

# Web Dev Bootcamp 2024/25

JavaScript - Lab 10

14.12.2024

Sponsored by:

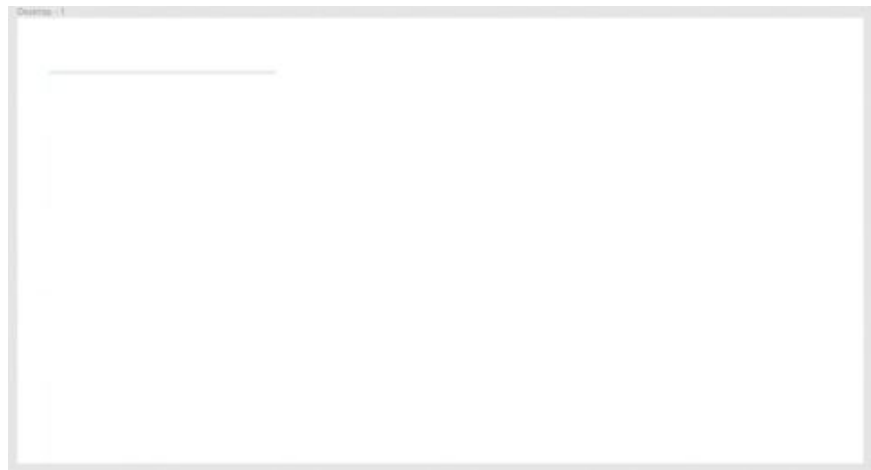


# Agenda

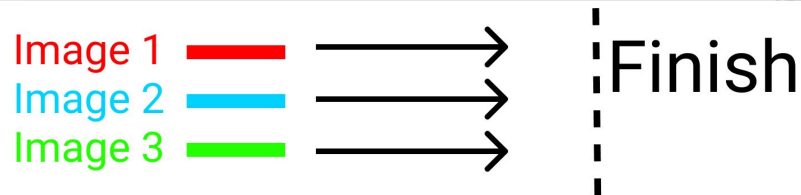
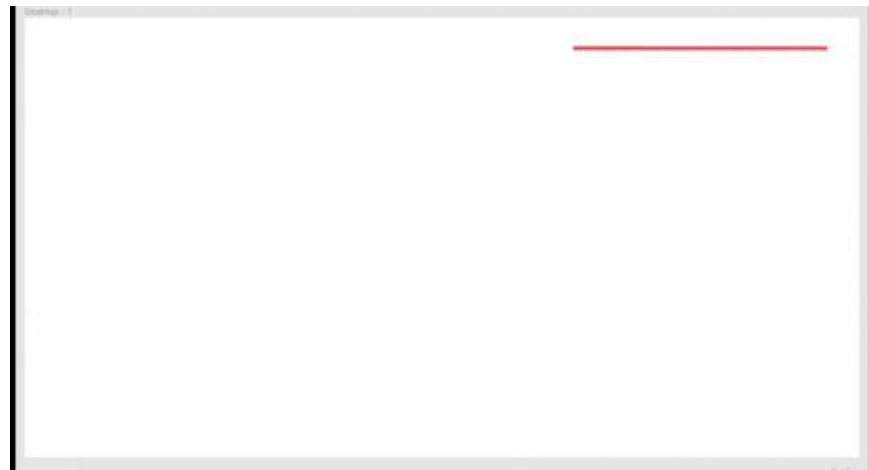
- ~~1. Diskusi tentang tugas minggu lalu~~
2. Synchronous vs Asynchronous Programming
3. Callback & Timeout
4. Promise
5. Await & Fetch
6. To-do for next session
7. Organisasi dan feedback

