Appendix C: Understanding async/await in JavaScript

`async` and `await` are keywords introduced in ES2017 that allow you to write asynchronous code in a way that looks synchronous — improving readability and maintainability.

Behind the scenes, `async`/`await` works on top of Promises and the microtask queue.

🔷 What is `async`?

A function declared with the `async` keyword always returns a Promise.

```
async function greet() {
  return "Hello!";
}

// Equivalent to:
function greet() {
  return Promise.resolve("Hello!");
}
```

🔷 What is `await`?

The `await` keyword pauses the execution of an `async` function until the Promise it waits for is resolved (or rejected).

```
async function main() {
  const result = await Promise.resolve("Done");
  console.log(result);
}
```

- If the Promise resolves: `await` returns the value.
- If it rejects: `await` throws an error (which should be caught with `try/catch`).

🔄 How It Works Internally

When an `async` function encounters `await`, it:

1. **Pauses execution** of that function.
2. Returns control to the call stack.
3. Places a continuation (callback) into the **microtask queue**.
4. Resumes after the current execution stack and all other microtasks complete.

This is why `async/await` does not block the main thread.

📄 Example: Without Arrow Functions

```
console.log("Start");

async function run() {
  console.log("Inside async function");

  await new Promise(function (resolve) {
    setTimeout(function () {
      console.log("Timeout inside Promise");
      resolve();
    }, 0);
  });

  console.log("After await");
}

run();

console.log("End");
```

Output:

```
Start
Inside async function
End
Timeout inside Promise
After await
```

⚠ Error Handling with `try` / `catch`

You should always wrap `await` calls in a `try/catch` block if the Promise might reject:

```
async function loadData() {
  try {
    const response = await fetch("/data.json");
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error("Failed to load", error);
  }
}
```

🔁 Chaining with async/await

This:

```
async function run() {
  const user = await getUser();
  const profile = await getProfile(user);
  return profile;
}
```

Is much cleaner than:

```
getUser()
  .then(user => getProfile(user))
  .then(profile => ...)
  .catch(err => ...);
```

✅ Summary

Keyword	What it does
`async`	Declares a function that returns a Promise
`await`	Pauses inside an async function and waits for a Promise to resolve

- Makes async logic easier to read and reason about
- Works best with `try/catch` blocks for error handling
- Executes **non-blocking**: doesn’t halt the main thread
