.log('perintah 1 done')
 5 }
 6 function p2() {
 7   console.log('perintah 2 done')
 8 }
 9 function p3() {
10   console.log('perintah 3 done')
11 }
12 p1()
13 p2()
14 p3()
15
16 /* Output :
17 perintah 1 done
18 perintah 2 done
19 perintah 3 done
20 */
```



**Ini adalah simulasi asynchronous dengan asumsi p2 memiliki proses waktu yang terlama dibanding p1 dan p3.**

Nah, karena proses asynchronous nggak menunggu proses perintah/kode sebelumnya selesai, maka yang akan terjadi adalah **urutan output yang dihasilkan bakal nggak sesuai dengan urutan perintah/kode yang ditulis**, karena Javascript mengerjakan mana yang selesai lebih dulu.

Ini karena pada proses p2 membutuhkan waktu yang lebih lama daripada proses lain, sehingga output p2 berada pada urutan terakhir.

```
1 // Kode dengan proses Asynchronous
2
3 function p1() {
4     console.log('perintah 1 done')
5 }
6 function p2() {
7     //setTimeout atau delay digunakan
8     //untuk mensimulasikan kalo proses p2 butuh waktu lebih lama
9     //daripada proses p1 dan p3
10    setTimeout(
11        function() {
12            console.log('perintah 2 done')
13        },100
14    )
15 }
16 function p3() {
17     console.log('perintah 3 done')
18 }
19 p1()
20 p2()
21 p3()
22
23 /* Output :
24 perintah 1 done
25 perintah 3 done
26 perintah 2 done
27 */
```



Terus gimana caranya agar kamu bisa menghandle proses asynchronous ini supaya tetap sesuai dengan urutan perintah/kode?

**Kamu bisa pakai callback sebagai salah satu solusinya.**





## Bagaimana caranya?

Kamu harus membuat p3 menjadi callback bagi p2, karena p2 butuh waktu proses yang lebih lama.

Dengan kata lain, kamu akan menyuruh JavaScript, untuk **menjalankan function p3 ketika proses console.log ('perintah 2 done') pada function p2 telah usai.**

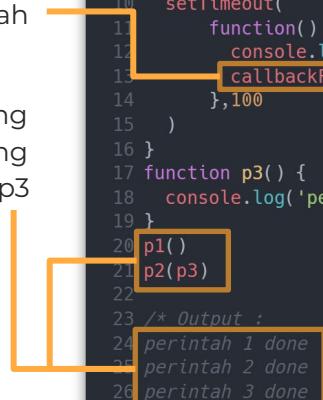
Jadi, kamu akan menjalankan function p3 di dalam function p2.

```
1 // Kode dengan proses Asynchronous
2
3 function p1() {
4   console.log('perintah 1 done')
5 }
6 function p2(callbackFunction) {
7   //setTimeout atau delay digunakan
8   //untuk mensimulasikan kalo proses p2 butuh waktu lebih lama
9   //daripada proses p1 dan p3
10  setTimeout(
11    function() {
12      console.log('perintah 2 done')
13      callbackFunction()
14    },100
15  )
16 }
17 function p3() {
18   console.log('perintah 3 done')
19 }
20 p1()
21 p2(p3)
22
23 /* Output :
24 perintah 1 done
25 perintah 2 done
26 perintah 3 done
27 */
```

## BEGINI PENJELASAN CODINGNYA~

Function p3 dijalankan ketika proses console.log pada p2 telah selesai.

Urutan output yang sesuai dengan urut perintah/code yang dieksekusi, walaupun proses p2 membutuhkan waktu yang lebih lama, dengan menggunakan callback kita membuat p3 menunggu proses p2 selesai.



