Exercise 1: Create a function processData that takes two parameters: a string and a callback function. Your task is to write a callback that converts the string to uppercase and then call it within processData.

Requirements: ● Define a function toUpperCase that will serve as a callback.

● Pass a string and toUpperCase to processData and log the output.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>

function toUpperCase(str) {
    return str.toUpperCase();
}

function processData(inputString, callback) {

    const result = callback(inputString);
    return result;
}


const myString = "hello, world!";
const uppercasedString = processData(myString, toUpperCase);


console.log(uppercasedString);

    </script>
</body>
</html>
```

Exercise 2: Write a function forEachElement that accepts an array and a callback. This function should apply the callback to each element of the array.

Requirements: ● Pass an anonymous function as the callback that multiplies each element by 2 and logs the result with the index.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
function forEachElement(array, callback) {
    for (let i = 0; i < array.length; i++) {
        callback(array[i], i);
    }
}


const numbers = [1, 2, 3, 4, 5];

forEachElement(numbers, function(element, index) {
    const result = element * 2;
    console.log("Index: " + index + ", Result: " + result);
});
    </script>


</body>
</html>
```

Exercise 3: Simulate a network request by creating a function fetchData that takes a URL and a callback as parameters. Use setTimeout to simulate a delay and then call the callback with a string representing a response.

Requirements: ● After a delay, log the “response” to the console. C

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>

function fetchData(url, callback) {
    console.log("Fetching data from:", url);

    setTimeout(function() {
        const response = "Response from " + url;
        callback(response);
    }, 2000);
}
const apiUrl = "https://api.example.com/data";
```

```
fetchData(apiUrl, function(response) {
    console.log("Received:", response); // Log the response to the console
});
```

Exercise 4: Modify fetchData from Exercise 3 to include error handling. Requirements: ● Call the callback with an error message if an error occurs; otherwise, pass the “response.” ● Handle the error gracefully by logging it if it occurs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
   <script>

function fetchData(url, callback) {
    console.log("Fetching data from:", url);


    setTimeout(function() {

        const hasError = Math.random() < 0.3;

        if (hasError) {
            const errorMessage = "Error: Unable to fetch data from " + url;
            console.error(errorMessage);
            callback(errorMessage, null);
        } else {
            const response = "Response from " + url;
            callback(null, response);
        }
    }, 2000);
}


const apiUrl = "https://api.example.com/data";


fetchData(apiUrl, function(error, response) {
    if (error) {
        console.log("Received:", error);
    } else {
        console.log("Received:", response);
    }
});

   </script>
</body>
</html>
```

Exercise 5: Using fetchData from Exercise 4, create another function processData that simulates processing the fetched data. Chain these functions together using nested callbacks. Requirements: ● First, call fetchData. Once the response is received, pass it to processData. ● processData should modify the data and log the processed result.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>
        // Function to simulate a network request with error handling
function fetchData(url, callback) {
    console.log("Fetching data from:", url);

    // Simulate a network delay of 2 seconds (2000 milliseconds)
    setTimeout(function() {
        // Simulate a random error occurrence
        const hasError = Math.random() < 0.3; // 30% chance of error

        if (hasError) {
            const errorMessage = "Error: Unable to fetch data from " + url; //
Simulated error message
            console.error(errorMessage); // Log the error message to the
console
            callback(errorMessage, null); // Call the callback with the error
message
        } else {
            const response = "Response from " + url; // Simulated successful
response
            callback(null, response); // Call the callback with null for error
and response
        }
    }, 2000);
}

// Function to process the fetched data
function processData(data, callback) {
    // Simulate data processing by modifying the response
    const processedData = data.toUpperCase(); // For example, convert the
response to uppercase
    callback(processedData); // Call the callback with the processed data
}

// Example usage
const apiUrl = "https://api.example.com/data";

// Call fetchData with a URL and a callback function
fetchData(apiUrl, function(error, response) {
    if (error) {
        console.log("Received:", error); // Log the error message
    } else {
        console.log("Received:", response); // Log the successful response

        // Once response is received, call processData
        processData(response, function(processedResult) {
            console.log("Processed Result:", processedResult); // Log the
processed result
        });
    }
});

    </script>
</body>
</html>
```