# Synchronous vs Asynchronous

## Synchronous



## Asynchronous

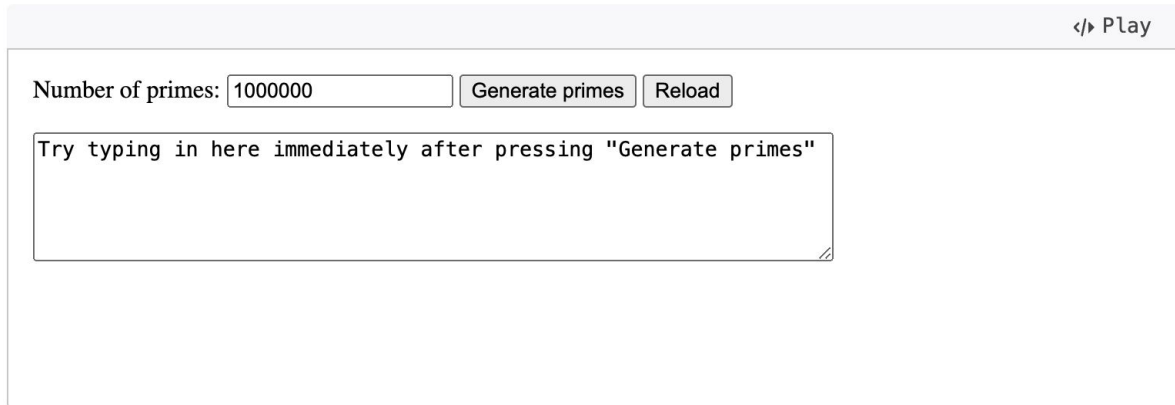


# Synchronous vs Asynchronous

Web applications kebanyakan menggunakan asynchronous programming pattern. Apalagi kalau banyak user interactions di app nya.

Kenapa jarang ada yang menggunakan synchronous...?

Coba kita cek [di sini](#).



The screenshot shows a web application interface with a light gray header bar containing a code icon and the text "Play". Below the header, there is a form with the label "Number of primes:" followed by a text input field containing "1000000". To the right of the input field are two buttons: "Generate primes" and "Reload". Below these elements is a large text area with a placeholder text "Try typing in here immediately after pressing 'Generate primes'". The text area has a small icon in the bottom right corner.

Source: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing>

# Asynchronous programming di JS



- Functional programming: ada granularity untuk functions.
  - Functions dibuat menjadi berbagai function yang lebih kecil.
  - Functions ini bisa dijalankan secara paralel.
- Kita bisa menggunakan teknik-teknik Functional Programming di JS, dan mengimplementasikannya dengan e.g. `promise` dan `await`.
- Kita juga bisa menggunakan *function chaining*, e.g. menggunakan `then()` dan `finally()`.
- A great blogpost on this topic:  
<https://medium.com/@aidobreen/js-promises-async-await-and-functional-programming-f2e5fa66b4ef>

# Callback

## Callback illustrated

```
Function One (){  
    // Do something  
}
```

```
Function Two (call_One){  
    // Do something else  
    call_One()  
}
```

`Two(One);`  code is being executed

Source: <https://www.freecodecamp.org/news/javascript-async-await-tutorial-learn-callbacks-promises-async-await-by-making-icecream/>

**Function yang memanggil function lain.**

Atau,

**suatu function yang menjadi suatu argument untuk function yang lain.**

Tapi Callback ini bagusnya dipakai hanya untuk operations yang simple.

Jangan menggunakan Callback untuk operations yang kompleks/rumit.

Kenapa? Karena kita akan mendapatkan ***Callback hell...***

# Timeout

```
setTimeout(code)
setTimeout(code, delay)

setTimeout(functionRef)
setTimeout(functionRef, delay)
setTimeout(functionRef, delay, param1)
setTimeout(functionRef, delay, param1, param2)
setTimeout(functionRef, delay, param1, param2, /* ..., */ paramN)
```

Source: <https://developer.mozilla.org/en-US/docs/Web/API/Window/setTimeout>

`setTimeout()` adalah native JS API method yang bisa kita gunakan sebagai timer.

Kita kasih arguments:

- Function yang akan di call/execute.
- Timer atau count-down sebelum akhirnya function itu akan dicall.

# Callback (and Timeout) Hell

```
getData(function(a) {  
  getMoreData(a, function(b) {  
    getEvenMoreData(b, function(c) {  
      getEvenEvenMoreData(c, function(d) {  
        getFinalData(d, function(finalData) {  
          console.log(finalData);  
        });  
      });  
    });  
  });  
});
```

