**Batch: D - 1 Roll No.: 16010122096**

**Experiment / assignment / tutorial No. 01**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| Title: Implementation of Advanced JavaScript Concept |

**AIM:** To Implement the Concept of Advanced JavaScript

**Problem Definition:**

-Demonstrate the Concept of Advanced JavaScript With the help of Example.

\*(Students have to perform the task assigned within group and demonstrate the same).

**Resources used:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

**CO 1:**.Build full stack applications in JavaScript using the MERN technologies.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Shelly Powers Learning Node O’ Reilly 2 nd Edition, 2016.

**Pre Lab/ Prior Concepts:**

Before implementing callbacks, promises, and `async/await`, understand JavaScript functions, asynchronous programming, and how to handle async operations. Familiarize yourself with `setTimeout`, `Promise` objects, and `async`/`await` syntax to effectively manage async tasks and improve code readability and error handling.

**Methodology:**

1]

function asyncOperation(callback) {

    // Simulate async operation

    setTimeout(() => {

        let result = "Data";

        callback(null, result); // On success

        // callback("Error", null); // On error

    }, 1000);

}

// Using the callback

asyncOperation((error, result) => {

    if (error) {

        console.error(error);

    } else {

        console.log(result);

    }

});

2]

function asyncOperation() {

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

        // Simulate async operation

        setTimeout(() => {

            let result = "Data";

            resolve(result); // On success

            // reject("Error"); // On error

        }, 1000);

    });

}

// Using the promise

asyncOperation()

    .then(result => console.log(result))

    .catch(error => console.error(error));

3]

function asyncOperation() {

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

        // Simulate async operation

        setTimeout(() => {

            let result = "Data";

            resolve(result); // On success

            // reject("Error"); // On error

        }, 1000);

    });

}

async function fetchData() {

    try {

        let result = await asyncOperation();

        console.log(result);

    } catch (error) {

        console.error(error);

    }

}

// Call the async function

fetchData()

**Implementation Details:**

**1] Callback Hell**

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Callback Hell Example</title>

</head>

<body>

    <h2>Callback Hell</h2>

    <br><br>

    <button *onclick*="calculateResults()">Calculate All</button>

    <p *id*="resultsOutput"></p>

    <script>

        const arr = [1, 2, 3, 4, 5];

        function calculateResults() {

            calculateSum((*sum*) => {

                calculateProduct(*sum*, (*product*) => {

                    calculateAverage(*product*, (*average*) => {

                        calculateMax(*average*, (*max*) => {

                            displayResults(*sum*, *product*, *average*, *max*);

                        });

                    });

                });

            });

        }

        function calculateSum(*callback*) {

            let sum = 0;

            arr.forEach(*element* => {

                sum += *element*;

            });

            setTimeout(() => callback(sum), 100);

        }

        function calculateProduct(*sum*, *callback*) {

            let product = 1;

            arr.forEach(*element* => {

                product \*= *element*;

            });

            setTimeout(() => callback(product), 100);

        }

        function calculateAverage(*product*, *callback*) {

            let average = arr.reduce((*a*, *b*) => *a* + *b*, 0) / arr.length;

            setTimeout(() => callback(average), 100);

        }

        function calculateMax(*average*, *callback*) {

            let max = Math.max(...arr);

            setTimeout(() => callback(max), 100);

        }

        function displayResults(*sum*, *product*, *average*, *max*) {

            document.getElementById('resultsOutput').innerText =

                `Sum: ${*sum*}, Product: ${*product*}, Average: ${*average*}, Max: ${*max*}`;

        }

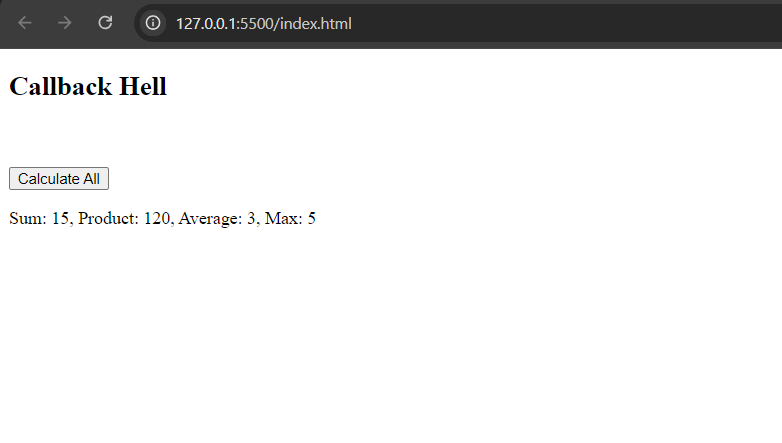
    </script>

</body>

</html>

**Output:**

****

****

**Promise Chaining:**

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Promise Chaining Example</title>

