JavaScript Callbacks and Promises (Detailed Notes)

1. What is a Callback?

- A callback is a function passed as an argument to another function, which is executed later.
- Used in asynchronous programming (e.g., reading files, API requests).
- Problem: Too many nested callbacks lead to **Callback Hell**.

Example of Callback:

```
function greet(name, callback) {
console.log("Hello " + name);
callback();
}

greet("Soman", () => {
console.log("Goodbye!");
});
```

2. Callback Hell

- Happens when callbacks are nested inside callbacks.
- Code becomes unreadable and hard to maintain.
- Also called the **Pyramid of Doom**.

Example:

```
getUser(1, function(user) {
getPosts(user.id, function(posts) {
getComments(posts[0].id, function(comments) {
console.log(comments);
});
});
});
```

3. What is a Promise?

- A **Promise** is an object representing the eventual completion or failure of an asynchronous operation.

- States:
1. Pending → initial state
2. Fulfilled → operation completed successfully (resolve)
3. Rejected → operation failed (reject)

Basic Example:

```
let promise = new Promise((resolve, reject) => {
let success = true;
if (success) resolve("Task successful!");
else reject("Task failed!");
});

promise
.then(result => console.log(result))
.catch(error => console.log(error))
.finally(() => console.log("Done"));
```

4. Why Use Promises?

- Solves the problem of Callback Hell.
- Makes code cleaner, easier to read, and maintain.
- Better error handling with `.catch()`.

5. Promise Methods

- **.then()** → Runs when resolved.
- **.catch()** → Runs when rejected.
- **.finally()** → Runs always, no matter resolved/rejected.

6. Promise Chaining

```
new Promise((resolve, reject) => {
resolve(2);
})
.then(num => num * 2)
.then(num => num * 2)
```

```
.then(num => console.log(num)); // Output: 8
```

7. Async/Await (Sugar for Promises)

```
async function fetchData() {
try {
let data = await fetch("https://jsonplaceholder.typicode.com/posts/1");
let json = await data.json();
console.log(json);
} catch (error) {
console.error(error);
}
}
fetchData();
```

8. Real World Examples

1. **API Calls** (fetch data from server)
2. **Image loading**
3. **Database queries in Node.js**
4. **User authentication (login/signup)**

9. Summary

- Callbacks → Functions passed to other functions, but can get messy.
- Callback Hell → Nested callbacks, unreadable code.
- Promises → Cleaner async handling with resolve/reject.
- Async/Await → Most modern and clean approach.