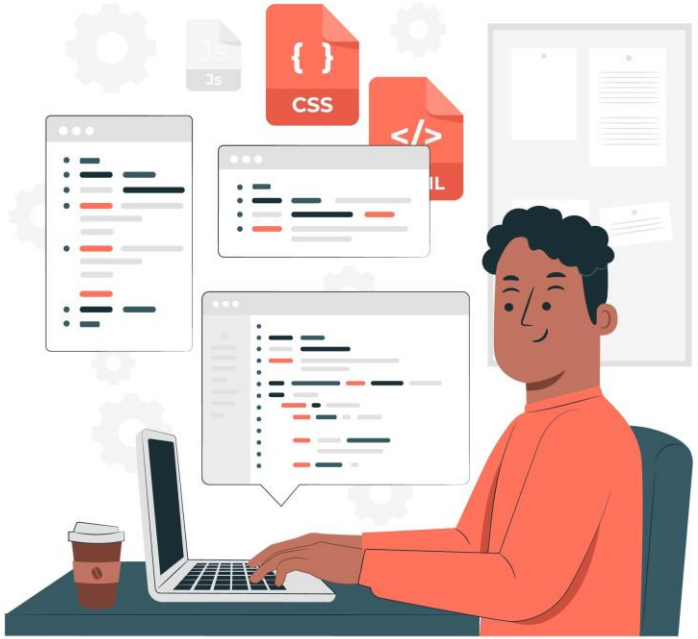
Modern JavaScript: Async



KEMNAKER

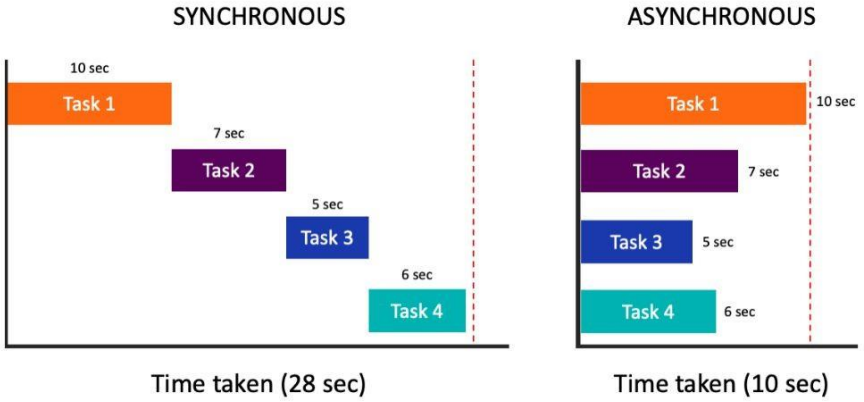
Outline

- Asynchronous

Asynchronous

Asynchronous Javascript

Di dalam dunia pemrograman terdapat dua cara dalam menjalankan program: Synchronous dan Asynchronous. Synchronous artinya program berjalan secara berurutan sedangkan Asynchronous artinya program berjalan bersama-sama. Perhatikanlah ilustrasi dibawah ini :



Apakah Javascript bahasa yang Synchronous saja?

JavaScript adalah bahasa pemrograman yang **secara alamiah bersifat synchronous**.
Secara default, javascript diproses dalam baris perbaris, artinya setiap baris kode tidak akan dieksekusi sebelum baris kode sebelumnya selesai diproses. Semua proses akan ditampung dan dikenal dengan istilah **call stack**.

Namun, JavaScript juga **mendukung operasi asynchronous, yang memungkinkan eksekusi kode untuk melanjutkan tanpa harus menunggu operasi tertentu selesai**. Terutama terjadi ketika kita berurusan dengan operasi I/O (Input/Output), seperti membaca data dari disk atau mengambil data dari jaringan

Contoh Kode Javascript yang menjadi Asynchronous

Perhatikan contoh program di bawah ini:

```
setTimeout(function() {
  console.log("saya dijalankan belakangan")
}, 3000)

console.log("saya dijalankan pertama")
```

Jika kita jalankan program di atas, maka yang akan tampil terlebih dahulu di console adalah “saya dijalankan pertama” walaupun sintaksnya ditulis belakangan setelah function setTimeout. Function setTimeout di atas merupakan salah satu contoh function asynchronous di Javascript.

Dalam skenario-nya, sebenarnya kode async telah diproses, hanya saja sebatas penjadwalan untuk dieksekusi pada tahapan berikutnya. Artinya, kode yang berperilaku async tidak akan langsung dieksekusi, tetapi di skip dan akan melakukan eksekusi baris perintah berikutnya.

Cara mengatasi Asynchronous di Javascript

Cara untuk mengatasi Asynchronous seperti function setTimeout adalah dengan Callback atau dengan Promise.

Callback

Apa itu Callback Function?