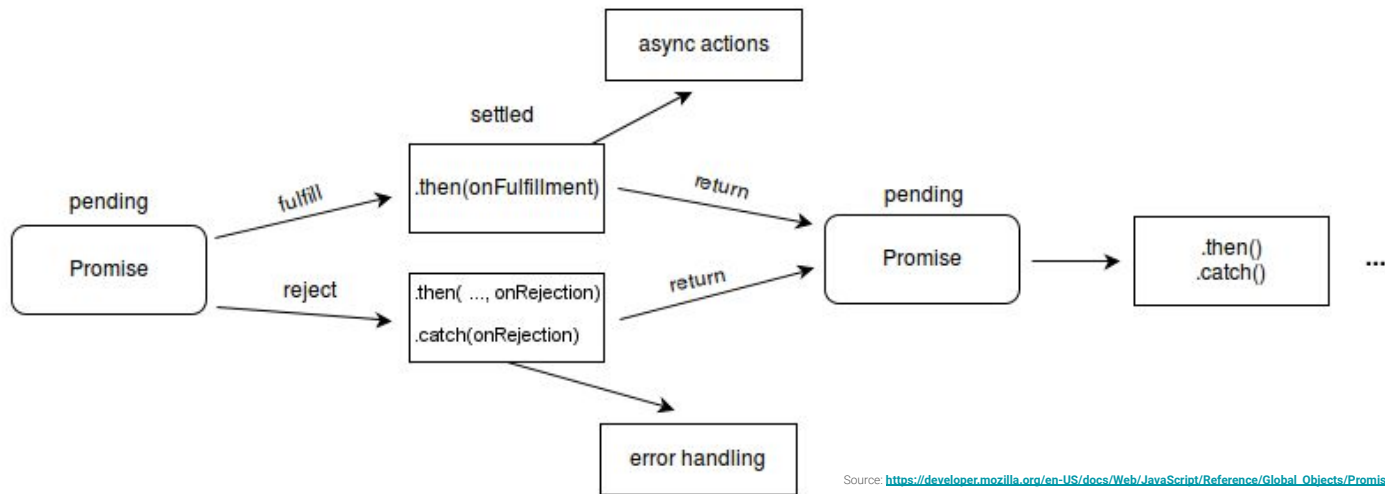


- Susah dibaca/dimengerti → *Spaghetti Code*.
- Error prone.
- Susah untuk debugging.
- Kalau hanya rely on `setTimeout()`, kalau timer yang diberikan kurang cukup, function/callback berikutnya bisa error.
- More on this:
  - <https://www.geeksforgeeks.org/what-to-understand-callback-and-callback-hell-in-javascript/>
  - [https://medium.com/@raihaan\\_tazdid/callback-hell-in-javascript-all-you-need-to-know-296f7f5d3c1](https://medium.com/@raihaan_tazdid/callback-hell-in-javascript-all-you-need-to-know-296f7f5d3c1)



# Promise

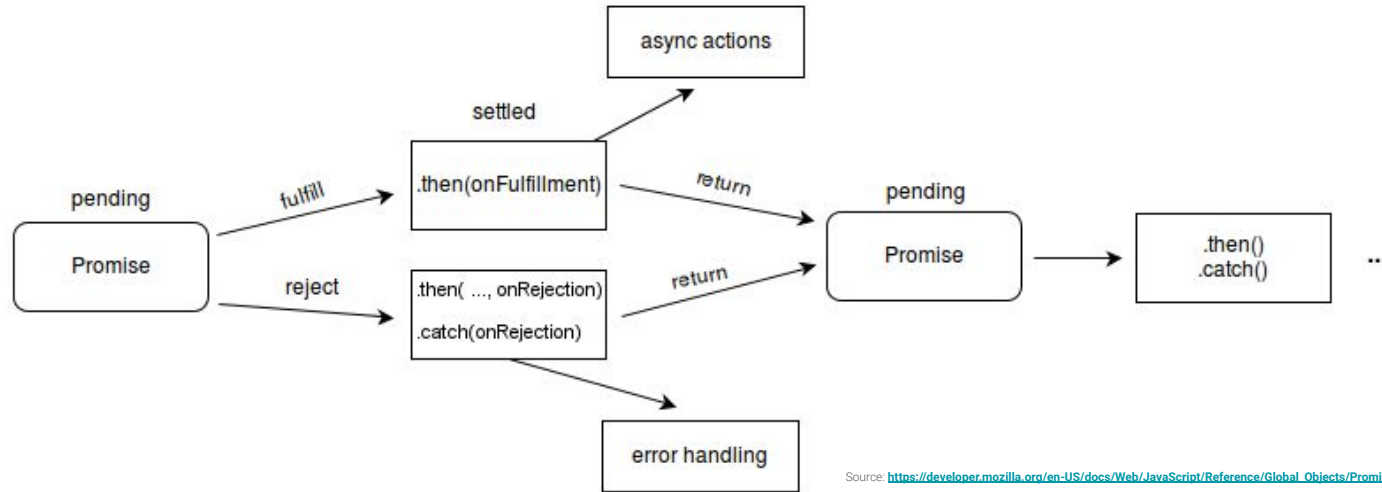
Promises ini bisa mengatasi masalah-masalah yang diberikan Callback dan Timeout. Kok bisa?



Dengan Promise, kita bisa mendeklarasikan sequences dari functions-functions yang kita mau.

- Cocok untuk operations yang kompleks/rumit.
- Code lebih mudah dibaca dan dimengerti.
- Debugging juga lebih gampang.

