
Asynchronous Programming JavaScript

A yellow square containing the letters 'JS' in a bold, dark grey font, representing the JavaScript logo.

JS

Synchronous Programming

- Operasi yang dilakukan secara berurutan dari atas ke bawah
- JavaScript secara default bersifat **Synchronous** dan **Single Thread**



```
console.log( "Satu" );  
console.log( "Dua" );  
console.log( "Tiga" );
```

```
console.log("Edo Membuka Bowser Google Chrome.");

console.log(`
Edo download file selama 1 jam.
  Downloading 1 Hour ...
  Download Complete.
`);

console.log("Edo membuka Youtube.");
```

- Jika proses **synchronous** terjadi di browser
- Kita perlu menunggu 1 jam agar bisa membuka youtube
- Karena kita perlu menunggu download selesai
- Pada kasus ini, operasi **synchronous** tidak efisien
- Sebagai gantinya, dapat dilakukan operasi **asynchronous**
- Tujuannya agar kita dapat melakukan berbagai proses secara paralel (bersamaan)

Callback Function

Callback

- Fungsi yang dikirim sebagai parameter ke Fungsi lain
- Callback function berjalan setelah Main Function berjalan

Example:

- Fungsi B dikirim sebagai parameter ke Fungsi A
- Fungsi A menjalankan Fungsi B di dalam Fungsi A

Referensi: [JavaScript Tutorial - Callbacks](#)

```

/**
 * Fungsi untuk format nama menjadi uppercase.
 *
 * @param {string} name
 */
function formatName(name) {
  const result = name.toUpperCase();
  return result;
}

/**
 * Fungsi untuk mendapatkan nama.
 *
 * @param {string} name
 * @param {function} callback
 */
function getName(name, callFormatName) {
  const result = callFormatName(name);
  console.log(result);
}

getName("Edo", formatName);

```

Bingung melihat **callback function**?

Soalnya di JavaScript, function bisa diperlakukan sebagai apa saja.

Function bisa diperlakukan (First-Class Function):

- Disimpan ke variabel
- Dikirim ke parameter
- return function

```
const formatName = (name) => name.toUpperCase();  
  
const getName = (name, callFormatName) => console.log(callFormatName(name));  
  
getName("Edo", formatName);
```

Jangan bingung, ini cuma **arrow function** dengan single statement kok.
Karena single statement, si function otomatis return (implisit return).

Asynchronous Programming JavaScript

Asynchronous Programming

Operasi dilakukan tanpa menunggu operasi lain selesai.

Sehingga operasi seakan-akan dilakukan secara bersamaan (paralel).

Kita perlu belajar **Asynchronous**, karena banyak operasi di JavaScript bersifat **Asynchronous**.

Operasi **Asynchronous** di JavaScript:

- Read File
- Access Database
- Network: Fetch

Kita dapat melakukan simulasi Asynchronous menggunakan fungsi **setTimeout**.

Referensi: [Devsaurus - Panduan Lengkap Asynchronous](#)

```
console.log("Satu");  
  
setTimeout(function() {  
    console.log("Dua");  
}, 3000);  
  
console.log("Tiga");
```

```
console.log("Edo Membuka Bowser Google Chrome.");

/**
 * setTimeout salah satu operasi Asynchronous.
 * fungsi setTimeout tidak mencegah operasi selanjutnya.
 * callback otomatis dijalankan setelah 5 detik.
 */
setTimeout(() => {
  console.log("Downloading 1 Hour...");
  console.log("Download complete");
}, 5000);

console.log("Edo membuka Youtube.");
```

Problem in **Asynchronous**

Problem

Asynchronous merupakan solusi terbaik untuk menjalankan operasi secara paralel.

Namun hal ini bisa memunculkan masalah jika antar operasi saling berkaitan dan membutuhkan.

Apalagi setiap operasi memiliki waktu yang berbeda-beda.

Maka kita perlu meng-handle proses **Asynchronous**.

