Session 3

3. JS Objects & Advanced Concepts

3.1 Callback Hell

When callbacks are **nested inside other callbacks**, it creates messy code:

```
doTask1(() => {
  doTask2(() => {
    doTask3(() => {
      doTask4(() => {
        console.log("Done!");
      });
    });
  });
});
```

This is called **callback hell** 😵

It's hard to read, debug, or scale.

→ Solution: Use **Promises** or **async/await**

3.2 Try/Catch Blocks

When something breaks in your code, `try/catch` helps you stop the entire app from crashing.

Let's break it down slowly:

- `try` contains the code that might throw an error.
- `catch` handles the error if one happens.

```
try {
  let result = unknownVar; // this will throw an error because unknownVar is not defined
} catch (e) {
  console.log("Error caught:", e.message);
}
```

Output:

```
Error caught: unknownVar is not defined
```

If you didn't use `try/catch`, your program would crash entirely. But with this, it handles the problem and keeps running.

3.3 Promises (Async Basics)

Let's say you ask your friend for a pizza or burger. You don't get it instantly. You wait. That's a **promise**.

In JS, a Promise is used when you're doing something that takes time, like fetching data from a server.

Here's a basic example:

```
let p = new Promise((resolve, reject) => {
  resolve("Data received"); // instantly resolves
});

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

Output:

```
Data received
```

What's going on:

- A new Promise is created.
- It resolves with the value `"Data received"`
- `.then()` is used to get the result once it's ready.

If something had gone wrong, you'd use `.catch()` to handle it (like try/catch).

Async/await is a newer, simpler way to work with Promises.

Instead of chaining `.then()` after `.then()`, you just pause and wait using `await`.

But let's clarify a common myth first:

> ❗ You cannot use await inside a regular function. It must be used inside a function declared with async.

Also:

- An `async` function **always returns a Promise**.
- You don’t *have* to use `await` inside an `async` function, but you usually do.

Let’s break this into smaller examples:

Example 1 – Waiting for a delayed message

```
async function sayHi() {
  console.log("Waiting...");
  await new Promise(resolve => setTimeout(resolve, 1000));
  console.log("Hi after 1 second");
}

sayHi();
```

Output:

```
Waiting...
Hi after 1 second
```

Here’s what happens:

- JS logs "Waiting..."
- Then `await` pauses the function for 1 second
- Then logs the second message
