|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Web Programming Laboratory 116U40L501** | **Semester:** | **V** |
| **Date of Performance:** | **30 / 09 / 2024** | **Batch No.:** | **B – 1** |
| **Faculty Name:** | **Prof. Madhura Pednekar** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No.: 4**

**Title: Implementation of Advanced JavaScript**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Implementation of Advanced JavaScript. |

|  |
| --- |
| **COs to be achieved:** |
| **CO3:** Create interactive web content with JavaScript, including form validation and event handling. |

|  |
| --- |
| **Apparatus/Software tools used:** |
| * VS Code * Web Browser |

|  |
| --- |
| **­­­­­Problem Statement:** |
| 1. **WAP to show the concept of Promises, Asyn and Await.** 2. **Build a simple weather application that fetches weather data asynchronously using callbacks. The application will allow users to input a city name, and upon submission, it will fetch the weather information for that city using a mock API call. The result will be displayed on the screen.** |

|  |
| --- |
| **JavaScript Code:** |
| 1. **Promises, Asyn, Await:**   // Function that returns a Promise which resolves after 2 seconds  function resolveAfter2Seconds() {      return new Promise((resolve) => {          setTimeout(() => {              resolve('Resolved after 2 seconds');          }, 2000);      });  }  // Using Async and Await to wait for the Promise to resolve  async function asyncFunction() {      console.log('Before waiting for promise');        // Await for the Promise to resolve      const result = await resolveAfter2Seconds();        console.log(result);  // Output: 'Resolved after 2 seconds'      console.log('After promise resolved');  }  // Execute the async function  asyncFunction(); |
| 1. **Simple Weather application:**   <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Weather App with Callbacks</title>  </head>  <body>      <h1>Simple Weather Application</h1>      <input type="text" id="cityInput" placeholder="Enter city name" />      <button onclick="getWeather()">Get Weather</button>      <p id="weatherInfo"></p>      <script>          // Mock function to simulate an API call          function mockFetchWeather(city, callback) {              console.log('Fetching weather data...');              setTimeout(() => {                  const weatherData = {                      city: city,                      temperature: Math.floor(Math.random() \* 35) + 1,  // Random temperature between 1 and 35                      condition: "Sunny"                  };                  callback(null, weatherData);  // Return weather data through callback              }, 2000);  // Simulating a delay of 2 seconds          }          // Function to get the weather and display it          function getWeather() {              const city = document.getElementById('cityInput').value;              if (!city) {                  alert('Please enter a city name.');                  return;              }              // Fetch the weather using a callback              mockFetchWeather(city, (err, data) => {                  if (err) {                      document.getElementById('weatherInfo').textContent = 'Error fetching weather.';                  } else {                      document.getElementById('weatherInfo').textContent =                          `Weather in ${data.city}: ${data.temperature}°C, ${data.condition}`;                  }              });          }      </script>  </body>  </html> |

|  |
| --- |
| **Stepwise-Procedure / Algorithm:** |
| 1. **Promises, Asyn, Await:**  * Define a function that returns a Promise. * Use setTimeout inside the Promise to simulate a delay. * Define an async function. * Inside the async function, log a message before the promise is resolved. * Use await to pause execution until the Promise is resolved. * Log the resolved value of the Promise. * Execute the async function. |
| 1. **Simple Weather application:**  * Create an HTML structure with:   + An input field to enter the city name.   + A button to trigger the weather-fetching function.   + A paragraph element to display the weather result. * Write a JavaScript function to simulate an API call using setTimeout with a callback function. * In the callback, simulate fetching weather data for the entered city. * Update the paragraph element's content with the fetched weather data. * Link the button click event to call the weather-fetching function. |

|  |
| --- |
| **Output** |
| 1. **Promises, Asyn, Await:** |
| 1. **Simple Weather application:** |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| **Students need to mention and explain with syntax the tags used for problem statement they have selected.**   1. **Promises, Asyn, Await:**  * Promise: Represents an operation that will complete in the future (resolved/rejected). * Syntax:   new Promise(function(resolve, reject) {  // operation  });   * Async: Used to define an asynchronous function that allows the use of await inside. * Syntax:   async function functionName() {  // code  }   * Await: Pauses the execution of the async function until the promise resolves. * Syntax**:**   let result = await promise;   1. **Simple Weather application:**  * HTML Tags: * <input>: Creates an input field for user data. * <button>: Creates a clickable button. * <p>: Defines a paragraph for displaying information. * JavaScript Elements: * Callback Functions: A function passed as an argument to another function, invoked when the asynchronous task (e.g., API call) is completed. * setTimeout: Delays the execution of code for a specific duration, simulating asynchronous operations like API calls. * DOM Manipulation: JavaScript can be used to manipulate HTML elements dynamically, such as updating text content, using methods like document.getElementById. |

|  |
| --- |
| **Conclusion:** |
| In conclusion, the first program demonstrated how to handle asynchronous operations in JavaScript using **Promises**, **async**, and **await**, allowing for cleaner and more readable code. The second program illustrates how to fetch data asynchronously using **callbacks**, showcasing basic event handling and DOM manipulation to create an interactive weather application. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |