Async Await

in JavaScript

2025

REPOST ❤️

What is Async/Await?

async and **await** are modern JavaScript features used to handle asynchronous operations in a more readable and maintainable way compared to traditional callbacks or Promises. They allow you to write asynchronous code that looks and behaves like synchronous code, making it easier to understand.

Key Concepts

Async/Await is a way to make your code wait for an asynchronous operation to finish before moving on to the next step.

- **Async**: A function declared with the async keyword always returns a Promise. It allows the use of await inside the function.

Await: The await keyword pauses the execution of the async function until the Promise is resolved or rejected. It can only be used inside an async function.

Key Concepts

Async:

- The async keyword is used to declare a function that always returns a Promise.
- Inside an async function, you can use await to pause execution until a Promise is settled.

```
async function fetchData() {
  return "Data fetched!";
}

fetchData().then(console.log); // Output: Data fetched!
```

Key Concepts

Await

- The await keyword is used inside async functions only.
- It waits for the Promise to resolve before moving on to the next line.

```
async function example() {
  let result = await Promise.resolve("Done!");
  console.log(result); // Output: Done!
}
```

Key Concepts

Error Handling with try...catch

- You can handle errors from awaited Promises using try...catch.

```
async function getData() {
  try {
    let data = await
fetch("https://api.example.com");
    let json = await data.json();
    console.log(json);
  } catch (error) {
    console.error("Error:", error);
  }
}
```

Key Concepts

Async Functions Return Promises

- **Even if you return a value (not a Promise), it will be wrapped in a Promise automatically.**

```
async function hello() {
  return "Hi!";
}

hello().then(console.log); // Output: Hi!
```

Key Concepts

Parallel Execution Using Promise.all()

- **Use Promise.all() inside async functions to run multiple Promises concurrently.**

```
async function loadAll() {
  let [user, profile] = await Promise.all([
    fetch("/user"),
    fetch("/profile")
  ]);
}
```

Key Concepts

Blocking vs Non-Blocking

- await pauses execution of the current async function but does not block the event loop.
    - So JavaScript can still run other code in the background.

Avoid Top-Level await (unless using ES modules)

- In regular scripts, await can't be used outside an async function.

Mastering **async and await** is essential for writing clean, readable, and efficient asynchronous code in JavaScript. By understanding how these keywords work, you can eliminate callback hell, improve error handling, and structure your code to behave more predictably.
Start small — refactor a function, handle an API call, or debug an existing async issue. Over time, you'll find that asynchronous programming becomes second nature. Whether you're building front-end UIs or handling back-end APIs, async/await will become one of your most powerful tools in JavaScript.

M Moaz Ahmad

Was this helpful?
If this post helped you understand async/await better, give it a like, share it with your network, and follow me for more developer-friendly content like this!
Let’s keep growing together. 🚀