# Promise



Promise memiliki 4 *states*:

1. **Pending**
2. **Fulfilled**
3. **Rejected**
4. Resolved/Settled

Kita bisa membuat sequence dari Promises dengan menggunakan instance methods:

- [then\(\)](#)
- [catch\(\)](#)
- [finally\(\)](#)

More on this:

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

# Promise

1. Initialize/declare  
suatu Promise
2. Memanggil Promise  
yang tadi sudah kita  
declare

```
// Initialize a promise
const myPromise = new Promise(function(resolve, reject) => {})
```

```
myPromise
  .then((result) => {
    console.log(result);
  })
  .catch((error) => {
    console.log(error);
  })
  .finally(() => {
    //code here will be executed regardless of the status
    //of a promise (fulfilled or rejected)
  });
```

10 mins. break :)  
sampai 19:50

# Async/Await

## Promise

```
function kitchen(){  
  return new Promise ((resolve, reject)=>{  
    if(true){  
      resolve("promise is fulfilled")  
    }  
  
    else{  
      reject("error caught here")  
    }  
  })  
}
```

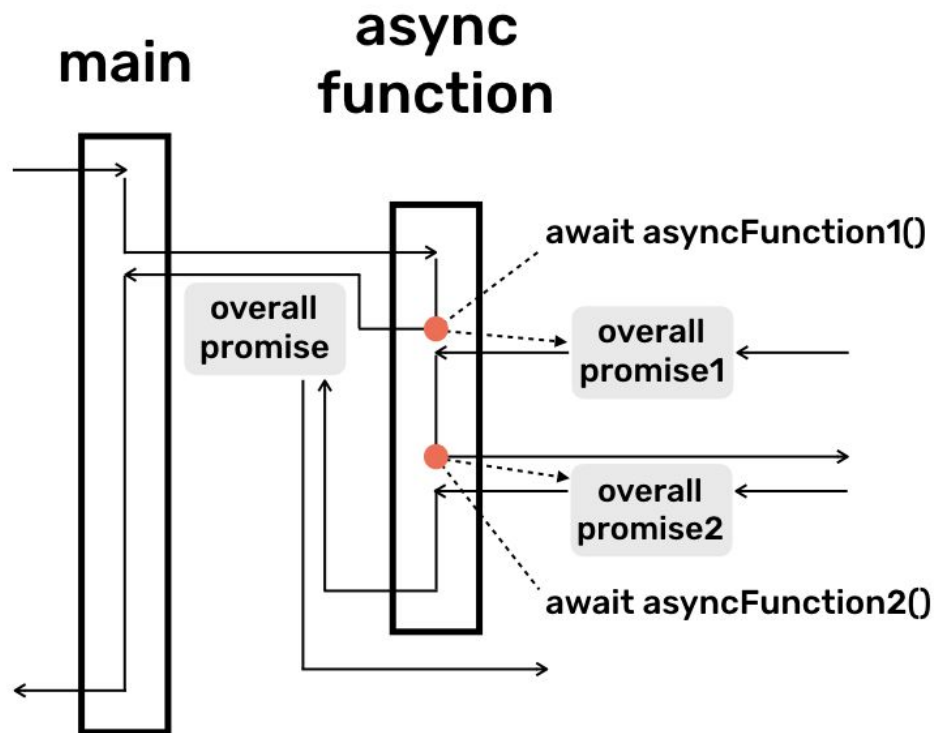
```
kitchen() // run the code  
.then() // next step  
.then() // next step  
.catch() // error caught here  
.finally() // end of the promise [optional]
```

## Await

```
// 🪄 Magical keyword  
async function kitchen(){  
  
  try{  
    // Let's create a fake problem  
    await abc;  
  }  
  
  catch(error){  
    console.log("abc does not exist", error)  
  }  
  
  finally{  
    console.log("Runs code anyways")  
  }  
}
```

```
kitchen() // run the code
```

# Async/Await



Source: <https://dmitrykandalov.com/async-await>

`await` dipakai untuk menunggu hasil dari sebuah Promise.

Biasanya, `await` expression ini digunakan dalam suatu **async function**.

Kalau ada `await` dalam suatu **async** function, maka code lines yang lain (yang ada di dalam **async** function itu) akan di pause sampai akhirnya Promise yang ditunggu oleh `await` ini selesai (bisa *fulfilled* atau bisa juga *rejected*).