</head>

<body>

    <h2>Promise Chaining</h2>

    <br><br>

    <button *onclick*="calculateResults()">Calculate All</button>

    <p *id*="resultsOutput"></p>

    <script>

        const arr = [1, 2, 3, 4, 5];

        function calculateResults() {

            calculateSum()

                .then(*sum* => {

                    return calculateProduct(*sum*).then(*product* => ({ sum, product }));

                })

                .then(({ *sum*, *product* }) => {

                    return calculateAverage(*product*).then(*average* => ({ sum, product, average }));

                })

                .then(({ *sum*, *product*, *average* }) => {

                    return calculateMax().then(*max* => ({ sum, product, average, max }));

                })

                .then(({ *sum*, *product*, *average*, *max* }) => {

                    displayResults(*sum*, *product*, *average*, *max*);

                })

                .catch(*error* => {

                    console.error('Error occurred:', *error*);

                });

        }

        function calculateSum() {

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

                try {

                    let sum = 0;

                    arr.forEach(*element* => {

                        sum += *element*;

                    });

                    setTimeout(() => resolve(sum), 100);

                } catch (error) {

                    reject(error);

                }

            });

        }

        function calculateProduct(*sum*) {

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

                try {

                    let product = 1;

                    arr.forEach(*element* => {

                        product \*= *element*;

                    });

                    setTimeout(() => resolve(product), 100);

                } catch (error) {

                    reject(error);

                }

            });

        }

        function calculateAverage(*product*) {

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

                try {

                    let average = arr.reduce((*a*, *b*) => *a* + *b*, 0) / arr.length;

                    setTimeout(() => resolve(average), 100);

                } catch (error) {

                    reject(error);

                }

            });

        }

        function calculateMax() {

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

                try {

                    let max = Math.max(...arr);

                    setTimeout(() => resolve(max), 100);

                } catch (error) {

                    reject(error);

                }

            });

        }

        function displayResults(*sum*, *product*, *average*, *max*) {

            document.getElementById('resultsOutput').innerText =

                `Sum: ${*sum*}, Product: ${*product*}, Average: ${*average*}, Max: ${*max*}`;

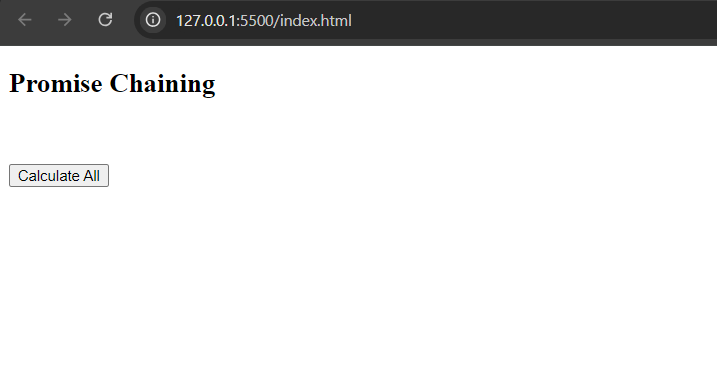
        }

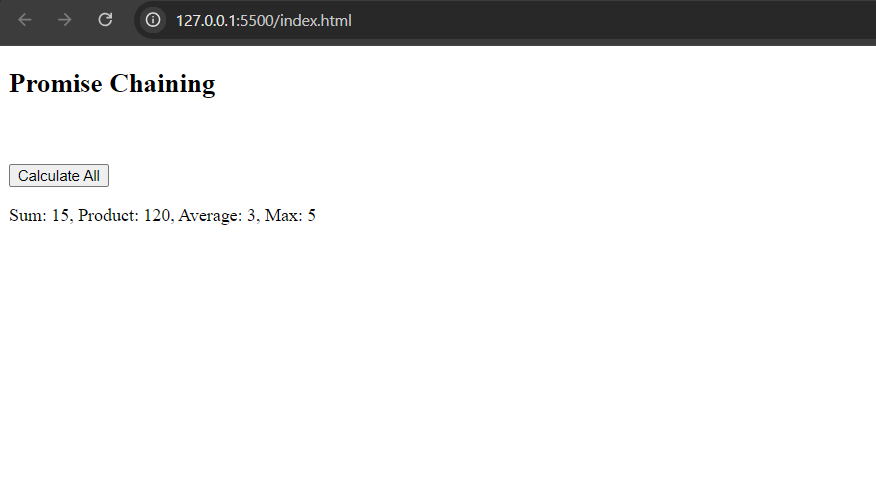
    </script>

</body>

</html>

**Output:**

****

****

**Async Await:**

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Async/Await Example</title>

</head>

<body>

    <h2>Async/Await</h2>

    <br><br>

    <button *onclick*="calculateResults()">Calculate All</button>

    <p *id*="resultsOutput"></p>

    <script>

        const arr = [1, 2, 3, 4, 5];

