📝 Day 7: Callbacks and Promises

✅ Today you'll learn how to handle **asynchronous code** in Node.js using **Callbacks** and **Promises**.

✅ What is a Callback?

- A function **passed as a parameter** to another function.
- It is **called later** when work is done.

🧪 Example:

```
js
CopyEdit
function greet(name, callback) {
  console.log('Hello', name);
  callback();
}

greet('Supriya', () => console.log('Goodbye!'));
```

✅ Key point:

A function becomes a callback when you pass it as a parameter and call it inside another function.

✅ Are callbacks always async?

❌ No!

✅ Synchronous callback:

```
js
CopyEdit
[1, 2, 3].forEach(num => console.log(num));
```

✅ Asynchronous callback:

```
js
CopyEdit
setTimeout(() => console.log('Later!'), 1000);
```

✅ What is "callback hell"?

✅ Example:

```
js
CopyEdit
getUser(id, (user) => {
  getOrders(user.id, (orders) => {
    getItems(orders[0], (items) => {
      console.log(items);
    });
  });
});
```

😱 Hard to read and maintain.

✅ Promises to the rescue!

✅ A **Promise** is an object representing **a future value**.

✅ States:

- Pending
- Fulfilled
- Rejected

✅ Basic Promise Example:

```
js
CopyEdit
const myPromise = new Promise((resolve, reject) => {
  resolve('It worked!');
});

myPromise.then(console.log);
```

✅ Output:

```
nginx
CopyEdit
It worked!
```

✅ Using setTimeout with a Promise

```
js
CopyEdit
const waitTwoSeconds = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('Finished waiting 2 seconds!');
  }, 2000);
});

waitTwoSeconds.then(console.log);
```

✅ Using resolve and reject

```
js
CopyEdit
function doubleAsync(num) {
  return new Promise((resolve, reject) => {
    if (num) {
      setTimeout(() => resolve(num * 2), 1000);
    } else {
      reject('Please provide a number!');
    }
  });
}

doubleAsync(5)
  .then(console.log)
  .catch(console.error);
```

✅ Handles **success** with .then().
✅ Handles **error** with .catch().

✅ Async/Await — Syntactic Sugar for Promises

✅ async = returns a Promise automatically.
✅ await = pause until Promise resolves.

✅ Simple Example:

```
js
CopyEdit
async function sayHello() {
  return 'Hello!';
}
```

```
sayHello().then(console.log);
```

✅ Output:

```
CopyEdit
Hello!
```

✅ Using await:

```
js
CopyEdit
async function run() {
  console.log('Start');
  await wait(2000);
  console.log('After 2 seconds');
}

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

run();
```

✅ Output:

```
bash
CopyEdit
Start
...wait 2 seconds...
After 2 seconds
```

✅ Using await with try/catch:

```
js
CopyEdit
async function doubleAndLog(num) {
  try {
    const result = await doubleAsync(num);
    console.log(result);
  } catch (err) {
    console.error(err);
  }
}

doubleAndLog(5);
```

✅ Error automatically goes to `catch`.

✅ **Key differences:**

Feature	Promise	Async/Await
Returns	Promise	Promise (automatically)
Handle result	.then()	await
Handle errors	.catch()	try/catch
Code style	Chained	Linear, easy to read

✅ Why use async/await?

- Avoid .then() chains.
- Looks synchronous.
- Easier error handling.

✅ Mini Task:

Write an async function called `tripleAsync` that:

- Takes a number
- Returns a Promise that resolves with number * 3 after 1 second
- Use `await` to get the result and log it

✅ Next Steps?

- Build more routes.
- Use Promises everywhere.
- Refactor to async/await.
- Connect to databases!