K. J. Somaiya College of Engineering, Mumbai-77
Department of Computer Engineering

Batch: D-2 **Roll No.:** 16010122151
Experiment / assignment / tutorial No.__1__
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:

1. Google
2. Geeksforgeeks
3. Tutorialspoint
4. VSCode

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:

Methodology:

1. Callback Hell, Promise Chaining, and Async/Await

Callback Hell

1. Define Functions: Create functions with callbacks for asynchronous tasks.
2. Nest Callbacks: Chain multiple asynchronous operations using callbacks.
3. Handle Errors: Include error handling for each callback.

Promise Chaining

1. Define Functions Returning Promises: Create functions that return promises.
2. Chain Promises: Use .then() to chain multiple asynchronous operations.
3. Handle Errors: Add .catch() to handle errors in the chain.

Async/Await

1. Define Async Function: Create an async function to handle asynchronous operations.
2. Use Await: Inside the async function, use await to pause execution until promises resolve.
3. Handle Errors: Use try...catch blocks to handle errors.

2. Filter Unique Array Members

1. Define Function: Create a function unique(arr) that takes an array as input.
2. Use a Set: Convert the array to a Set to remove duplicates.
3. Return Array: Convert the Set back to an array and return it.

3. Filter Anagrams

1. Define Function: Create a function to filter anagrams from an array of words.
2. Sort Characters: For each word, sort its characters and use this as a key.

3. Group Words: Group words with the same key (sorted characters) as anagrams.
4. Return Groups: Return the grouped anagrams.

4. Iterable Keys

1. Create a Map: Define a Map object with key-value pairs.
2. Get Keys: Retrieve the keys from the map using map.keys().
3. Convert to Array: Convert the iterable keys to an array using Array.from().
4. Apply Array Methods: Use array methods like .push() on the array of keys.

5. Fetch Users from GitHub

1. Define Async Function: Create an async function getUsers(names) that takes an array of GitHub usernames.
2. Fetch Data: Use fetch() to request user data from the GitHub API for each username.
3. Handle Responses: Convert the responses to JSON.
4. Return Users: Collect and return the array of user data

Implementation Details:

Task

1) WAP for Callback hell, Promise Chaining and Async await for any application
Code:

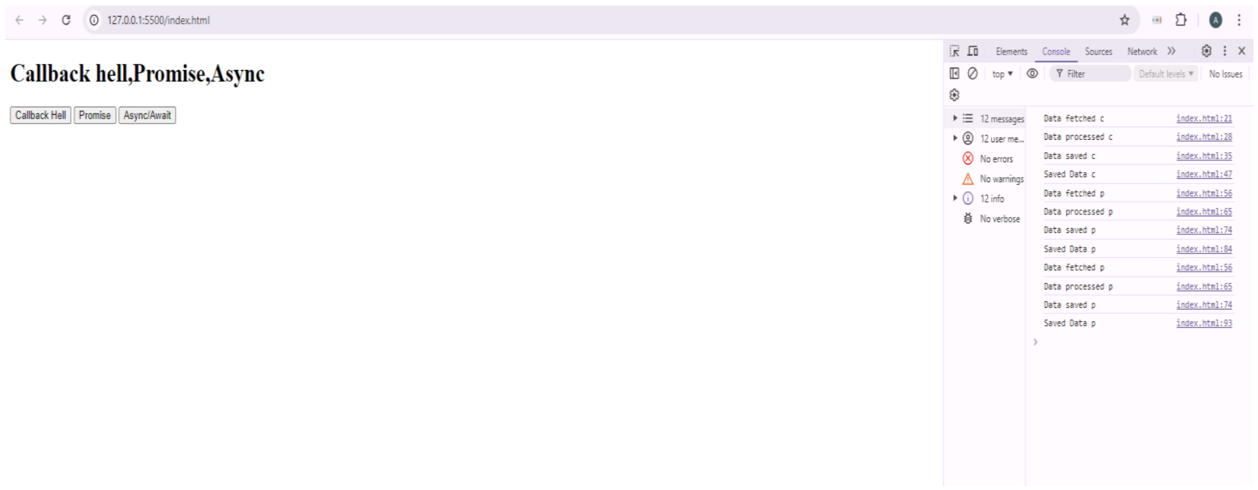
```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