```
1 // Kode dengan proses Asynchronous
2
3 function p1() {
4   console.log('perintah 1 done')
5 }
6 function p2(callbackFunction) {
7   //setTimeout atau delay digunakan
8   //untuk mensimulasikan kalo proses p2 butuh waktu lebih lama
9   //daripada proses p1 dan p3
10  setTimeout(
11    function() {
12      console.log('perintah 2 done')
13      callbackFunction()
14    }, 100
15  )
16 }
17 function p3() {
18   console.log('perintah 3 done')
19 }
20 p1()
21 p2(p3)
22
23 /* Output :
24 perintah 1 done
25 perintah 2 done
26 perintah 3 done
27 */
```



**Callback udah nih!**

**Yuk lanjut ke Promise. Kita bikin janji  
lewat sebuah kode, guys!**

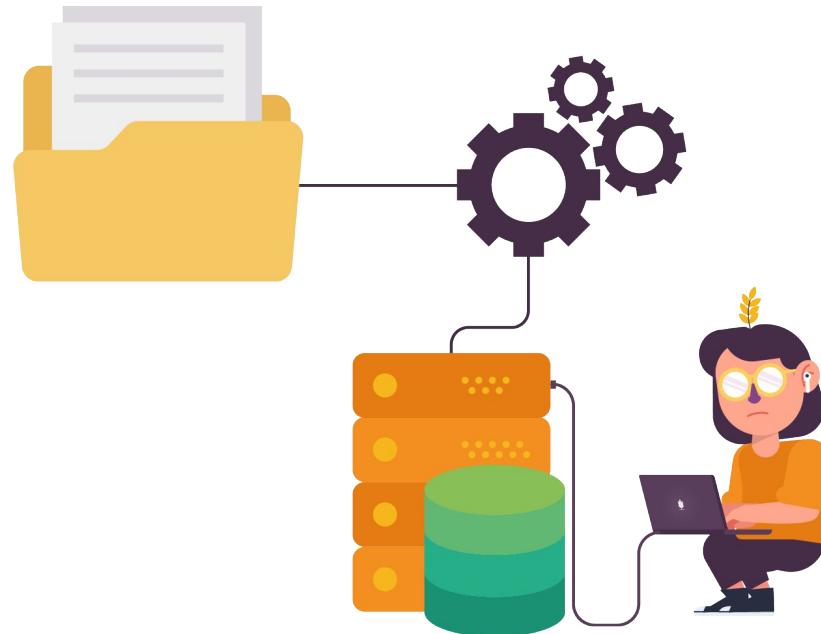
**Penasaran gimana caranya?? Cekidot~**





## Promise

Promise bisa dikatakan sebagai **object yang menyimpan hasil dari sebuah operasi asynchronous**, baik itu hasil yang diinginkan (resolved value) ataupun alasan kenapa operasi itu gagal (failure reason).





## Kita ambil contoh seperti saat kamu memesan ojek online deh..

Saat kamu mencari driver lewat aplikasi, **aplikasi akan berjanji (promise)** memberi tahu hasil dari pencarian kita.

Hasilnya bisa di antara dua, yaitu driver ditemukan (resolved value) atau alasan kenapa driver tidak ditemukan (failure reason).





Promise berada di salah satu di antara **3 kondisi (state)**:

- Pending**, operasi sedang berlangsung
- Fulfilled**, operasi selesai dan berhasil
- Rejected**, operasi selesai namun gagal

Sama kayak kasus memesan ojek online, status permintaan pada aplikasi online ada di antara tiga kondisi:

- Mencari driver (pending)
- Menemukan driver (fulfilled)
- Driver tidak ditemukan (rejected)



Pending



Fulfilled



Rejected



## Yuk, kita belajar cara bikin Promise!

Untuk membuat promise, kamu bisa menggunakan keyword new Promise.

Function setelah keyword new Promise disebut **executor**. Di dalam executor terdapat dua callback function:

- **resolve (value)** adalah callback function yang dieksekusi jika operasi yang dijalankan oleh executor berhasil (fulfilled).
- **reject (error)** adalah callback function yang akan dieksekusi jika operasi gagal (rejected).



