Coding Challenges

Assignment Answer Sheet

Assignment Answer Sheet

Coding Challenges on Error Handling and Prototypes

1. Double using callback. Write a function that takes in an array of integers and a callback function, and returns a new array where each element is doubled using the callback.

Ans:- ere is a function in JavaScript that takes an array of integers and a callback function to return a new array where each element is doubled using the callback:

javascript

Copy code

```
// Function to double each element using a callback
```

```
function doubleArray(arr, callback) {
  // Creating a new array where each element is doubled using the callback
  return arr.map(callback);
}

// Callback function to double a number
function double(num) {
  return num * 2;
}

// Example usage:
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = doubleArray(numbers, double);
```

```
console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

Explanation:

1. `doubleArray` function:
    - Takes an array `arr` and a callback `callback`.
    - Uses the `map` method to create a new array where each element of the original array is processed by the callback.
2. `double` function:
    - A simple callback function that takes a number as input and returns it multiplied by 2.
3. Example:
    - The array `[1, 2, 3, 4, 5]` is passed along with the `double` callback function.
    - The result is an array with each element doubled: `[2, 4, 6, 8, 10]`.
    - ipulated string: `HELLO WORLD`.

2. String Manipulation. Write a function "manipulateString" that takes in a string and converts the characters to uppercase letters. The function should return a callback function "logString" that logs the sentence "The manipulated string is: " along with the manipulated string or the new string to the console.

Ans:- Here is the JavaScript code for a `manipulateString` function that takes in a string, converts it to uppercase, and returns a callback function `logString` which logs the manipulated string to the console:

javascript

Copy code

```
// Function to manipulate the string and return a callback

function manipulateString(str) {

  // Convert the string to uppercase

  const upperStr = str.toUpperCase();
```

```
  // Callback function to log the manipulated
string

  function logString() {

    console.log("The manipulated string is: " +
upperStr);

  }



  // Return the callback function

  return logString;

}



// Example usage:

const manipulated = manipulateString("hello
world");

manipulated(); // Output: The manipulated string
```

```
is: HELLO WORLD
```

Explanation:

1. `manipulateString` function:
   - Takes a string `str` as input.
   - Converts the string to uppercase using the `toUpperCase()` method and stores it in `upperStr`.
2. `logString` callback function:
   - Logs the manipulated string with the sentence "The manipulated string is: " followed by the uppercase string.
3. Example usage:
   - The string `"hello world"` is passed to `manipulateString`.
   - The function returns the `logString` function, which when called, logs the manipulated string: `HELLO WORLD`.

3. Age in Days. Write a JavaScript function called ageInDays that accepts an object containing a person's first name, last name, and age in years as input. The function should concatenate the first and

last name into a single string and store it in a variable called fullName. It should then calculate the person's age in days and store it in a variable called ageInDays.

The ageInDays function should then return a callback function that logs a message to the console. The message should include the person's full name and age in days, and should be in the format: "The person's full name is [full name] and their age in days is [age in days]."

Note that the ageInDays function should not log the message to the console directly, but should instead return a callback function that can be used to log the message at a later time.

Ans:- Here is a JavaScript function `ageInDays` that takes an object with a person's details (first name, last name, and age in years) and returns a callback function to log the person's full name and age in days:

javascript

Copy code

```
// Function to calculate age in days and return a callback function

function ageInDays(person, logResult) {
```

```
  // Concatenate the first and last name into
fullName

  const fullName = `${person.firstName}
${person.lastName}`;



  // Calculate the person's age in days

  const ageInDays = person.age * 365;



  // Callback function to log the full name and
age in days

  function logMessage() {

    console.log(`The person's full name is
${fullName}, and their age in days is
${ageInDays}`);

  }
```

```
  // Return the callback function

  logResult(logMessage);

}


// Example usage:

const person = { firstName: 'Mithun', lastName:
'S', age: 20 };


// Callback to log the result

function logResult(callback) {

  callback(); // Logs the result when the
callback is invoked

}


// Call the function
```

```
ageInDays(person, logResult);


// Expected output:

