**Topics intended to be covered in this assignment**

* **Asynchronous programming concepts**
* **Promises, callbacks**
* **Promise methods, Error handling**

**Questions**

1. **Define ‘Asynchronous programming’.**

**Answer: Asynchronous programming** is a programming paradigm that allows tasks to run independently of the main program flow, enabling the program to continue executing other tasks while waiting for a particular operation (like a network request, file I/O, or database query) to complete.

1. **What are callbacks? Give an example.**

**Answer:** A **callback** is a function that is **passed as an argument to another function** and is **invoked after some operation is completed**. It allows asynchronous or deferred execution of code, making it a key concept in asynchronous programming, especially in JavaScript.

**Example:**

function fetchData(callback) {

console.log("Fetching data...");

setTimeout(function() {

console.log("Data fetched");

callback(); // calling the callback after data is fetched

}, 2000);

}

function processData() {

console.log("Processing data...");

}

fetchData(processData);

1. **What is a promise? What are its different states? Also write the methods which change a promise to their corresponding states.**

**Answer:**

A **Promise** is a JavaScript object that represents the eventual **completion (or failure)** of an asynchronous operation and its resulting value. It allows you to **handle asynchronous tasks** in a more manageable and readable way compared to traditional callbacks.

A Promise can be in one of **three states**:

| **State** | **Description** |
| --- | --- |
| **Pending** | Initial state, neither fulfilled nor rejected. |
| **Fulfilled** | The operation completed successfully. |
| **Rejected** | The operation failed with an error. |

### **How States Change:**

| **Method/Action** | **Resulting State** | **Description** |
| --- | --- | --- |
| resolve(value) | Fulfilled | Marks the promise as successful and returns a value. |
| reject(error) | Rejected | Marks the promise as failed with an error. |
| (No method called yet) | Pending | Default state when the Promise is created. |

**Example:**

let promise = new Promise(function(resolve, reject) {

let success = true;

if (success) {

resolve("Data fetched successfully"); // Fulfilled

} else {

reject("Failed to fetch data"); // Rejected

}

});

promise.then(result => console.log(result)) // Handles fulfillment

.catch(error => console.log(error)); // Handles rejection

1. **Write a JS promise which resolves after 5s, with a value of “Hello Javascript”. Also, write code to print the length of this string after resolving.**

**Answer:**

let myPromise = new Promise(function(resolve, reject) {

setTimeout(function() {

resolve("Hello Javascript");

}, 5000); // 5 seconds delay

});

myPromise.then(function(result) {

console.log("Resolved value:", result);

console.log("Length of string:", result.length);

}).catch(function(error) {

console.log("Error:", error);

});

1. **What is ‘callback hell’ problem? How does the use of promises help in reducing this?**

**Answer:**

**Callback Hell** is a situation in JavaScript where **multiple nested callbacks** are used to handle asynchronous operations, leading to:

* Deeply nested code
* Hard-to-read and maintain logic
* Difficult debugging and error handling

This often looks like a **"pyramid of doom"** or messy stair-like code structure.

Example:

doTask1(function(result1) {

doTask2(result1, function(result2) {

doTask3(result2, function(result3) {

doTask4(result3, function(result4) {

console.log("All tasks done"); }); }); }); });

1. **What is the difference between Promise.all() and Promise.allSettled()? Give an example.**

**Answer:**

Both Promise.all() and Promise.allSettled() are used to run multiple promises in parallel, but they behave **differently when some promises fail**.

### **1.** Promise.all()

* Waits for **all promises to fulfill**.
* **If any promise is rejected**, the **entire operation fails**, and it immediately returns the error.
* Does **not return partial results**.

#### **Use case:** When **all tasks must succeed** (e.g., loading multiple essential resources).

### **2.** Promise.allSettled()

* Waits for **all promises to settle**, regardless of whether they **fulfill or reject**.
* Always returns an array of results with:
  + { status: "fulfilled", value: ... } or
  + { status: "rejected", reason: ... }

#### **Use case:** When you want to know **outcomes of all promises**, even if some fail (e.g., logging results of multiple API calls).