More on this:

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await>
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async\\_function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function)

# Try dan Catch

Bagaimana caranya kita meng-handle errors dalam suatu **async** function?  
Kita bisa menggunakan **try()** dan **catch()**.

```
async function runProcess() {  
  try {  
    const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');  
    const json = await response.json();  
    console.log(json);  
  } catch (error) {  
    console.log(error);  
  }  
}
```

```
runProcess();
```

# Fetch

```
async function getData() {  
  const response = await fetch('https://jsonplaceholder.typicode.com/posts/1');  
  const data = await response.json();  
  console.log(data);  
}  
  
getData();
```

Source: <https://www.freecodecamp.org/news/asynchronous-programming-in-javascript/>

**fetch()** adalah suatu method dari native [JS Fetch API](#), yang gunanya adalah untuk memanggil dan memproses HTTP requests.

More on this:

- [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)
- [https://developer.mozilla.org/en-US/docs/Web/API/Fetch\\_API/Using\\_Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch)
- <https://www.freecodecamp.org/news/asynchronous-programming-in-javascript/>



# Async/Await & Fetch

- Cocok untuk digunakan di operations yang memerlukan kompleks Promise chaining.
- Code jadi lebih rapi dan gampang dimengerti.
- Ada dedicated methods untuk handle errors.
- Debugging juga lebih mudah.
- Bisa langsung call Fetch API dalam async functions.

More on this:

- <https://www.freecodecamp.org/news/javascript-async-await/>
- [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Promises#async\\_and\\_await](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Promises#async_and_await)
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise/try](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/try)

# To-do untuk next session

The screenshot displays a web browser with the URL `pokemon-search-app.freecodecamp.rocks`. On the left, a mobile app interface is shown with a search bar labeled "Search for Pokémon Name or ID:" and a purple "Search" button. Below the search bar is a table with two columns: "Base" and "Stats". The table has rows for "HP:", "Attack:", "Defense:", "Sp. Attack:", "Sp. Defense:", and "Speed:", each with an empty input field.

On the right, the browser's developer tools are open to the "Sources" tab, showing the `script.js` file. The code is as follows:

```
15
16 const getPokemon = async () => {
17   try {
18     const pokemonNameOrId = searchInput.value.toLowerCase();
19     const response = await fetch(
20       `https://pokeapi-proxy.freecodecamp.rocks/api/pokemon/${pokemonNameOrId}`
21     );
22     const data = await response.json();
23
24     // Set Pokémon info
25     pokemonName.textContent = `${data.name.toUpperCase()}`;
26     pokemonID.textContent = `#${data.id}`;
27     weight.textContent = `Weight: ${data.weight}`;
28     height.textContent = `Height: ${data.height}`;
29     spriteContainer.innerHTML = `
30       
31     `;
32
33     // Set stats
34     hp.textContent = data.stats[0].base_stat;
35     attack.textContent = data.stats[1].base_stat;
36     defense.textContent = data.stats[2].base_stat;
37     specialAttack.textContent = data.stats[3].base_stat;
38     specialDefense.textContent = data.stats[4].base_stat;
39     speed.textContent = data.stats[5].base_stat;
40
41     // Set types
42     types.innerHTML = data.types
43       .map(obj => `<span class="type ${obj.type.name}">${obj.type.name}</span>`)
44       .join('');
45   } catch (err) {
46     resetDisplay();
47     alert('Pokémon not found');
48     console.log('Pokémon not found: ${err}');
49   }
50 };
51
```

The status bar at the bottom indicates "Line 24, Column 24" and "Coverage: n/a".

# To-do untuk next session

11	<b>JavaScript Algo. &amp; Data Structures</b>	<ul style="list-style-type: none"> <li>Asynchronous Programming</li> <li>Fetch and Promises</li> </ul>	Sabtu, <b>14 Des. 2024</b> 19:00 - 21:00 WIB	<b>Pre-read:</b> <ol style="list-style-type: none"> <li><a href="#">Async vs Synchronous JavaScript</a></li> <li><a href="#">Async JavaScript Tutorial</a></li> </ol> <b>Homework:</b> <ol style="list-style-type: none"> <li><a href="#">freeCodeCamp: Build a Pokemon Search App</a></li> </ol>
13	<b>Build Your Portfolio</b>	<ul style="list-style-type: none"> <li>Show your current portfolio on GitHub Pages</li> <li>Update your CV</li> </ul>	Sabtu, <b>21 Des. 2024</b> 19:00 - 21:00 WIB	<b>Pre-read:</b> -  <b>Homework:</b> <ol style="list-style-type: none"> <li>Update your GitHub Pages and CV</li> </ol>

# Organisasi dan feedback

- Feedback: <https://forms.gle/KYTwqJrWR7nYCtZh8>
- Minggu depan:
  - Ada yang mau showcase portfolio? :)
  - Sari code reviews
  - Ngobrol santai