        async function calculateResults() {

            try {

                const sum = await calculateSum();

                const product = await calculateProduct(sum);

                const average = await calculateAverage();

                const max = await calculateMax();

                displayResults(sum, product, average, max);

            } catch (error) {

                console.error('Error occurred:', error);

            }

        }

        function calculateSum() {

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

                try {

                    let sum = 0;

                    arr.forEach(*element* => {

                        sum += *element*;

                    });

                    setTimeout(() => resolve(sum), 100);

                } catch (error) {

                    reject(error);

                }

            });

        }

        function calculateProduct(*sum*) {

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

                try {

                    let product = 1;

                    arr.forEach(*element* => {

                        product \*= *element*;

                    });

                    setTimeout(() => resolve(product), 100);

                } catch (error) {

                    reject(error);

                }

            });

        }

        function calculateAverage() {

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

                try {

                    let average = arr.reduce((*a*, *b*) => *a* + *b*, 0) / arr.length;

                    setTimeout(() => resolve(average), 100);

                } catch (error) {

                    reject(error);

                }

            });

        }

        function calculateMax() {

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

                try {

                    let max = Math.max(...arr);

                    setTimeout(() => resolve(max), 100);

                } catch (error) {

                    reject(error);

                }

            });

        }

        function displayResults(*sum*, *product*, *average*, *max*) {

            document.getElementById('resultsOutput').innerText =

                `Sum: ${*sum*}, Product: ${*product*}, Average: ${*average*}, Max: ${*max*}`;

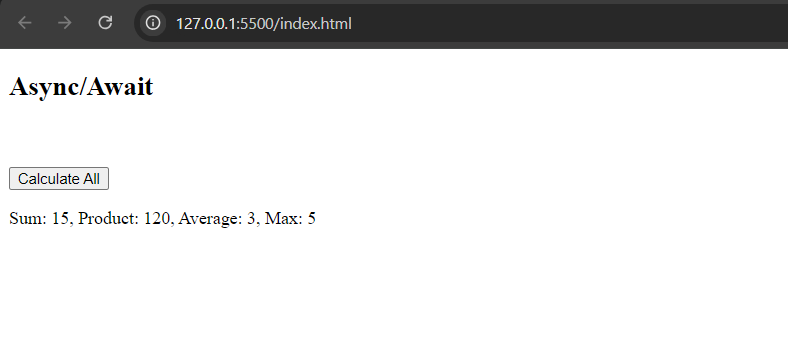
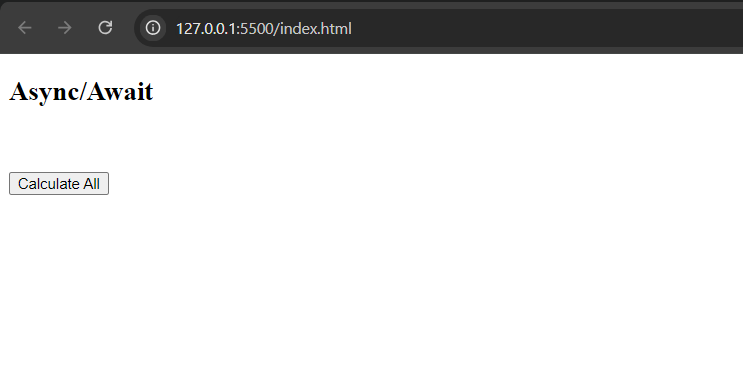
        }

    </script>

</body>

</html>

**Output:**

****

**2] Filter Arrays**

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Unique Elements</title>

</head>

<body>

    <h2>Finding Unique Elements</h2>

    <br><br>

    <button *onclick*="uniqueElements()">Show Unique Elements</button>

    <p *id*="uniqueOutput"></p>

    <script>

        function uniqueElements() {

            const arr = [1, 2, 3, 4, 5, 1, 2, 3, 6];

            const uniqueArr = arr.filter((*value*, *index*, *self*) => *self*.indexOf(*value*) === *index*);

            document.getElementById('uniqueOutput').innerText =

                `Initial Array: ${arr.join(', ')}\nUnique Elements: ${uniqueArr.join(', ')}`;