"I will call back later!"

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

https://www.w3schools.com/js/js_callback.asp

Penjelasan lebih lanjut:
https://id.javascript.info/callbacks

Apa itu Callback Function?

- **Callback** artinya "dipanggil kembali".
- **Callback function** adalah **fungsi** yang **dikirimkan sebagai argumen** ke fungsi lain, lalu akan **dieksekusi** (dipanggil) oleh fungsi tersebut pada waktu tertentu.

Jadi, ini seperti **memberikan nomor telepon** ke teman dan berkata:
"Kalau sudah sampai rumah, telepon aku ya."
Telepon = callback function.

Kenapa Dipakai di JavaScript?

Karena JavaScript itu **non-blocking** (nggak nunggu satu proses selesai dulu), callback sering dipakai untuk:

- Menjalankan kode **setelah** proses tertentu selesai (misalnya loading data dari server).
- Mengatur **urutan eksekusi**.

Contoh Sederhana (Sinkron)

```
function sapa(nama) {
  console.log(`Halo, ${nama}`);
}

function prosesNama(callback) {
  let nama = "Cika";
  callback(nama); // memanggil fungsi yang dikirim
}

prosesNama(sapa);
```

Penjelasan:

- sapa adalah fungsi biasa yang menerima parameter nama.
- prosesNama menerima parameter bernama callback (ini akan diisi oleh fungsi sapa).
- Di dalam prosesNama, kita memanggil callback(nama) → berarti memanggil sapa("Cika").

Contoh Sederhana (Sinkron)

```
//contoh 2
// Fungsi utama yang menerima dua angka dan sebuah callback 'operasi'
function hitung(angka1, angka2, operasi) {
  console.log(`Menjalankan perhitungan untuk ${angka1} dan ${angka2}...`);
  // 'operasi' adalah callback yang kita panggil di sini
  const hasil = operasi(angka1, angka2);
  console.log(`Hasilnya adalah: ${hasil}`);
}

// Ini adalah fungsi-fungsi yang akan kita jadikan callback
function tambah(a, b) {
  return a + b;
}

function kali(a, b) {
  return a * b;
}

// Panggil fungsi 'hitung' dan teruskan fungsi 'tambah' sebagai argumen (callback)
hitung(10, 5, tambah); // Output: Hasilnya adalah: 15

// Panggil fungsi 'hitung' dan teruskan fungsi 'kali' sebagai argumen (callback)
hitung(10, 5, kali);   // Output: Hasilnya adalah: 50
```

Contoh Asinkron (Misalnya pakai setTimeout)

```
function selesai() {
  console.log("Proses selesai!");
}

console.log("Mulai proses...");
setTimeout(selesai, 2000); // 2000ms = 2 detik
console.log("Proses lain berjalan...");
```

Alurnya:

- console.log("Mulai proses...") langsung dijalankan.
- setTimeout(selesai, 2000) akan menunggu 2 detik, lalu memanggil selesai.
- console.log("Proses lain berjalan...") langsung dijalankan tanpa menunggu.
- Setelah 2 detik, selesai() dipanggil → muncul "Proses selesai!".

Contoh Asinkron (Misalnya pakai setTimeout)

```
//contoh 2
console.log("Program dimulai.");

// Fungsi 'setTimeout' menerima callback dan waktu tunda (dalam milidetik)
setTimeout(function() {
  // Fungsi ini adalah callback, ia tidak langsung dijalankan
  console.log("Ini pesan dari dalam callback! Muncul setelah 2 detik.");
}, 2000); // 2000 ms = 2 detik

console.log("Program selesai.");
```

Callback dengan Fungsi Anonim

```
setTimeout(function() {
  console.log("Halo dari callback anonim!");
}, 1000);
```

- Kita tidak selalu perlu membuat fungsi terpisah. Bisa langsung bikin di tempat.
- Di sini fungsi tidak punya nama (**anonymous function**) tapi tetap jadi callback.

Ilustrasi Sehari-hari

Bayangkan kamu pesan makanan online:
1.Kamu memesan burger.
2.Kamu bilang ke kasir: *"Kalau burger sudah jadi, panggil saya."*
→ Kasir menyimpan **callback** (instruksi untuk memanggilmu).
3.Sambil menunggu, kamu main HP (JavaScript tetap mengerjakan tugas lain).
4.Setelah burger jadi, kasir memanggilmu (menjalankan callback function).

Ilustrasi Sehari-hari

```
console.log("1. Pesan burger...");
setTimeout(() => {
  console.log("3. Burger sudah jadi!");
}, 2000);
console.log("2. Sambil nunggu, main HP...");
```

[Step 1] "Pesan burger..."

