💡 Day 4: Async/Await & Fetch API – Technical Deep Dive

🔁 1. Promises – Recap & Foundation

In JavaScript, asynchronous code allows us to run time-consuming operations (like network requests) without blocking the main thread.

A **Promise** represents the eventual result (success or failure) of an asynchronous operation.

js
CopyEdit

```
const p = new Promise((resolve, reject) => {
  setTimeout(() => resolve("Success"), 2000);
});

p.then((data) => console.log(data)).catch((err) => console.log(err));
```

- `resolve(value)` → marks the promise as successful.
- `reject(error)` → marks the promise as failed.
- `.then()` → used when promise is successful.
- `.catch()` → used to handle errors.

🔄 2. What is `async` and `await` in JavaScript?

👉 `async`:

Used to declare a function that will always return a Promise.

👉 `await`:

Can only be used inside an `async` function. It pauses execution until the awaited promise settles.

Example:

js
CopyEdit

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

example().then(console.log); // Output: Hello Async!
```

With await:

js
CopyEdit

```
async function fetchData() {
  const promise = new Promise((resolve) => {
    setTimeout(() => resolve("Data loaded"), 2000);
  });

  const result = await promise;
  console.log(result); // Output: Data loaded
}
```

✅ `await` makes your code look **synchronous**, even though it's async under the hood.

🌐 3. Real-world Example: fetch() + async/await

The `fetch()` function is used to make HTTP requests. It returns a promise.

Basic Usage:

js
CopyEdit

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(response => response.json())
  .then(data => console.log(data));
```

With async/await:

js

CopyEdit

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

getUsers();
```

> ⚠️ `fetch()` will not throw an error for HTTP errors (like 404), only for network errors. You must check `response.ok`.

🛡️ 4. Error Handling – `try...catch` with async/await

Use `try...catch` to gracefully handle errors in async code:

js
CopyEdit

```
async function getPosts() {
  try {
    const res = await
fetch('https://jsonplaceholder.typicode.com/posts/1');
    const data = await res.json();
    console.log(data);
  } catch (err) {
    console.error("Something went wrong:", err);
  }
}
```

⏳ 5. Custom Delay Function with Promises

You can create delays using a Promise and `setTimeout`:

js
CopyEdit

```
function delay(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function delayedHello() {
  console.log("Waiting...");
  await delay(3000);
  console.log("Hello after 3 seconds!");
}
```

🧠 TASKS FOR PRACTICE (MANDATORY ✅)

Task 1:

Make a function `getTodos()` using async/await to fetch data from:

arduino
CopyEdit
`https://jsonplaceholder.typicode.com/todos`

- Show only the first 5 todos (use `.slice(0, 5)`).
- Wrap it inside a `try...catch`.

Task 2:

Create a `delay(ms)` function (as above).
Call it with 2 seconds and print “Waited 2 seconds” after waiting.

Task 3 (Bonus 💥):

Make a function that fetches user data from this API:

arduino

CopyEdit
`https://jsonplaceholder.typicode.com/users/3`

- Log the user's name and email.
- Handle errors if the request fails.

📝 Summary Table

Concept	Description
`Promise`	Object representing future completion/failure of async operation
`async`	Declares a function returning a promise
`await`	Waits until the promise resolves
`fetch()`	Makes HTTP requests, returns a Promise
`try...catch`	Handles errors during async operations