Callback Function:

In JavaScript, a callback function is a function that is passed as an argument to another function and is executed after the completion of a certain task or event. Callbacks are commonly used in asynchronous programming to handle operations that take time to complete, such as fetching data from a server, reading files, or handling user input.

Basic Example:

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

function sayGoodbye() {
    console.log('Goodbye!');
}

greet('John', sayGoodbye); // Output: Hello, John followed by
Goodbye!
```

In this example:

- **The greet function takes two parameters: name and callback.**
- **It logs a greeting message to the console and then calls the callback function.**
- **The sayGoodbye function is passed as a callback to greet, so it gets executed after the greeting message is logged.**

Asynchronous Example:

```javascript
//Asynchronous Example:
function fetchData(callback) {
    setTimeout(() => {
        const data = 'Data from server';
        callback(data);
    }, 2000); // Simulating delay of 2 seconds
}

function processData(data) {
    console.log('Received data:', data);
}

fetchData(processData); // Output after 2 seconds: Received data: Data from server
```

In this example:

- **The fetchData function simulates fetching data from a server with a delay of 2 seconds using setTimeout.**
- **After the data is fetched, the callback function (processData) is called with the fetched data as an argument.**
- **The processData function receives the data and logs it to the console.**

Anonymous Function as Callback:

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

greet('Alice', function() {
    console.log('Nice to meet you!');
});
```

In this example, instead of defining a separate named function as the callback, we define an anonymous function inline and pass it directly as the second argument to greet.

Handling Errors with Callbacks:

```
function fetchData(callback, errorCallback) {
  setTimeout(() => {
    const error = false; // Simulating success
    if (error) {
      errorCallback("Error occurred");
    } else {
      const data = "Data from server";
      callback(data);
    }
  }, 2000); // Simulating delay of 2 seconds
}

function processData(data) {
  console.log("Received data:", data);
}

function handleFetchError(error) {
  console.error("Error:", error);
}

fetchData(processData, handleFetchError);
```

In this example:

- **The fetchData function simulates fetching data from a server.**
- **If an error occurs during the fetch operation, the errorCallback function (handleFetchError) is called.**
- **If the fetch operation is successful, the callback function (processData) is called with the fetched data.**

Callback functions are fundamental in JavaScript, especially for handling asynchronous operations, event handling, and ensuring non-blocking behavior in applications. They provide a way to execute code after certain tasks or events have completed.