// The person's full name is Mithun S, and their age in days is 7300
```

Explanation:

1. `ageInDays` function:
    - Takes an object `person` containing the first name, last name, and age.
    - Concatenates the `firstName` and `lastName` to form `fullName`.
    - Calculates the person's age in days by multiplying the `age` by 365.
    - Returns a callback function `logMessage` that logs the full name and age in days.
2. `logResult` function:
    - This is a generic callback function that will be passed as a parameter to `ageInDays`.
    - It takes the callback (`logMessage`) and executes it, logging the message to the console.

3. Example usage:
    - The person object `{ firstName: 'Mithun', lastName: 'S', age: 20 }` is passed to `ageInDays`.
    - The expected output will be: `The person's full name is Mithun S, and their age in days is 7300`.

4. Arrange in alphabetical order. Write a program that accepts a list of objects representing books [title, author, and year] and a callback function. The program should use the map function to create a new list containing only the titles of the books, and then pass this new list to the callback function. The callback function should then log the titles to the console in alphabetical order.

Ans:- Here is a JavaScript program that accepts a list of book objects, extracts the titles using `map()`, and passes them to a callback function that logs the titles in alphabetical order:

javascript

Copy code

```
// Function to process book list and sort titles alphabetically

function processBooks(books, callback) {

  // Use map to create a new list containing
```

```
only the titles
  const bookTitles = books.map(book =>
book.title);


  // Pass the list of titles to the callback
function
  callback(bookTitles);
}


// Callback function to log the titles in
alphabetical order
function logTitlesInOrder(titles) {
  // Sort the titles alphabetically
  const sortedTitles = titles.sort();


  // Log each title to the console
  console.log("Book titles in alphabetical
order:");
```

```
  sortedTitles.forEach(title =>
console.log(title));

}


// Example usage:

const books = [

  { title: "The Great Gatsby", author: "F. Scott
Fitzgerald", year: 1925 },

  { title: "Moby Dick", author: "Herman
Melville", year: 1851 },

  { title: "1984", author: "George Orwell",
year: 1949 },

  { title: "To Kill a Mockingbird", author:
"Harper Lee", year: 1960 }

];


// Call the processBooks function and pass
logTitlesInOrder as the callback

processBooks(books, logTitlesInOrder);
```

```
// Expected output:
// Book titles in alphabetical order:
// 1984
// Moby Dick
// The Great Gatsby
// To Kill a Mockingbird
```

Explanation:

1. `processBooks` **function:**

   Takes a list of book objects (`books`) and a callback function (`callback`).

   Uses the `map()` method to create a new array `bookTitles` that contains only the titles of the books.

   Passes this array to the callback function.

2. `logTitlesInOrder` **callback function:**

   Receives the array of titles as an argument.

Uses the `sort()` method to sort the titles in alphabetical order.

Logs the sorted titles to the console.

3. Example usage:

The list of book objects is passed to `processBooks`.

The callback `logTitlesInOrder` sorts the titles and logs them in alphabetical order:

css

Copy code

```
1984

Moby Dick

The Great Gatsby

To Kill a Mockingbird
```

5. Greeting Promise. You need to write a function that takes a name as input and returns a promise that resolves with a greeting message. The function should greet the person using

their name, with a message in the format "Hello, {name}!".

For example, if the input to the function is "Mithun", the promise should resolve with the string "Hello, Mithun!".

Ans:- Here's a JavaScript function that takes a name as input and returns a promise that resolves with a greeting message in the format `"Hello, {name}!"`:

javascript

Copy code

```
// Function to return a promise with a greeting message

function greetPerson(name) {

  return new Promise((resolve, reject) => {

    if (name) {

      // If name is provided, resolve with the greeting message

      resolve(`Hello, ${name}!`);

    } else {

      // If no name is provided, reject the promise with an error
```

```
      reject("Name is required");
    }
  });
}

// Example usage:
greetPerson("Mithun")
  .then((message) => {
    console.log(message); // Output: Hello, Mithun!
  })
  .catch((error) => {
    console.log(error);
  });
```

Explanation:

1. `greetPerson` **function:**
    - **Takes a** `name` **as input.**

- Returns a promise. Inside the promise:
  - If the `name` is provided, it resolves with the greeting message `"Hello, {name}!"`.
  - If the `name` is not provided, it rejects the promise with an error message `"Name is required"`.

2. Example usage:
   - Calling `greetPerson("Mithun")` resolves the promise with the message `"Hello, Mithun!"`.
   - The `then()` block logs the greeting message to the console.
   - If no name is provided, the `catch()` block will handle the error.

6. Fetch results asynchronously. Write a function that asynchronously fetches data from an API [https://jsonplaceholder.typicode.com/todos/1]and logs the result to the console.

Ans:- Here's a JavaScript function that asynchronously fetches data from the API `https://jsonplaceholder.typicode.com/todos/1` and logs the result to the console using `async`/`await`:

javascript

Copy code