```
1 const progress = 100;
2
3 const download = new Promise((resolve, reject) => {
4   if (progress === 100) {
5     resolve('Download complete');
6   } else {
7     reject('Download failed');
8   }
9});
```



## FYI, pemanggilan variabel juga ada aturannya yaa. Apa tuh?

Kamu nggak bisa memanggil variabel yang menampung promise secara langsung, guys. Kalau kamu melakukan ini, maka **output yang dihasilkan adalah sebuah object Promise { <pending> }**.

Hal ini terjadi karena blok kode yang ada di dalam Promise belum selesai diproses, sementara variabel download telah dipanggil duluan pada console.log.



```
1 let progress = 100;
2
3 const download = new Promise((resolve, reject) => {
4   if (progress == 100) {
5     resolve('Download complete');
6   } else {
7     reject('Download failed');
8   }
9 });
10
11 console.log(download);
12 // Output => Promise { <pending> }
13
```



## Lalu, bagaimana cara akses data dari object promise?

Untuk mengakses data dari sebuah variabel yang berbentuk promise, kamu bisa **menggunakan method .then()** pada variabel yang mengandung promise.

Setiap value yang dimasukan pada function resolve() yang berada pada promise, dapat diakses pada method .then().

Sedangkan setiap value yang dimasukan pada function reject() yang berada pada promise dapat diakses pada method catch().



```
1 let progress = 100;
2
3 const download = new Promise((resolve, reject) => {
4   if (progress == 100) {
5     resolve('Download complete');
6   } else {
7     reject('Download failed');
8   }
9 });
10
11 download.then((parameterResolve) => {
12   console.log(parameterResolve);
13   // Output => 'Download complete'
14 });
15
```



## Begini penjelasan cara kerja pada coding di samping~

Karena kondisi yang terpenuhi berada di dalam blok 'if' maka method **resolve()** yang akan tereksekusi.

Dan karena yang tereksekusi pada blok promise adalah method resolve, maka ketika variabel yang mengandung promise dipanggil, yang akan jalan adalah method **then()**.

Method **.catch()**, nggak akan dijalankan karena pada object promise, method yang dijalankan adalah method **resolve()**. Method **.catch()** hanya akan dijalankan apabila method reject tereksekusi pada object promise.

```
 1 let progress = 100;
 2
 3 const download = new Promise((resolve, reject) => {
 4   if (progress == 100) {
 5     resolve('Download complete');
 6   } else {
 7     reject('Download failed');
 8   }
 9 });
10
11 download
12   .then((parameterResolve) => {
13     console.log(parameterResolve);
14   })
15   .catch((parameterReject) => {
16     console.log(parameterReject);
17   });
18
```



The diagram illustrates the execution flow of the code. Orange arrows point from the 'if' condition in line 4 to the 'resolve' call in line 5, and from the 'else' condition to the 'reject' call in line 7. Another orange arrow points from the 'download' variable in line 11 to the '.then' block in line 12. A large red 'X' is placed over the entire '.catch' block in line 15, indicating that it will not be executed under the current conditions.

Lalu gimana cara **implementasi promise** untuk bikin proses asynchronous jadi synchronous ya??

Nah, sekarang mari kita coba praktekkan aja yuk! Biar kamu makin jago~





## Oke, kita balik sebentar ke contoh cara mengatasi kode asynchronous ini

Sebelumnya kamu kan sudah coba menggunakan callback, sekarang **mari coba ubah kode di samping, dari proses asynchronous menjadi proses synchronous menggunakan promise.**

