Here’s a complete guide on **callbacks**, **Promises**, and **async/await** in Node.js — three core ways to handle **asynchronous operations**.

**🔄 1. Callbacks in Asynchronous Operations**

**✅ What is a Callback?**

A **callback** is a function passed as an argument to another function, which is called **after** an asynchronous operation completes.

**📄 Example: Using a Callback with fs.readFile**

const fs = require('fs');

fs.readFile('example.txt', 'utf8', (err, data) => {

if (err) {

console.error('Error:', err);

return;

}

console.log('File content:', data);

});

⚠️ Problem: Nesting multiple callbacks leads to **callback hell**, which makes code harder to read and maintain.

**🧩 2. Introduction to Promises**

A **Promise** represents the **eventual result** of an asynchronous operation. It has three states:

* **Pending**
* **Resolved (fulfilled)**
* **Rejected (error)**

**📦 Example: Basic Promise**

const myPromise = new Promise((resolve, reject) => {

const success = true;

if (success) {

resolve('It worked!');

} else {

reject('Something went wrong.');

}

});

myPromise

.then(result => console.log(result))

.catch(error => console.error(error));

**📁 Example: Reading a File Using Promises**

const fs = require('fs').promises;

fs.readFile('example.txt', 'utf8')

.then(data => console.log('File:', data))

.catch(err => console.error('Error:', err));

✅ Promises avoid nesting and support cleaner chaining with .then() and .catch().

**🚀 3. Async/Await Basics**

**Async/await** is syntactic sugar over Promises, making asynchronous code look **synchronous**.

**🔹 Rules:**

* Use async before a function to allow the use of await inside.
* Use await before a Promise to **pause** execution until it resolves.

**📘 Example: Reading a File with Async/Await**

const fs = require('fs').promises;

async function readFileAsync() {

try {

const data = await fs.readFile('example.txt', 'utf8');

console.log('Async/Await:', data);

} catch (err) {

console.error('Error:', err);

}

}

readFileAsync();

**⚖️ Summary Comparison**

| **Method** | **Pros** | **Cons** |
| --- | --- | --- |
| **Callbacks** | Simple, low-level | Hard to manage in nested flows |
| **Promises** | Better flow control, chaining | Verbose with many .then() calls |
| **Async/Await** | Clean, readable syntax | Must handle errors with try/catch |

**🧠 Quick Recap**

* Use **callbacks** when dealing with legacy Node.js code.
* Use **Promises** for chaining asynchronous steps.
* Use **async/await** for most modern Node.js code — it's clean and readable.

Let me know if you want a comparison with real-world API calls, or a small app demo using async/await and Promises.