```
// Function to asynchronously fetch data from the
API and log it

async function fetchTodo() {

  try {

    // Asynchronously fetch data from the API

    const response = await
fetch('https://jsonplaceholder.typicode.com/todos/1'
);



    // Parse the response as JSON

    const data = await response.json();



    // Log the result to the console
```

```
    console.log(data);
  } catch (error) {
    // Handle errors if the request fails
    console.error('Error fetching data:', error);
  }
}

// Example usage
fetchTodo();
```

Explanation:

1. `fetchTodo` **function:**

- Uses the `async` keyword to make the function asynchronous.
- Inside the function, `await` is used to wait for the promise returned by the `fetch()` function.
- Once the fetch request resolves, the response is parsed as JSON using `await response.json()`.
- If the request is successful, the data is logged to the console.
- In case of an error, the `catch` block logs the error message.

2. Example usage:
   - When `fetchTodo()` is called, it fetches the data from the API endpoint `https://jsonplaceholder.typicode.com/todos/1` and logs the result to the console.

Expected Output:

json

Copy code

```
{
```

```
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
```

7. Multiple requests. Create an asynchronous function that retrieves data from two different API endpoints: "https://jsonplaceholder.typicode.com/todos/1" and "https://jsonplaceholder.typicode.com/posts/1". The first API returns a to-do task, while the second API provides post details. The function should combine the results from both APIs and log them as an object, where the keys are "todo" and "post", and the corresponding values are the responses from the respective APIs.

Ans:- Here is an asynchronous JavaScript function that retrieves data from two different API endpoints: one for a to-do task and another for

post details. It combines the results into an object with keys `"todo"` and `"post"` and logs the combined object:

javascript

Copy code

```
// Asynchronous function to fetch data from two APIs

async function fetchTodoAndPost() {

  try {

    // Fetch data from both APIs simultaneously
using Promise.all

    const [todoResponse, postResponse] = await
Promise.all([


fetch('https://jsonplaceholder.typicode.com/todos/1'
),


fetch('https://jsonplaceholder.typicode.com/posts/1'
)

    ]);
```

```
  // Parse both responses as JSON
  const todo = await todoResponse.json();
  const post = await postResponse.json();

  // Combine the results into an object
  const combinedResult = {
    todo: todo,
    post: post
  };

  // Log the combined result
  console.log(combinedResult);
} catch (error) {
```

```
    // Handle any errors during fetching
    console.error('Error fetching data:', error);
  }
}

// Example usage
fetchTodoAndPost();
```

Explanation:

1. `fetchTodoAndPost` **function:**
   - **The** `Promise.all()` **method is used to send both API requests simultaneously. This helps avoid waiting for one request to finish before starting the other, improving efficiency.**
   - **After both requests resolve, the responses are parsed into JSON using** `await` **to handle them asynchronously.**
   - **The parsed results are combined into an object with keys** `"todo"` **and** `"post"`**, and the values**

are the corresponding API results.

- **The combined object is logged to the console.**

2. Error handling:

- **The `try`/`catch` block handles any potential errors that might occur during the fetch process, ensuring the program doesn't crash if something goes wrong.**

Expected Output:

json

Copy code

```
{
  "todo": {
    "userId": 1,
    "id": 1,
    "title": "delectus aut autem",
    "completed": false
  },
  "post": {
```

```
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident
occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit..."
  }
}
```


This logs the combined data from both endpoints into a single object.

8. Get Data from API and Display it on the browser console. Write a JavaScript program that uses the Fetch method to retrieve data from an API, and then logs the data to the console. For example, you could use the API at https://jsonplaceholder.typicode.com/posts to retrieve a list of posts, and then display them to the browser console

Ans:-JavaScript Program to Retrieve Data from an API and Display in the Browser Console

> **This program demonstrates how to use the Fetch method to retrieve data from the API `https://jsonplaceholder.typicode.com/posts` and then log the data to the browser console.**

Step-by-Step Code Implementation

javascript

Copy code