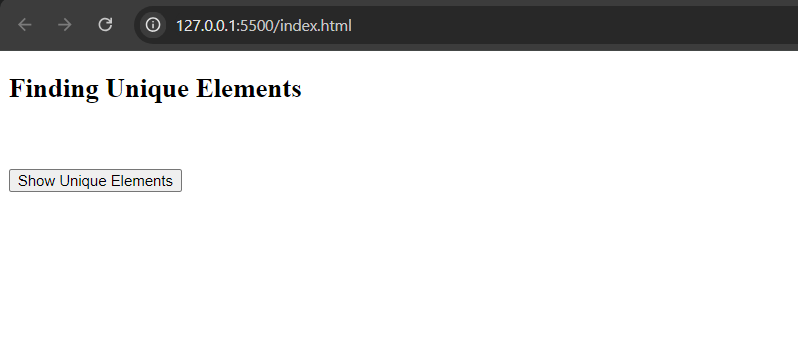
        }

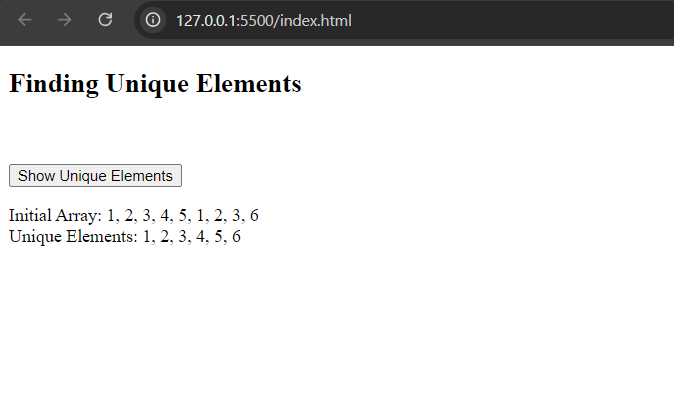
    </script>

</body>

</html>

**Output:**





**3] Filter Anagrams:**

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Filter Anagrams</title>

    <style>

        #anagramOutput {

            white-space: pre-wrap; */\* Ensures the text is displayed with line breaks \*/*

        }

    </style>

</head>

<body>

    <h2>Filter Anagrams</h2>

    <br><br>

    <button *onclick*="filterAnagrams()">Show Anagrams</button>

    <p *id*="anagramOutput"></p>

    <script>

        function filterAnagrams() {

            const words = ["listen", "silent", "enlist", "hello", "world", "dell", "led"];

            function sortedString(*str*) {

                return *str*.split('').sort().join('');

            }

            const anagramGroups = new Map();

            words.forEach(*word* => {

                const sorted = sortedString(*word*);

                if (!anagramGroups.has(sorted)) {

                    anagramGroups.set(sorted, []);

                }

                anagramGroups.get(sorted).push(*word*);

            });

            const filteredAnagrams = Array.from(anagramGroups.values()).filter(*group* => *group*.length > 1);

            const uniqueAnagrams = [...new Set(filteredAnagrams.flat())];

            document.getElementById('anagramOutput').innerText = 'Anagrams: ' + uniqueAnagrams.join(', ');

