JavaScript `async` and `await`

Writing asynchronous code using `.then()` and `.catch()` works well, but as your code grows, it can still feel a bit hard to follow.

JavaScript gives us a cleaner way to work with Promises:
`async` and `await`

What is `async` ?

If you put the keyword `async` before a function, it **automatically returns a Promise**.

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

greet().then(function (message) {
  console.log(message); // "Hello"
});
```

Even though we just returned a string, `greet()` becomes a Promise.

What is `await` ?

The `await` keyword is used **inside an `async` function**. It tells JavaScript to **wait for the Promise to resolve**, then continue.

Basic Example

Let's simulate a delay using `setTimeout` wrapped in a Promise:

```
function waitTwoSeconds() {
  return new Promise(function (resolve) {
    setTimeout(function () {
      resolve("Waited for 2 seconds");
    }, 2000);
  });
}

async function runTask() {
  console.log("Start");

  const result = await waitTwoSeconds();
  console.log(result);

  console.log("End");
}

runTask();
```

Output:

```
Start
Waited for 2 seconds
End
```

Why Use `async` and `await`?

Let's compare the same logic using `.then()`:

```
waitTwoSeconds().then(function (result) {
  console.log(result);
});
```

It works, but once you have **multiple async operations**, the `.then()` style gets harder to follow.

With `await`, your code looks more like regular, synchronous code — even though it's asynchronous.

Handling Errors with `try...catch`

If a Promise rejects, you can catch the error using `try...catch`.

```
function fakeTask(fail) {
  return new Promise(function (resolve, reject) {
    setTimeout(function () {
      if (fail) {
        reject("Something went wrong");
      } else {
        resolve("Task completed");
      }
    }, 1000);
  });
}

async function run() {
  try {
    const result = await fakeTask(false);
    console.log(result);
  } catch (error) {
    console.log("Caught error:", error);
  }
}
```

```
run();
```

Summary

- `async` makes a function return a Promise.
- `await` pauses inside the function until the Promise is done.
- You can use `try...catch` to handle errors just like synchronous code.
- `async` / `await` makes asynchronous code easier to **read** and **write**.