Example:

const p1 = Promise.resolve("Task 1 done");

const p2 = Promise.reject("Task 2 failed");

const p3 = Promise.resolve("Task 3 done");

Promise.all([p1, p2, p3]) // Using Promise.all()

.then(results => {

console.log("Promise.all results:", results); })

.catch(error => {

console.log("Promise.all error:", error); });

Promise.allSettled([p1, p2, p3]) // Using Promise.allSettled()

.then(results => {

console.log("Promise.allSettled results:", results);

});

1. **Write a JS function that throws a SyntaxError if the value age (passed as parameter of the function) is greater than 80. The error message will be “You are too old.”**

**Answer:**

function myFunction(age){

if(age>=80)

throw new Error("You are too old."); }

try{ console.log(myFunction(82)); }

catch(error){ console.log("An error occured.", error.message); }

1. **For this code, answer the following:  
     
   const p1 = Promise.any([  
    new Promise((resolve, reject) => setTimeout(() => reject(“Part 1 is rejected.”), 2000)),  
    new Promise((resolve, reject) => setTimeout(() => resolve(“Part 2 is resolved.”), 5000)),  
   ]);  
     
   p1.then((res) => console.log(res))**
   1. **Write the output of this code. Briefly explain the reasoning.**

**Answer:**

Output: Part 2 is resolved

We are using Promise.any() which:

* Takes an **array of promises**.
* **Resolves** as soon as **any one** of the promises **fulfills**.
* If **all promises reject**, then it throws an **AggregateError**.
  1. **What will be the output of this code if Promise.any() is replaced by Promise.race()? Is the output for this case generated faster or slower? Give reasons.**

**Answer:**

The function call needs to be modified with,

p1.then((res) => console.log(res)).catch((err) => console.log("Error:", err));

Output: ERROR!

Error: Part 1 is rejected

Explanation:

#### With Promise.any():

* The first promise **rejects after 2 seconds** → ignored.
* The second promise **resolves after 5 seconds** → used.
* So the result is shown **after 5 seconds**.

#### With Promise.race():

* The first promise **rejects after 2 seconds** → immediately triggers rejection.
* So the output (error) is shown **after 2 seconds**.

### **Why this happens:**

* **Promise.any()** waits for the **first promise that fulfills**, ignoring all rejections.
* **Promise.race()** settles as soon as **any promise settles**, whether it's a **resolve or reject**.

1. **Given is a code snippet**

**const p = new Promise((resolve, reject) => {**

**resolve(“Part a”);**

**resolve(“Part b”);**

**})**

**p.then((res)=>console.log(res));**

* 1. **What will be the output of the above? Justify your answer.**

**Answer:**

Part a

* 1. **What code should we write to get both the resolved results.**

**Answer:**

const p1 = Promise.resolve("Part a");

const p2 = Promise.resolve("Part b");

Promise.all([p1, p2]).then(results => {

console.log(results[0]); // Part a

console.log(results[1]); // Part b

});

And there can be another method by using multiple then we can have both the output,

Promise.resolve("Part a")

.then(res1 => {

console.log(res1);

return "Part b";

})

.then(res2 => {

console.log(res2);

});

1. **Using async / await syntax, and fetch (), make an API request to https://official-joke-api.appspot.com/random\_joke   
   and console the results. Also handle errors if any.**

**Answer:**

async function getJoke() {

try {

const response = await fetch("https://official-joke-api.appspot.com/random\_joke");

// Check if response is OK

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`); }

const data = await response.json(); // Parse JSON response

console.log("Joke setup:", data.setup);

console.log("Punchline:", data.punchline);

} catch (error) {

console.error("Error fetching joke:", error.message); } }

// Call the function

getJoke();