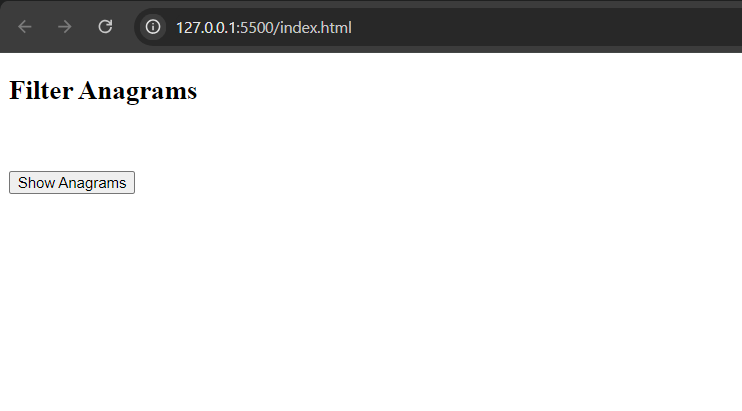
        }

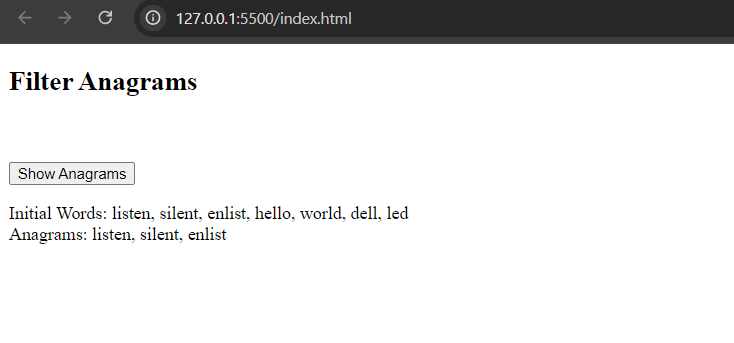
    </script>

</body>

</html>

**Output:**





4] **Iterable Keys:**

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Iterable Keys</title>

</head>

<body>

    <h2>Object Keys</h2>

    <button *onclick*="showObjectKeys()">Show Object Keys</button>

    <p *id*="objectKeysOutput"></p>

    <script>

        function showObjectKeys() {

            const obj = { a: 1, b: 2, c: 3 };

            const keys = Object.keys(obj);

            document.getElementById('objectKeysOutput').innerText = 'Object Keys: ' + keys.join(', ');

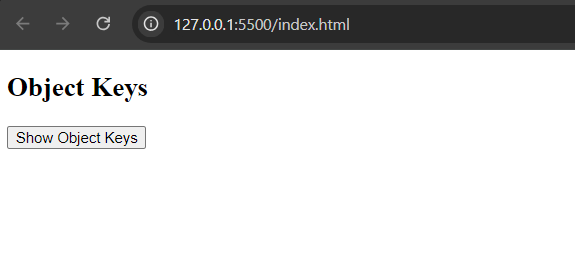
        }

    </script>

</body>

</html>

**Output:**



****

**5] Fetch users from github:**

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Fetch GitHub Users</title>

  </head>

  <body>

    <h2>Fetch GitHub Users</h2>

    <button onclick="fetchGitHubUsers()">Get GitHub Users</button>

    <pre id="userOutput"></pre>

    <script>

      const URL = "https://api.github.com/users";

      const fetchGithubs = async (users) => {

        const answer = [];

        for (const user of users) {

          const res = await fetch(`${URL}/${user}`);

          const data = await res.json();

          // Remove null fields

          const cleanedData = Object.fromEntries(

            Object.entries(data).filter(([\_, v]) => v != null)

          );

          answer.push(cleanedData);

        }

        return answer;

      };

      async function fetchGitHubUsers() {

        const users = [

          "romill-09",

          "siddhi0138",

          "RahilKothari9",

          "Arya1754",

          "Vivaan025",

          "harshnagrani",

        ];

        const userData = await fetchGithubs(users);

        document.getElementById("userOutput").innerText = JSON.stringify(

          userData,

          null,

          2

        );

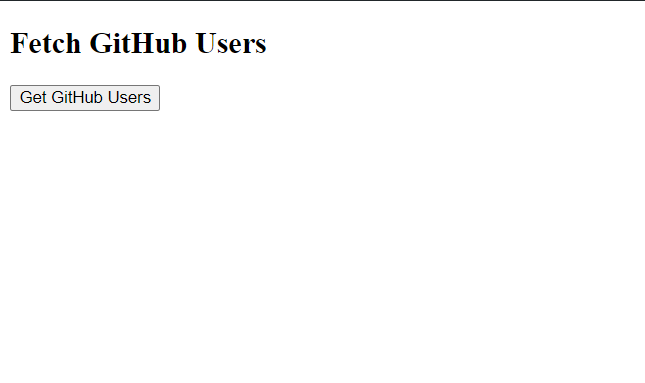
      }

    </script>

  </body>

</html>

**Output:**

****

****

**Steps for execution:**

1. **Setup Environment**: Create or open your HTML file.
2. **Define HTML Structure**: Add buttons, paragraphs, or <pre> elements for user interaction and output display.
3. **Implement JavaScript Functions**: Write functions for tasks such as filtering unique elements, detecting anagrams, and fetching data from GitHub.
4. **Attach JavaScript Functions**: Link functions to HTML elements using event handlers.
5. **Test Functionality**: Verify each function works as intended through user interactions.
6. **Handle Errors**: Implement error handling to manage any issues that arise during execution.
7. **Display Results**: Update HTML elements to show the results from your JavaScript functions.

**Conclusion:** Implementing JavaScript functions with HTML elements involves setting up the environment, defining structure, coding functions, linking interactions, testing, handling errors, and displaying results. This approach ensures functional and interactive web features.