↓

[Step 2] setTimeout() dipanggil

↓

- Timer mulai jalan (2 detik)
- Callback disimpan di "antrian tugas" (task queue)

↓

[Step 3] "Sambil nunggu, main HP..."

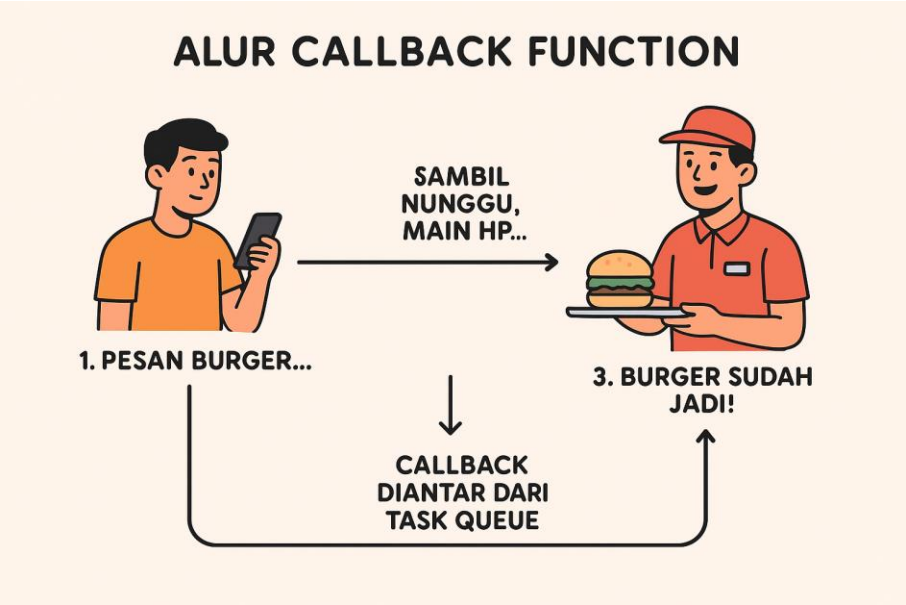
↓

[Step 4] Setelah 2 detik, callback diambil dari task queue

↓

[Step 5] "Burger sudah jadi!" dicetak

Ilustrasi Sehari-hari



Mekanisme di Balik Layar (Event Loop):

1. Call Stack → Menjalankan kode utama.
2. Web APIs → Menangani setTimeout, AJAX, dll.
3. Task Queue → Menyimpan callback yang siap dijalankan.
4. Event Loop → Mengecek apakah stack kosong, lalu ambil callback dari queue.

Contoh Callback dalam aplikasi antrian periksa dokter

1

```
// Deklarasi function yang memiliki callback sebagai parameter
function periksaDokter(nomerAntri, callback) {
  if(nomerAntri > 50 ) {
      callback(false)
  } else if(nomerAntri < 10) {
      callback(true)
  }
}
```

- **Parameter**:
  - nomerAntri → nomor urut pasien.
  - callback → fungsi yang akan dipanggil dengan hasil true atau false.
- **Logika dalam fungsi**:
  - Jika nomor antrian **lebih dari 50**, panggil callback(false) → artinya terlalu lama menunggu.
  - Jika nomor antrian **kurang dari 10**, panggil callback(true) → artinya sebentar lagi giliran.
  - Kalau nomor antriannya di **antara 10–50**, callback tidak terpanggil (kode ini sebenarnya kurang lengkap karena tidak menangani semua kasus).

Contoh Callback dalam aplikasi antrian periksa dokter

Setelah deklarasi function yang memiliki callback, kini kita jalankan function callback tersebut.

2

```
// Menjalankan function periksaDokter yang sebelumnya sudah dideklarasi
periksaDokter(65, function(check) {
  if(check) {
      console.log("sebentar lagi giliran saya")
  } else {
      console.log("saya jalan-jalan dulu")
  }
})
```

- Kita memanggil periksaDokter(65, ...).
- Di dalam periksaDokter, karena 65 > 50, maka dijalankan callback(false).
- Callback (function(check){...}) dipanggil dengan nilai check = false.
- Maka kode yang dieksekusi:

  console.log("saya jalan-jalan dulu")

Contoh Callback dalam aplikasi antrian periksa dokter

Atau bisa ditulis seperti ini

CallbackPeriksaDokter.js

Asinkronus > Callback > CallbackPeriksaDokter.js > ...

```
1   //deklarasi function yang memiliki callback sebagai parameter
2   function periksaDokter(nomorAntri, callback){
3     if (nomorAntri > 10){
4       callback(false)
5     }else{
6       callback(true)
7     }
8   }
9
10  // Setelah deklarasi function yang memiliki callback, kini kita jalankan function callback tersebut.
11  //menjalankan function periksaDokter yang sebelumnya sudah dideklarasi
12  //callback = kurangDariSepuluh
13  periksaDokter(20, function(kurangDariSepuluh){
14    if (kurangDariSepuluh){
15      console.log("saya menunggu di klinik saja")
16    }else{
17      console.log("saya pergi dulu sebentar")
18    }
19  })
```

batas toleransi pasien untuk nunggu dokter adalah nomor 10.