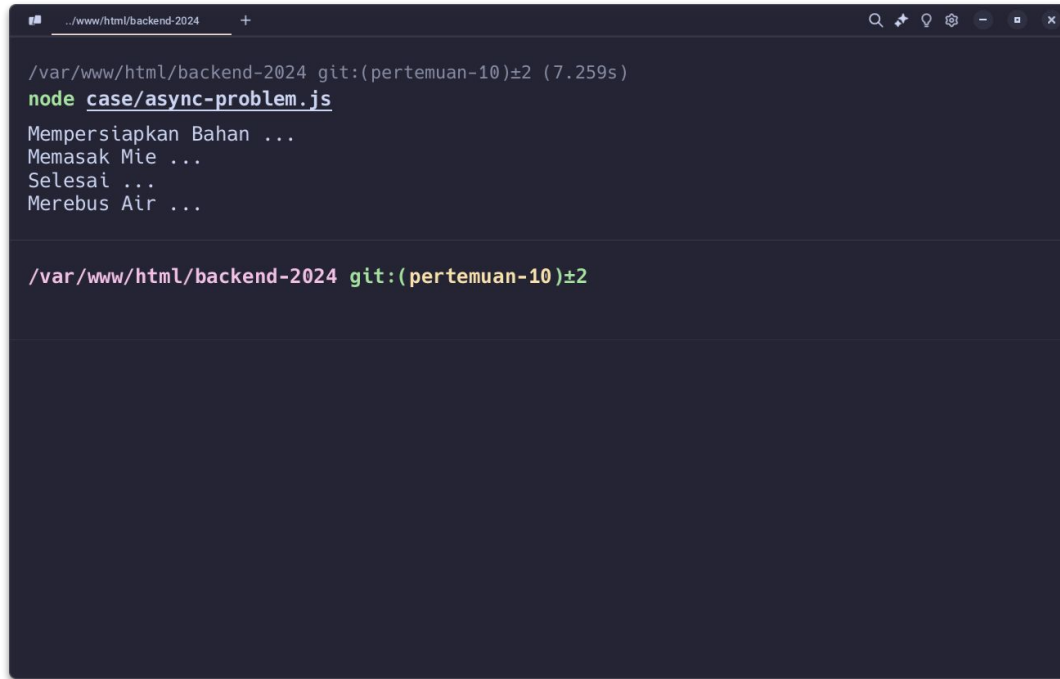


Analogi Proses Memasak Mie (Asynchronous):

- Persiapan: Waktu 3 Menit
- Rebus Air: Waktu 7 Menit
- Masak: Waktu 5 Menit

```
const persiapan = () => {  
  setTimeout(function () {  
    console.log("Mempersiapkan Bahan ...");  
  }, 3000);  
};  
  
const rebusAir = () => {  
  setTimeout(function () {  
    console.log("Merebus Air ...");  
  }, 7000);  
};  
  
const masak = () => {  
  setTimeout(function () {  
    console.log("Memasak Mie ...");  
    console.log("Selesai");  
  }, 5000);  
};
```

```
const main = () => {  
  persiapan();  
  rebusAir();  
  masak();  
};  
  
main();
```


A terminal window with a dark background and light-colored text. The window title is ".../www/html/backend-2024". The prompt is "/var/www/html/backend-2024 git:(pertemuan-10)±2 (7.259s)". The command "node case/async-problem.js" has been executed, resulting in four lines of output: "Mempersiapkan Bahan ...", "Memasak Mie ...", "Selesai ...", and "Merebus Air ...". The prompt is now "/var/www/html/backend-2024 git:(pertemuan-10)±2".

```
.../www/html/backend-2024 +
/var/www/html/backend-2024 git:(pertemuan-10)±2 (7.259s)
node case/async-problem.js
Mempersiapkan Bahan ...
Memasak Mie ...
Selesai ...
Merebus Air ...

/var/www/html/backend-2024 git:(pertemuan-10)±2
```

- Muncul problem karena operasi dilakukan secara asynchronous
- Sedangkan operasi yang dilakukan saling berkaitan
- Ada banyak operasi asynchronous di JavaScript (Read File, Access DB, network, dll)
- Kita perlu meng-handle operasi asynchronous yang saling berkaitan

Solution in **Asynchronous**

Solution


Solusi yang dapat digunakan:

- Callback - Old Way
- Promises - Modern Way (ES6)
- Async Await - Modern Way (ES Next)

Solution with **Callback**

```
const persiapan = () => {  
  console.log("Mempersiapkan Bahan ...");  
};  
  
const rebusAir = () => {  
  console.log("Merebus Air ...");  
};  
  
const masak = () => {  
  console.log("Memasak Mie ...");  
  console.log("Selesai");  
};
```

```
const main = () => {  
  setTimeout(() => {  
    persiapan();  
  
    setTimeout(() => {  
      rebusAir();  
  
      setTimeout(() => {  
        masak();  
      }, 5000);  
    }, 7000);  
  }, 3000);  
};  
  
main();
```



```

const main = () => {
  setTimeout(() => {
    callback();
    setTimeout(() => {
      callback();
      setTimeout(() => {
        callback();
        setTimeout(() => {
          callback();
          setTimeout(() => {
            callback();
            setTimeout(() => {
              callback();
              setTimeout(() => {
                callback();
                setTimeout(() => {
                  callback();
                }, 4000);
              }, 5000);
            }, 2000);
          }, 3000);
        }, 1000);
      }, 4000);
    }, 2000);
  }, 3000);
};

main();

```

Solved by **Callback**.