```
// Kode dengan proses Asynchronous
2
3 function p1() {
4     console.log('perintah 1 done')
5 }
6 function p2() {
7     //setTimeout atau delay digunakan
8     //untuk mensimulasikan kalo proses p2 butuh waktu lebih lama
9     //daripada proses p1 dan p3
10    setTimeout(
11        function() {
12            console.log('perintah 2 done')
13        },100
14    )
15 }
16 function p3() {
17     console.log('perintah 3 done')
18 }
19 p1()
20 p2()
21 p3()
22
23 /* Output :
24 perintah 1 done
25 perintah 3 done
26 perintah 2 done
27 */
```



## Begini penjelasan cara kerja pada coding di samping~

Letakan semua kode pada function p2 di dalam sebuah blok Promise.

**Ketika proses console.log('perintah 2 done') selesai,** kamu **trigger method resolve()** untuk menjalankan kode selanjutnya yang berada pada method then(), kalau di sini adalah function p3 ya guys!

Jadi, fungsi method resolve() adalah untuk mentrigger method then() agar menjalankan semua kode pada method tersebut, sehingga proses selanjutnya (function p3) dapat dijalankan.

```
1 // Kode dengan proses Asynchronous
2 function p1() {
3   console.log('perintah 1 done');
4 }
5 function p2() {
6   // Proses P2 karena butuh waktu lebih lama
7   // Kita masukan ke dalam promise
8   return new Promise((resolve, reject) => {
9     setTimeout(function () {
10       console.log('perintah 2 done');
11       resolve();
12     }, 100);
13   });
14 }
15 function p3() {
16   console.log('perintah 3 done');
17 }
18 p1();
19 p2().then(() => {
20   p3();
21 });
22
23 /* Output :
24 perintah 1 done
25 perintah 2 done
26 perintah 3 done
27 */
28
```



Nah, sekarang kita sampai di sub topik terakhir, yaitu **Async/Await**.

Cekidot~

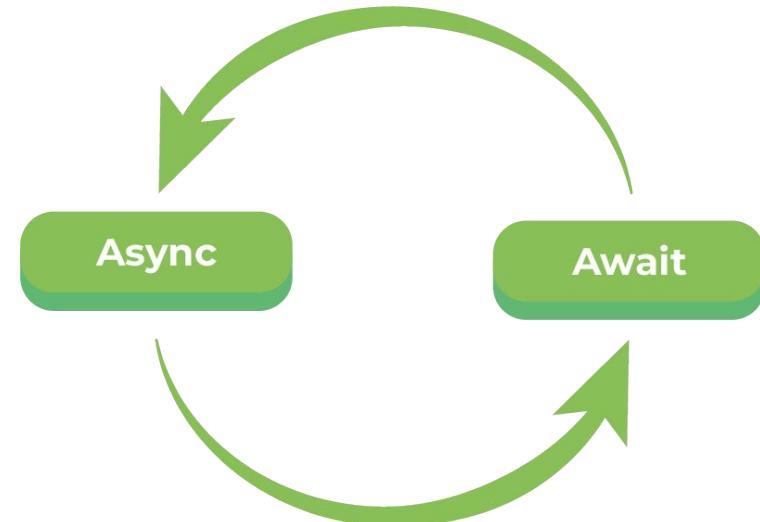




## Async/Await

Async/await adalah fitur yang mempermudah dalam menangani proses asynchronous. Ada 2 kata kunci di sini yaitu **async** dan **await**.

**'Async'** akan mengubah function menjadi asynchronous, sementara **'await'** akan menunda eksekusi hingga proses asynchronous selesai.





## Biar lebih jelas, kita simulasikan dengan coding aja ya~

Keyword `async` yang ada di awal keyword `function` berfungsi untuk **mengubah function jadi asynchronous function**.



```
1 // async function
2 async function asyncFunc() {
3   // tunggu proses promises
4   let resolvedValue = await promiseData;
5
6   console.log(resolvedValue);
7   console.log('hello');
8 }
9
10 // calling the async function
11 asyncFunc();
```



Sementara itu, keyword await berfungsi **menunda eksekusi pada data yang berupa object Promise.**

Dari kode di samping berarti `console.log(resolvedValue)` nggak akan dieksekusi sebelum proses `promiseData()` selesai. Await juga bisa digunakan berkali-kali di dalam function.