•Kalau nomor antrian **≤ 10** → pasien rela nunggu di klinik.

•Kalau nomor antrian **> 10** (contohnya 20) → pasien milih pergi dulu, mungkin balik lagi nanti.

Contoh Callback dalam aplikasi antrian periksa dokter + setTimeout di dalamnya:

1

```
function periksaAntrianDokter(nomerAntri, callback) {
  console.log(`sekarang antrian ke-${nomerAntri}`)
  setTimeout(function () {
    if(nomerAntri === 10 ) {
      console.log("saya masuk ruangan dokter")
      callback(0)
    } else {
      console.log("saya masih menunggu")
      callback(nomerAntri+1)
    }
  }, 1000)
}
```

Skenario :
Nomor urut pasien 10.

cara menggunakan callback dengan setTimeout di dalamnya adalah seperti ini:

2

```
periksaAntrianDokter(7, function(nomorAntriBaru){
  return nomorAntriBaru
});
```

Jika menggunakan kode di atas maka pada terminal akan muncul

```
sekarang antrian ke-7
saya masih menunggu
```

lalu jika kita ingin membuat kondisi bisa masuk ke ruangan dokter maka kita coba seperti contoh dibawah ini:

3

```
periksaAntrianDokter(7, function(nomorAntriBaru){
  periksaAntrianDokter(nomorAntriBaru, function(nomorAntriBaru1){
    periksaAntrianDokter(nomorAntriBaru1, function(nomorAntriBaru2){
      periksaAntrianDokter(nomorAntriBaru2, function(nomorAntriBaru3){
        return nomorAntriBaru3
      })
    })
  })
});
```

jika menggunakan kode diatas maka akan sampai ke “saya masuk ruangan dokter”

Waspadai Callback Hell / Pyramid of Doom

Jika kita menggunakan contoh pada kode sebelumnya makan akan berhasil tetapi kita perlu memanggil fungsi periksaAntrianDokter sekitar 4 kali berulang, bagaimana jika kita harus melakukan hal tersebut lebih dari itu bahkan ribuan kali ?

Callback Hell / Pyramid of Doom

- **Callback Hell** terjadi ketika kita punya **callback di dalam callback** berkali-kali.
- Akhirnya kode jadi:
  - **Susah dibaca** (terlalu banyak indentasi)
  - **Susah di-maintain** (sulit diubah/diperbaiki)
- Biasanya terjadi saat kita punya proses **asinkron berurutan**.

Solusi 1 : Memanggil Callback dengan recursive function

- kita bisa menggunakan recursive function untuk memanggil callback tersebut seperti contoh di bawah ini
- bisa membuat callback tetap, tapi rekursif memanggil dirinya sendiri sampai kondisi selesai.
- Lebih rapi, tapi **masih callback-based**.

```
function execute(nomorAntri){
  periksaAntrianDokter(nomorAntri, function(nomorAntriBaru){
    if (nomorAntriBaru !== 0){
      execute(nomorAntriBaru)
    }
  })
}

execute(7)
```

Solusi 2 : Evolusi Callback: Promises dan Async/Await

Karena masalah "Callback Hell", JavaScript modern memperkenalkan cara yang lebih baik untuk menangani operasi asinkronus:

1. **Promises**: Sebuah objek yang merepresentasikan hasil dari operasi asinkronus di masa depan (bisa berhasil atau gagal). Promises memungkinkan kita untuk merantai operasi asinkronus dengan cara yang lebih rapi menggunakan .then() dan .catch().
2. **Async/Await**: **Lebih modern dan rapi**. Sebuah "syntactic sugar" di atas Promises. Ini memungkinkan kita menulis kode asinkronus seolah-olah kode tersebut sinkronus, membuatnya jauh lebih mudah dibaca dan dipahami.

Meskipun ada Promises dan Async/Await, memahami *callback* tetap **sangat penting**
karena merupakan konsep dasar dari pemrograman asinkronus di JavaScript
dan masih banyak digunakan di berbagai *library* dan *framework*.

***"Syntactic sugar"** (secara harfiah "gula sintaksis") adalah istilah dalam dunia pemrograman untuk menyebut sintaksis (tata cara penulisan kode) yang dirancang agar lebih mudah dibaca, ditulis, dan dipahami oleh manusia, tanpa menambahkan fungsionalitas baru pada bahasa pemrograman tersebut.

Promise

Apa itu Promise di JavaScript?