But wait, WTH is this?

How if there are 10 asynchronous operations?

CALLBACK HELL.

Promise

Modern Way to Handle Asynchronous

Promise

Object yang mengembalikan nilai di masa mendatang (**future** not now).

Karena mengembalikan nilai di masa mendatang, **promise** sangat cocok untuk menangani proses **asynchronous**.

Promise menjadi salah satu solusi terbaik sebagai pengganti callback (callback hell).

Referensi: [JavaScript Tutorial - Promises](#)

Promise

Pembuatan promise terbagi menjadi 2 bagian:

- **Producing:** Membuat atau menghasilkan Promise
- **Consuming:** Menggunakan atau mengkonsumsi Promise

Promise memiliki 3 state:

- **Pending:** Ketika promise berjalan
- **Fulfilled:** Ketika promise berhasil (resolve)
- **Rejected:** Ketika promise gagal (reject)

Referensi: [JavaScript Tutorial - Promises](#)

```
/**
 * Membuat fungsi download
 * @returns {object} Promise
 */
const download = () => {
  /**
   * Promise dibuat menggunakan class Promise.
   * Promise menerima callback function/executor.
   * Executor menerima 2 params: resolve dan reject.
   * resolve dipanggil jika proses berhasil.
   * reject dipanggil jika proses gagal.
   */
  return new Promise((resolve, reject) => {
    const status = true;

    setTimeout(() => {
      if (status) {
        resolve("Download Selesai");
      } else {
        reject("Download Gagal");
      }
    }, 5000);
  });
};

console.log(download());
```

Promise - Consuming

Sebelumnya kita telah melakukan tahapan **Producing Promise**.

Selanjutnya kita perlu melakukan **Consuming Promise** yaitu Menggunakan atau mengkonsumsi hasil promise.

Consuming promise:

- **.then**: menangkap promise ketika status berhasil (resolve)
- **.catch**: menangkap promise ketika status gagal (reject)

```
/**  
 * .then menangkap hasil resolve  
 * .catch menangkap hasil reject  
 */  
download()  
  .then((res) => {  
    console.log(res);  
  })  
  .catch((err) => {  
    console.log(err);  
  });
```

Solution with **Promise**

```
const persiapan = () => {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Menyiapkan Bahan ...");
    }, 3000);
  });
};

const rebusAir = () => {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Merebus Air ...");
    }, 7000);
  });
};

const masak = () => {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Masak Mie ...");
    }, 5000);
  });
};
```

Producing Promise

```
const main = () => {
  persiapan()
    .then((res) => {
      console.log(res);
      return rebusAir();
    })
    .then((res) => {
      console.log(res);
      return masak();
    })
    .then((res) => {
      console.log(res);
    });
};

main();
```

Consuming Promise

Async Await

Async Await

ES2017 memperkenalkan `async await` yang dibangun di atas promise.

Kita dapat menulis kode `asynchronous` menggunakan Async Await.

Memudahkan penulisan kode asynchronous agar terlihat seperti synchronous dan readable.

`Async Await` hanya [syntactic sugar](#) untuk promise.

Note: untuk memahami async await harus memahami promise terlebih dahulu.

Referensi: [JavaScript Tutorial - Async Await](#)

Solution with **Async Await**

```
/**
 * async digunakan untuk memberitahu ada proses asynchronous
 * await digunakan untuk menunggu promise selesai.
 * await hanya bisa digunakan di dalam function.
 */
async function main() {
  console.log(await persiapan());
  console.log(await rebusAir());
  console.log(await masak());
}

main();
```

```
const main = async () => {
  console.log(await persiapan());
  console.log(await rebusAir());
  console.log(await masak());
};

main();
```

Callback

VS

Promise

VS

Async Await

```
const main = () => {
  setTimeout(() => {
    persiapan();

    setTimeout(() => {
      rebusAir();

      setTimeout(() => {
        masak();
      }, 5000);
    }, 7000);
  }, 3000);
};

main();
```

Callback

```
const main = () => {
  persiapan()
  .then((res) => {
    console.log(res);
    return rebusAir();
  })
  .then((res) => {
    console.log(res);
    return masak();
  })
  .then((res) => {
    console.log(res);
  });
};

main();
```

Promise

```
const main = async () => {
  console.log(await persiapan());
  console.log(await rebusAir());
  console.log(await masak());
};

main();
```

Async Await

Which is better and readable?