```
1 // async function
2 async function asyncFunc() {
3   // tunggu proses promises
4   let resolvedValue = await promiseData;
5
6   console.log(resolvedValue);
7   console.log('hello');
8 }
9
10 // calling the async function
11 asyncFunc();
```



Oh iya, keyword await cuma bisa dipakai di dalam async function ya. Jadi kalau digunakan tanpa async atau di luar async function, maka akan error.

```
1 // async function
2 async function asyncFunc() {
3   // tunggu proses promises
4   let resolvedValue = await promiseData;
5
6   console.log(resolvedValue);
7   console.log('hello');
8 }
9
10 // calling the async function
11 asyncFunc();
```



Sekarang, mari lihat perbandingan cara menghandle promise menggunakan method **.then()** vs menggunakan **async/await**

### Menggunakan .then()

```
1 // data prmoise
2 const promiseData = new Promise(function (resolve, reject) {
3   setTimeout(function () {
4     resolve('Promise resolved');
5   }, 4000);
6 });
7
8 // handle promise menggunakan .then
9 promiseData.then((resolvedValue) => {
10   console.log(resolvedValue);
11   console.log('hello');
12 });
13
```

### Menggunakan async/await, sehingga nggak perlu pakai method 'then()' lagi.

```
1 // data prmoise
2 const promiseData = new Promise(function (resolve, reject) {
3   setTimeout(function () {
4     resolve('Promise resolved');
5   }, 4000);
6 });
7
8 // handle promise menggunakan async function
9 async function asyncFunc() {
10   // tunggu proses promises
11   let resolvedValue = await promiseData;
12   console.log(resolvedValue);
13   console.log('hello');
14 }
15
16 // panggil async function
17 asyncFunc();
18
```



**Yeaayy!! Selamat yaa, guys! Akhirnya kamu berhasil menuntaskan Topik 2 tentang Asynchronous ini. Biar makin jago, yuk kerjain quiz berikut ini!**



# Saatnya kita Quiz!





## 1. Apa output dari syntax kode di bawah?

- A. I have resolved!, second and I have resolved!, second
- B. I have resolved!, second and second, I have resolved!
- C. second, I have resolved! and I have resolved!, second

● ● ●

```
1 const myPromise = () => Promise.resolve('I have resolved!');
2
3 function firstFunction() {
4   myPromise().then(res => console.log(res));
5   console.log('second');
6 }
7
8 async function secondFunction() {
9   console.log(await myPromise());
10  console.log('second');
11 }
12
13 firstFunction();
14 secondFunction();
```



## 1. Apa output dari syntax kode di bawah?

- A. I have resolved!, second and I have resolved!, second
- B. I have resolved!, second and second, I have resolved!
- C. second, I have resolved! and I have resolved!, second

• • •

```
1 const myPromise = () => Promise.resolve('I have resolved!');
2
3 function firstFunction() {
4   myPromise().then(res => console.log(res));
5   console.log('second');
6 }
7
8 async function secondFunction() {
9   console.log(await myPromise());
10  console.log('second');
11 }
12
13 firstFunction();
14 secondFunction();
```

second, I have resolved! and I have resolved!, second



## 2. Apa output dari syntax kode di bawah?

- A. Promise {<pending>}
- B. Undefined
- C. “I made it”



```
1 async function getData() {
2   return await Promise.resolve('I made it!');
3 }
4
5 const data = getData();
6 console.log(data);
```



## 2. Apa output dari syntax kode di bawah?

- A. Promise {<pending>}
- B. Undefined
- C. “I made it”



