JavaScript Callback Functions - Detailed Notes

What is a Callback Function?

A callback function is a function passed as an argument to another function and is executed after some operation has been completed.

Why Use Callbacks?

JavaScript is asynchronous by nature. Callbacks allow non-blocking behavior and enable custom behavior in functions.

Syntax

```
function mainFunction(callback) {
  callback();
}

function greet() {
  console.log("Hello!");
}

mainFunction(greet);
```

Types of Callbacks

- Synchronous: Executes immediately.
- Asynchronous: Executes later (e.g., setTimeout, API calls).

Real-World Examples

1. Event Listeners:
```
document.getElementById("btn").addEventListener("click", () => console.log("Clicked!"));
```

2. Array Iteration:
```
[1,2,3].forEach(num => console.log(num * 2));
```

Custom Callback

```
function processUserInput(callback) {
  const name = "Charlie";
  callback(name);
}

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

processUserInput(greet);
```

Error Handling Callback

JavaScript Callback Functions - Detailed Notes

```
function login(user, pass, callback) {
  if (user === "admin" && pass === "123") callback(null, "Success");
  else callback("Error", null);
}
```

Callback Hell Example

```
step1(() => {
  step2(() => {
    step3(() => {
      console.log("Done");
    });
  });
});
```

Mini Project: Custom Filter

```
function customFilter(arr, callback) {
  return arr.filter(callback);
}

const result = customFilter([1,5,10], n => n > 5);
```

Pros and Cons

Pros:
- Enables async behavior
- Improves flexibility

Cons:
- Can lead to callback hell
- Hard to debug

Callback vs Promise vs Async/Await

Callback: Manual error handling
Promise: .then().catch()
Async/Await: Cleaner with try/catch