- Promise adalah **objek** yang mewakili **hasil** dari operasi **asinkron**:
  - **Sekarang** → kita belum punya hasilnya.
  - **Nanti** → hasilnya akan datang (sukses atau gagal).
- Sederhananya, Promise itu **janji**:
- "Saya akan mengembalikan hasilnya nanti, tunggu saja."

More info : https://id.javascript.info/promise-basics

Tiga status Promise

- **Pending** → masih menunggu.
- **Fulfilled** → sukses, janji ditepati.
- **Rejected** → gagal, janji tidak ditepati.

resolve() dan reject()

```
new Promise((resolve, reject) => { ... })
```

Di sini:

- **resolve(value)** → dipanggil jika operasi sukses, dan value akan diteruskan ke .then().
- **reject(reason)** → dipanggil jika operasi gagal, dan reason akan diteruskan ke .catch().

Contoh Promise

Misal kita periksa apakah nomor antrian < 10,
kalau iya kita panggil pasien,
kalau tidak kita tolak.

Alur kode:

1. panggilPasien(7) mengembalikan Promise.
2. setTimeout menunggu 1 detik.
3. Karena 7 < 10, maka resolve(...) dipanggil
4. Pesan sukses diteruskan ke .then(...).
5. Jika nomor ≥ 10, maka reject(...) dipanggil, dan pesan error masuk ke .catch(...)

```
function panggilPasien(nomor) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (nomor < 10) {
        resolve(`Pasien nomor ${nomor}, silakan masuk`);
      } else {
        reject(`Pasien nomor ${nomor} tidak ada di daftar antrian`);
      }
    }, 1000);
  });
}

panggilPasien(7)
  .then((pesan) => {
    console.log("SUCCESS:", pesan);
  })
  .catch((error) => {
    console.error("ERROR:", error);
  });
```

Promise

Sesuai dengan namanya, Promise berarti janji. Seperti janji yang biasanya memakan waktu dan janji bisa ditepati (resolve) atau diingkari (reject). contoh

1

```
var isMomHappy = false;

// Promise
var willIGetNewPhone = new Promise(
    function (resolve, reject) {
        if (isMomHappy) {
            var phone = {
                brand: 'Samsung',
                color: 'black'
            };
            resolve(phone); // fulfilled atau janji dipenuhi
        } else {
            var reason = new Error('mom is not happy');
            reject(reason); // reject (ingkar)
        }

    }
);
```

Menjalankan Promise

cara menggunakan Promise pada kode sebelumnya seperti dibawah ini:

2

```
function askMom() {
  willIGetNewPhone
      .then(function (fulfilled) {
          console.log(fulfilled);
      })
      .catch(function (error) {
          console.log(error.message);
      });
}

askMom()
```

Contoh Promise Lainnya

contoh function yang memiliki return promise:

1

```
function periksaDataPasien(nomorIdPasien) {
  var dataPasien = [
    {id: 1, nama: "John", jenisKelamin: "Laki-laki"},
    {id: 2, nama: "Michael", jenisKelamin: "Laki-laki"},
    {id: 3, nama: "Sarah", jenisKelamin: "Perempuan"},
    {id: 4, nama: "Frank", jenisKelamin: "Laki-laki"}
  ]
  return new Promise( function (resolve, reject){
    var pasien = dataPasien.find(x=> x.id === nomorIdPasien)
    if (pasien === undefined){
      reject("data pasien tidak ada")
    }else{
      resolve(pasien)
    }
  })
}
```

Cara menggunakan Promise yang memiliki parameter

cara menggunakan promise tersebut adalah seperti dibawah ini:

2

```
periksaDataPasien(5).then(function(data){
  console.log(data)
}).catch(function(err){
  console.log(err)
})
```

Kapan saat yang tepat pakai Promise ?

1. **Operasi paralel / bersamaan** →
   Misalnya kita mau ambil data dari 3 API berbeda **tanpa harus menunggu urutan**.

```
Promise.all([
  fetch("/api/user"),
  fetch("/api/posts"),
  fetch("/api/comments")
]).then(([user, posts, comments]) => {
  console.log("✅ Semua data sudah diambil");
});
```

2. **Callback singkat** →
   Kalau cuma satu-dua .then(), kode masih rapi, nggak perlu async.
3. **Library atau API pihak ketiga** →
   Kadang ada library yang langsung return Promise, dan dokumentasinya pakai .then().

Async/Await

Apa itu Async/Await?

Async/await adalah fitur yang hadir sejak ES2017.
Fitur ini mempermudah kita dalam menangani proses asynchronous.

- **async/await** adalah cara yang **lebih rapi dan mudah dibaca** untuk menangani Promise.
- await membuat JavaScript **menunggu** Promise selesai, tapi tanpa menghentikan seluruh program.
- Hanya bisa digunakan di dalam fungsi yang diberi keyword **async**.