```
// Function to fetch data from the API and log it to the console

async function fetchPosts() {

  try {

    // Step 1: Fetch data from the API

    const response = await fetch('https://jsonplaceholder.typicode.com/posts');



    // Step 2: Convert the response to JSON
```

```
    const posts = await response.json();

    // Step 3: Log the data to the browser console
    console.log(posts);
  } catch (error) {
    // Step 4: Error handling
    console.error('Error fetching data:', error);
  }
}

// Step 5: Call the function to execute
fetchPosts();
```

Explanation of the Program

1. `fetchPosts` **Function:**

This asynchronous function handles the process of fetching data from the API.

2. **`fetch` Method:**
   The `fetch` method is used to make an HTTP request to the URL `https://jsonplaceholder.typicode.com/posts`. This method returns a promise.
3. **JSON Parsing:**
   Once the response is received, the `response.json()` method is used to convert the response into JSON format.
4. **Console Logging:**
   The JSON data (an array of posts) is logged to the browser console using `console.log()`.
5. **Error Handling:**
   The `try-catch` block is used to catch and log any potential errors that may occur during the fetch process.

Example Output

When this code runs, you should see an array of posts in the browser console, each containing an ID, title, body, and user ID:

json

Copy code

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit suscipit recusandae consequuntur expedita et cum ..."
  },
  ...
]
```

This is how you can effectively use the Fetch method to retrieve data from an API and display it on the browser console.

.

9. Error Handling Write a JavaScript program that uses the Fetch method to retrieve data from an API, and then handles errors that may occur. For example, you could use the API at https://jsonplaceholder.typicode.com/posts/123456789 to simulate an error, and then display an error message on the webpage.

Ans:- Here's how you can write a JavaScript program that uses the Fetch method to retrieve data from an API and handle errors that may occur. We will simulate an error by requesting an invalid URL (`https://jsonplaceholder.typicode.com/posts/123456789`) and then display the error message on the webpage.

JavaScript Program with Error Handling

html

Copy code

```
<!DOCTYPE html>

<html lang="en">

<head>
```

```
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Fetch API with Error Handling</title>
</head>
<body>

  <h1>API Data Fetch Example</h1>
  <div id="output"></div> <!-- This is where we'll
display the result or error -->

  <script>
    // Function to fetch data from an API and handle
errors
    async function fetchData() {
      const outputDiv =
```

```
document.getElementById('output');



      try {

        // Fetch from an invalid API endpoint to
simulate an error

        const response = await
fetch('https://jsonplaceholder.typicode.com/posts/12
3456789');



        // Check if the response is not OK (i.e.,
404 or other errors)

        if (!response.ok) {

          throw new Error(`Error: ${response.status}
- ${response.statusText}`);

        }



        // Parse the response as JSON
```

```
        const data = await response.json();


        // Display the data on the webpage
        outputDiv.innerHTML =
`<pre>${JSON.stringify(data, null, 2)}</pre>`;
      } catch (error) {
        // Display the error message on the webpage
        outputDiv.innerHTML = `<p style="color:
red;">${error.message}</p>`;
      }
    }


    // Call the fetchData function to execute
    fetchData();
  </script>
```

```
</body>

</html>
```

Explanation:

1. **HTML Structure:**
   - The `outputDiv` element (`<div id="output"></div>`) will display either the fetched data or the error message.
2. `fetchData` **Function:**
   - **Error Simulation:** The URL `https://jsonplaceholder.typicode.com/posts/123456789` is invalid and simulates a 404 error.
   - `response.ok` **Check:** The `response.ok` property checks if the HTTP status code is in the 200–299 range, which indicates a successful request. If not, we throw an error.
   - **Error Handling:** If the API returns an error, it's caught in the `catch` block, and the error message is displayed on the webpage.

3. **Error Display: The error message is styled in red (`<p style="color: red;">`).**

Example of Error Output: If the API endpoint is invalid, the output on the webpage might look something like this:
javascript
Copy code

```
Error: 404 - Not Found
```

4.

How It Works:

- **The program fetches data from an API.**
- **If the fetch request fails (e.g., due to a 404 error), the error is caught and displayed in the browser with an appropriate message.**
- **This example demonstrates how you can handle errors gracefully when working with external APIs.**