```
1 async function getData() {
2   return await Promise.resolve('I made it!');
3 }
4
5 const data = getData();
6 console.log(data);
```

Sebuah `async function` akan selalu mereturn/menghasilkan data berbentuk `objectPromise`. Jadi apabila kamu ingin mendapatkan value “I made it” di luar `async function`, maka kamu bisa menggunakan method `then` pada variable `data` seperti `data.then(res => console.log(res))`



- 3. Urutan eksekusi kode dengan asynchronous tidak akan menunggu suatu perintah/kode sampai selesai. Hal ini dikenal dengan istilah?**
- A. Blocking
  - B. Non-blocking
  - C. Half-blocking



- 3. Urutan eksekusi kode ngan asynchronous tidak akan menunggu suatu perintah/kode sampai selesai. Hal ini dikenal dengan istilah?**
- A. Blocking
  - B. Non-blocking
  - C. Half-blocking

Sebuah process yang di-assign ke suatu thread tanpa menunggu process tersebut selesai dijalankan disebut Non-Blocking, sedangkan jika harus menunggu process sebelumnya sampai selesai dijalankan disebut Blocking.



## 4. Apa output dari syntax kode di bawah?

- A. Reject state: Not a suspect
- B. Reject state: Please fill the correct temp!
- C. Resolved state: Covid symptom!



```
1 function isCovid(temperature) {  
2   return new Promise((resolve, reject) => {  
3     console.log('Temperature:', temperature);  
4     if (temperature <= '38') {  
5       return reject('Not a suspect');  
6     } else if (temperature === undefined) {  
7       return reject('Please fill the correct temp!');  
8     } else {  
9       resolve('Covid symptom!');  
10    }  
11  });  
12}  
13  
14 isCovid()  
15   .then((resolve) => console.log(resolve))  
16   .catch((reject) => console.error(reject));  
17
```



## 4. Apa output dari syntax kode di bawah?

- A. Reject state: Not a suspect
- B. Reject state: Please fill the correct temp!
- C. Resolved state: Covid symptom!

```
1 function isCovid(temperature) {  
2   return new Promise((resolve, reject) => {  
3     console.log('Temperature:', temperature);  
4     if (temperature <= '38') {  
5       return reject('Not a suspect');  
6     } else if (temperature === undefined) {  
7       return reject('Please fill the correct temp!');  
8     } else {  
9       resolve('Covid symptom!');  
10    }  
11  });  
12}  
13  
14 isCovid()  
15   .then((resolve) => console.log(resolve))  
16   .catch((reject) => console.error(reject));  
17
```

Output-nya kondisi reject yang kedua yaitu Please fill the correct temp!. Karena tidak ada argumen atau nilai temperature di dalam function isCovid().



**5. Di bawah ini yang aman digunakan untuk mengatasi output yang menghasilkan error saat implementasi async-await?**

- A. try{} catch(err){} block
- B. try{} (err) {} block
- C. try {catch(err){}} block



**5. Di bawah ini yang aman digunakan untuk mengatasi output yang menghasilkan error saat implementasi async-await?**

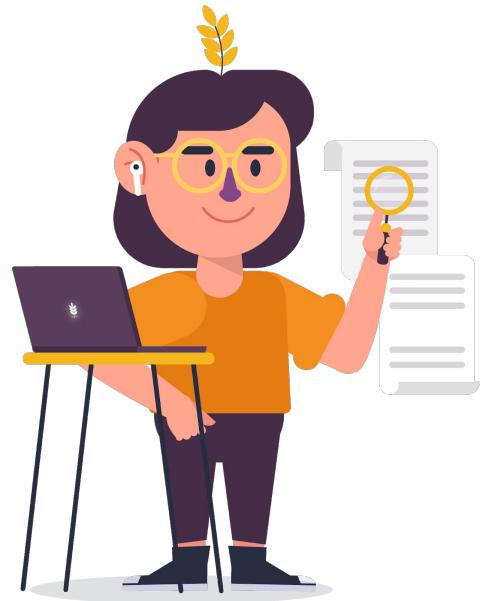
- A. try{} catch(err){} block
- B. try{} (err) {} block
- C. try {catch(err){}} block

Format yang benar adalah **try{} catch(err){} block**



### Referensi:

- <https://medium.com/coderupa/panduan-komplit-asynchronous-programming-pada-javascript-part-4-async-await-fc504c344238>
- <https://devsaurus.com/javascript-asynchronous#promise>



# Terima Kasih!



Next Topic

loading...