More info : https://id.javascript.info/async-await

Apa itu Async/Await?

Kalau Promise dengan `.then()` ibarat:

```javascript
panggilPasien()
  .then(...)
  .catch(...);
```

maka dengan **async/await** jadi:

```javascript
await panggilPasien();
```

lebih mirip **alur cerita** biasa.

Apa itu Async/Await?

Kalau Promise dengan `.then()` ibarat:

```javascript
panggilPasien()
  .then(...)
  .catch(...);
```

maka dengan **async/await** jadi:

```javascript
await panggilPasien();
```

lebih mirip **alur cerita** biasa.

Contoh Kode Async /await

```
function panggilPasien(nomor) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (nomor < 10) {
        resolve(`Pasien nomor ${nomor}, silakan masuk`);
      } else {
        reject(`Pasien nomor ${nomor} tidak ada di daftar antrian`);
      }
    }, 1000);
  });
}

async function prosesAntrian() {
  try {
    console.log("Mengecek nomor antrian...");
    const hasil = await panggilPasien(7); // Tunggu sampai Promise selesai
    console.log("SUCCESS:", hasil);
  } catch (error) {
    console.error("ERROR:", error);
  }
}

prosesAntrian();
```

Contoh lain

1

```
function doAsync() {
  return new Promise( function (resolve, reject){
    var check = true
    if (check){
      resolve("berhasil")
    }else{
      reject("gagal")
    }
  })
}
```

Cara menggunakan Async/Await

cara menggunakan promise tersebut dengan async/await seperti dibawah ini:

2

```
async function hello(){
    var result = await doAsync()
    console.log(result)
}

hello()
```

Error Handling

ketika menggunakan promise maka pasangan dari then adalah catch yang di mana catch itu adalah error handling dari promise, tapi bagaimana dengan async/await, async/await menggunakan try dan catch untuk error handlingnya seperti contoh di bawah ini:

2

```
async function hello(){
  try {
    var result = await doAsync()
    console.log(result)
  } catch(err){
    console.log(err)
  }
}

hello()
```

Kapan saat yang tepat pakai Async/await?

1. **Alur butuh urutan langkah**
   Misalnya pasien harus dipanggil **berurutan** dari nomor 1 → 7.

```
async function antrian() {
  for (let i = 1; i <= 7; i++) {
    const hasil = await panggilPasien(i);
    console.log(hasil);
  }
}
```

2. **Kode panjang dan kompleks** →
   Lebih mudah dibaca karena tampil seperti kode sinkron (step by step).

3. **Error handling** lebih rapi dengan try...catch dibanding .catch() berantai.

```
async function ambilData() {
  try {
    const res = await fetch("/api/data");
    const data = await res.json();
    console.log(data);
  } catch (err) {
    console.error("Error:", err);
  }
}
```

Contoh perbandingan callback vs promise vs async/await

```
//callback hell
function panggilPasien(nomor, callback) {
  setTimeout(() => {
    console.log(`Memanggil pasien nomor ${nomor}`);
    callback();
  }, 1000);
}

panggilPasien(1, () => {
  panggilPasien(2, () => {
    panggilPasien(3, () => {
      console.log("Semua pasien sudah dipanggil");
    });
  });
});
```

Contoh perbandingan callback vs promise vs async/await

```
//versi promise
// Kode lebih lurus (tidak terlalu banyak nested).
// Bisa chain .then().
function panggilPasienPromise(nomor) {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log(`Memanggil pasien nomor ${nomor}`);
      resolve();
    }, 1000);
  });
}

panggilPasienPromise(1)
  .then(() => panggilPasienPromise(2))
  .then(() => panggilPasienPromise(3))
  .then(() => {
    console.log("Semua pasien sudah dipanggil");
  });

//   Di sini:
// resolve(value) → dipanggil jika operasi sukses, dan value akan diteruskan ke .then().
// reject(reason) → dipanggil jika operasi gagal, dan reason akan diteruskan ke .catch().
```

```
//kalau pakai looping :
let chain = Promise.resolve(); // mulai dengan promise kosong

for (let i = 1; i <= 3; i++) {
  chain = chain.then(() => panggilPasienPromise(i));
}

chain.then(() => {
  console.log("Semua pasien sudah dipanggil");
});
```

Contoh perbandingan callback vs promise vs async/await

```
//Versi Async/Await (paling bersih)
function panggilPasienPromise(nomor) {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log(`Memanggil pasien nomor ${nomor}`);
      resolve();
    }, 1000);
  });
}

async function jalankanAntrian() {
  await panggilPasienPromise(1);
  await panggilPasienPromise(2);
  await panggilPasienPromise(3);
  console.log("Semua pasien sudah dipanggil");
}

jalankanAntrian();

// Kelebihan:
// Kode terlihat seperti kode sinkron, tapi tetap berjalan asinkron.
// Mudah dibaca dan di-maintain.
// Cocok untuk proses berurutan yang panjang.
```

```
//atau pakai looping
async function jalankanAntrian() {
  for (let i = 1; i <= 3; i++) {
    await panggilPasienPromise(i);
  }
  console.log("Semua pasien sudah dipanggil");
}

jalankanAntrian();
```

Kesimpulan : Kapan Pakai Callback vs Promise vs Async/Await?