Kesimpulan

- JavaScript dapat menjalankan operasi **Asynchronous**.
- Operasi **Asynchronous** dapat di-handle menggunakan callback dan **promise**.
- Callback memiliki masalah: Callback Hell.
- **Promise** merupakan solusi terbaik untuk menuliskan kode asynchronous.
- **Async Await** bukan sebagai pengganti promise.
- Async Await bertujuan untuk menuliskan kode yang lebih mudah dibaca dan terlihat seperti synchronous.

Best Practice:

- Gunakan ~~callback~~ **Promise** untuk menulis (**Producing**) kode asynchronous
- Gunakan **Async Await** untuk menggunakan (**Consuming**) promise

Promise + Async Await
is Awesome

Task

Task

Tujuan:

- Membuat fitur download menggunakan Promise
- Refactor penggunaan callback ke **Promise** atau **Async Await**
- Refactor legacy code ke ES6

Ketentuan:

- Upload Task ke Github
- Kirim link repository ke Elena: [Link](#)
- Boilerplate task: [Link](#)

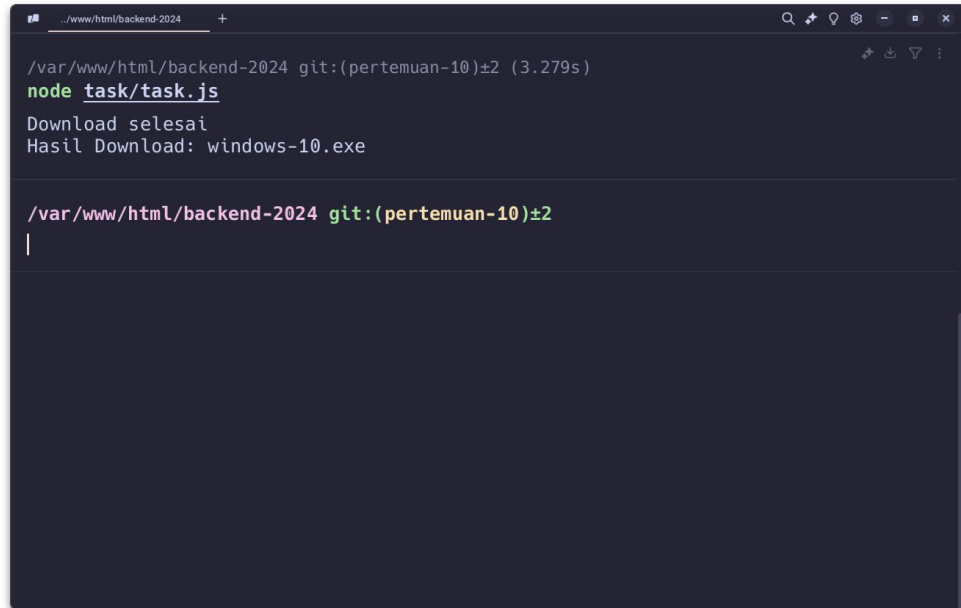

```

/**
 * Fungsi untuk menampilkan hasil download
 * @param {string} result - Nama file yang didownload
 */
function showDownload(result) {
  console.log("Download selesai");
  console.log("Hasil Download: " + result);
}

/**
 * Fungsi untuk download file
 * @param {function} callback - Function callback show
 */
function download(callShowDownload) {
  setTimeout(function () {
    const result = "windows-10.exe";
    callShowDownload(result);
  }, 3000);
}

download(showDownload);

```



```

./www/html/backend-2024 +
/var/www/html/backend-2024 git:(pertemuan-10)±2 (3.279s)
node task/task.js
Download selesai
Hasil Download: windows-10.exe

/var/www/html/backend-2024 git:(pertemuan-10)±2
|

```

Referensi

- JavaScript Info: [The Modern JavaScript Tutorial](#)
- JavaScript Tutorial: [JavaScript Tutorial](#)
- ES6 Tutorial: [ES6 Tutorial](#)

Next

Silahkan pelajari terlebih dahulu materi tentang **Object Oriented Programming** di JavaScript

- Devsaurus: [JavaScript Class](#) (Tulisan - Indonesia)
- JavaScript Tutorial: [ES6 Class](#) (Tulisan - English)
- JavaScript Info: [Classes](#) (Tulisan - English)

Rekomendasi:

- Jika sudah punya basic OOP, silahkan belajar di Devsaurus
- Jika ingin mendalami OOP lebih detail silahkan ke JavaScript Tutorial atau JavaScript Info