Callback vs Promise:

**1️⃣ Callbacks**

**🔹 What are Callbacks?**

A **callback** is a function passed as an argument to another function. It is executed after the operation is completed.

const fs = require("fs");

fs.readFile("hello.txt", "utf8", (err, data) => {

if (err) {

console.log("Error reading file:", err);

return;

}

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

});

**Problems with Callbacks**

1. **Callback Hell (Pyramid of Doom)** – When multiple nested callbacks are needed, code becomes **hard to read**.
2. **Error Handling is Tricky** – Errors must be handled manually in each callback.
3. **Inversion of Control** – The flow of execution is **controlled by the callback**, not the function itself.

**Promises**

**🔹 What are Promises?**

A **Promise** represents the **eventual completion** (or failure) of an asynchronous operation.  
Instead of using callbacks, it allows chaining .then() and handling errors with .catch().

**✅ Example: Using fs.promises.readFile()**

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

fs.readFile("hello.txt", "utf8")

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

.catch((err) => console.log("Error reading file:", err));

Example: Using async/await

const readFileAsync = async () => {

try {

const data = await fs.readFile("hello.txt", "utf8");

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

} catch (err) {

console.log("Error reading file:", err);

}

};

readFileAsync();

**Comparison Table: Callbacks vs Promises**

| **Feature** | **Callbacks** | **Promises** |
| --- | --- | --- |
| **Syntax** | Nested functions | Chained .then(), catch() |
| **Readability** | Hard to read (callback hell) | More readable |
| **Error Handling** | Handled inside each callback | .catch() for all errors |
| **Chaining** | Difficult | Easy using .then() |
| **Inversion of Control** | Yes (callback controls execution) | No (code controls execution) |
| **Debugging** | Harder to trace | Easier, stack trace available |

**Callback Hell vs Promise Chaining**

**🚨 Callback Hell Example**

fs.readFile("file1.txt", "utf8", (err, data1) => {

if (err) return console.error(err);

fs.readFile("file2.txt", "utf8", (err, data2) => {

if (err) return console.error(err);

fs.readFile("file3.txt", "utf8", (err, data3) => {

if (err) return console.error(err);

console.log("All files read!");

});

});

});

Promise Chaining Solution

fs.readFile("file1.txt", "utf8")

.then((data1) => fs.readFile("file2.txt", "utf8"))

.then((data2) => fs.readFile("file3.txt", "utf8"))

.then((data3) => console.log("All files read!"))

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

**When to Use What?**

| **Use Case** | **Callbacks** | **Promises** |
| --- | --- | --- |
| Simple async tasks | ✅ Yes | ✅ Yes |
| Multiple async operations (chained) | ❌ No | ✅ Yes |
| Error handling | ❌ No | ✅ Yes |
| Readability & maintainability | ❌ No | ✅ Yes |