- **Callback**
  - Untuk kode **lama/legacy** yang belum pakai Promise.
  - Kalau interaksi sederhana sekali-dua kali (misalnya fs.readFile di Node).
  - Tapi hindari untuk flow yang panjang → rawan **callback hell**.
- **Promise**
  - Kalau perlu chaining yang jelas.
  - Banyak library modern (axios, fetch) sudah default pakai Promise.
  - Cocok kalau butuh Promise.all() atau Promise.race() untuk menjalankan banyak async sekaligus.
- **Async / Await**
  - Best practice sekarang (paling mudah dibaca).
  - Cocok untuk alur panjang atau logic yang harus step-by-step.
  - Disarankan untuk project baru (lebih maintainable).

Tugas Praktek

Soal1

Pada folder **"tugas-asynchronouse-javascript"** kerjakan soal dibawah ini.

1. **(Callback)**

   Kita mempunyai tumpukan buku untuk dibaca. Setiap buku memiliki waktu yang dibutuhkan untuk menghabiskan buku tersebut. Sudah disediakan function readBooks yang menerima tiga parameter: waktu, buku yang dibaca, dan sebuah callback. Salin code berikut ke dalam sebuah file bernama `callback.js` .

```
// di callback.js
function readBooks(time, book, callback ) {
    console.log("saya membaca " + book.name )
    setTimeout(function(){
        let sisaWaktu = 0
        if(time >= book.timeSpent) {
            sisaWaktu = time - book.timeSpent
            console.log("saya sudah membaca " + book.name + ", sisa waktu saya " + sisaWaktu)
            callback(sisaWaktu) //menjalankan function callback
        } else {
            console.log('waktu saya habis')
            callback(time)
        }
    }, book.timeSpent)
}

module.exports = readBooks
```

Soal1

Masih satu folder dengan file `callback.js`, buatlah sebuah file dengan nama `index.js` lalu tuliskan code seperti berikut.

```
// di index.js
var readBooks = require('./callback.js')

var books = [
    {name: 'LOTR', timeSpent: 3000},
    {name: 'Fidas', timeSpent: 2000},
    {name: 'Kalkulus', timeSpent: 4000},
    {name: 'komik', timeSpent: 1000}
]

// Tulis code untuk memanggil function readBooks di sini
```

lanjutkan code pada `index.js` untuk memanggil function readBooks. Buku yang akan dihabiskan adalah buku-buku di dalam array `books`. Pertama function readBooks menerima input waktu yang dimiliki yaitu 10000 ms (10 detik) dan books pada indeks ke-0. Setelah mendapatkan callback sisa waktu yang dikirim lewat callback, sisa waktu tersebut dipakai untuk membaca buku pada indeks ke-1. Begitu seterusnya sampai waktu habis atau semua buku sudah terbaca. Untuk melihat output, jalankan file `index.js` dengan node js :

```
$ node index.js
```

Soal2

2. **(Promise)**

Setelah no.1 berhasil, implementasikan function readBooks yang menggunakan callback di atas namun sekarang menggunakan Promise. Buatlah sebuah file dengan nama `promise.js`. Tulislah sebuah function dengan nama readBooksPromise yang me-return sebuah promise seperti berikut:

```
// di file promise.js
function readBooksPromise (time, book) {
  console.log("saya mulai membaca " + book.name )
  return new Promise( function (resolve, reject){
    setTimeout(function(){
      let sisaWaktu = time - book.timeSpent
      if(sisaWaktu >= 0 ){
          console.log("saya sudah selesai membaca " + book.name + ", sisa waktu saya " + sisaWaktu)
          resolve(sisaWaktu)
      } else {
          console.log("saya sudah tidak punya waktu untuk baca "+ book.name)
          reject(sisaWaktu)
      }
    }, book.timeSpent)
  })
}

module.exports = readBooksPromise
```

Masih di folder yang sama dengan promise.js, buatlah sebuah file dengan nama `index2.js`. Tuliskan code sebagai berikut

```
var readBooksPromise = require('./promise.js')

var books = [
    {name: 'LOTR', timeSpent: 3000},
    {name: 'Fidas', timeSpent: 2000},
    {name: 'Kalkulus', timeSpent: 4000}
]

// Lanjutkan code untuk menjalankan function readBooksPromise
```

Lakukan hal yang sama dengan soal no.1, habiskan waktu selama 10000 ms (10 detik) untuk membaca semua buku dalam array books.!

Soal3

3. **(Promise dan async/await)**

Buatlah sebuah file dengan nama `promise2.js`. Tulislah sebuah function dengan nama filterBookPromise yang me-return sebuah promise seperti berikut:

```
function filterBooksPromise(colorful, amountOfPage){
  return new Promise(function(resolve, reject){
    var books=[
        {name: "shinchan", totalPage: 50, isColorful: true},
        {name: "Kalkulus", totalPage: 250, isColorful: false},
        {name: "doraemon", totalPage: 50, isColorful: true},
        {name: "algoritma", totalPage: 250, isColorful: false},
    ]
    if (amountOfPage >= 40) {
        resolve(books.filter(x=> x.totalPage == amountOfPage && x.isColorful == colorful));
      } else {
        var reason= new Error("Maaf buku di bawah 40 halaman tidak tersedia")
        reject(reason);
      }
  });
}

module.exports = filterBooksPromise
```

Masih di folder yang sama dengan promise2.js, buatlah sebuah file dengan nama `index3.js`. Tuliskan code sebagai berikut:

```
var filterBooksPromise = require('./promise2.js')

// Lanjutkan code untuk menjalankan function filterBookPromise
```

gunakan promise dengan kondisi seperti di bawah ini:

- bukunya berwarna dan jumlah halamannya 50
- bukunya tidak berwarna dan jumlah halamannya 250 (**gunakan async/await untuk kondisi ini**)
- bukunya berwarna dan jumlah halamannya 30 (**gunakan async/await untuk kondisi ini**)

Soal3

berikut ini contoh output soal 3:

```
[
  { name: 'shinchan', totalPage: 50, isColorful: true },
  { name: 'doraemon', totalPage: 50, isColorful: true }
]
[
  { name: 'Kalkulus', totalPage: 250, isColorful: false },
  { name: 'algoritma', totalPage: 250, isColorful: false }
]
Maaf buku di bawah 40 halaman tidak tersedia
```

Soal4

4. **(Promise dan async/await)**

Buatlah sebuah file dengan nama `promise3.js`. Tulislah sebuah function dengan nama filterBookPromise yang me-return sebuah promise seperti berikut:

```
function filterCarsPromise(color, year){
  return new Promise(function(resolve, reject){
    var cars=[
        {brand: "toyota", name: "avanza", year: 2019, color:"black"},
        {brand: "daihatsu", name: "xenia", year: 2017, color: "silver"},
        {brand: "lamborghini", name: "gallardo", year: 2018, color: "grey"},
        {brand: "honda", name: "brio", year: 2019, color: "black"},
        {brand: "toyota", name: "agya", year: 2020, color: "black"},
        {brand: "honda", name: "jazz", year: 2018, color: "grey"},
        {brand: "suzuki", name: "ertiga", year: 2017, color: "silver"}
    ]

    var filteredCars = cars.filter(x=> x.color === color && x.year === year)
    if (filteredCars.length > 0) {
        resolve(filteredCars);
      } else {
        var reason= new Error("Maaf Data tidak di temukan")
        reject(reason);
      }
  });

}

module.exports = filterCarsPromise
```

Masih di folder yang sama dengan promise3.js, buatlah sebuah file dengan nama `index4.js`. Tuliskan code sebagai berikut:

```
var filterCarsPromise = require('./promise3.js')

// Lanjutkan code untuk menjalankan function filterCars
```

Soal4

gunakan promise dengan kondisi seperti di bawah ini:

- mobil berwarna hitam tahun 2019
- mobil berwarna silver tahun 2017
- mobil berwarna abu-abu tahun 2019 (**gunakan async/await untuk kondisi ini**)
- mobil berwarna abu-abu tahun 2018 (**gunakan async/await untuk kondisi ini**)
- mobil berwarna hitam tahun 2020 (**gunakan async/await untuk kondisi ini**)

berikut ini contoh output soal 4:

```
[ { brand: 'toyota', name: 'avanza', year: 2019, color: 'black' } ]
[
  { brand: 'daihatsu', name: 'xenia', year: 2017, color: 'silver' },
  { brand: 'suzuki', name: 'ertiga', year: 2017, color: 'silver' }
]
Maaf Data tidak di temukan
[
  { brand: 'lamborghini', name: 'gallardo', year: 2018, color: 'grey' },
  { brand: 'honda', name: 'jazz', year: 2018, color: 'grey' }
]
[ { brand: 'toyota', name: 'agya', year: 2020, color: 'black' } ]
```

KEMNAKER

Kesimpulan

- Asynchronous

Terima Kasih