```
    <title>Callback hell,Promise,Async</title>
</head>
<body>
    <h1>Callback hell,Promise,Async</h1>
    <button id="run-callback-hell">Callback Hell</button>
    <button id="run-promise-chaining">Promise </button>
    <button id="run-async-await">Async/Await</button>
    <script>
        function getData(callback) {
            setTimeout(() => {
                console.log("Data fetched c");
                callback(null, "Data c");
            }, 1000);
        }
        function processData(data, callback) {
            setTimeout(() => {
                console.log("Data processed c");
                callback(null, "Processed Data c");
            }, 1000);
        }
```

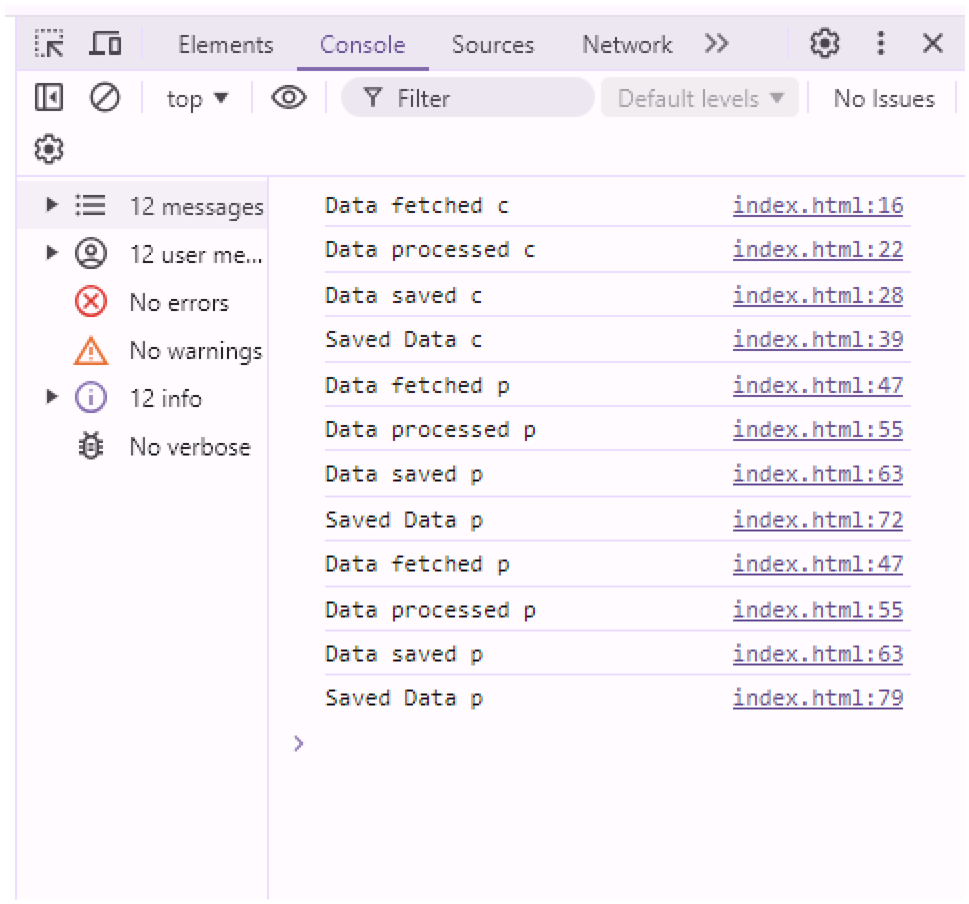
```
function saveData(data, callback) {
    setTimeout(() => {
        console.log("Data saved c");
        callback(null, "Saved Data c ");
    }, 1000);
}
document.getElementById('run-callback-hell').addEventListener('click', () => {
    getData((err, data) => {
        if (err) throw err;
        processData(data, (err, processedData) => {
            if (err) throw err;
            saveData(processedData, (err, savedData) => {
                if (err) throw err;
                console.log(savedData);
            });
        });
    });
});
function getDataPromise() {
    return new Promise((resolve) => {
        setTimeout(() => {
            console.log("Data fetched p");
            resolve("Data p");
        }, 1000);
    });
}
function processDataPromise(data) {
    return new Promise((resolve) => {
        setTimeout(() => {
            console.log("Data processed p");
            resolve("Processed Data p");
        }, 1000);
    });
}
function saveDataPromise(data) {
    return new Promise((resolve) => {
        setTimeout(() => {
            console.log("Data saved p");
            resolve("Saved Data p");
        }, 1000);
        });
    }
    document.getElementById('run-promise-chaining').addEventListener('click', () => {
        getDataPromise()
            .then(processDataPromise)
            .then(saveDataPromise)
            .then((savedData) => console.log(savedData));
    });
    async function handleData() {
        try {
            const data = await getDataPromise();
            const processedData = await processDataPromise(data);
            const savedData = await saveDataPromise(processedData);
            console.log(savedData);
        } catch (error) {
            console.error(error);
        }
    }
    document.getElementById('run-async-await').addEventListener('click', handleData);
</script>
```

```
</body>

</html>
```

Output Screenshots :



```
Data fetched c          index.html:21
Data processed c        index.html:28
Data saved c            index.html:35
Saved Data c            index.html:47
Data fetched p          index.html:56
Data processed p        index.html:65
Data saved p            index.html:74
Saved Data p            index.html:84
Data fetched p          index.html:56
Data processed p        index.html:65
Data saved p            index.html:74
Saved Data p            index.html:93
```



```
Data fetched c          index.html:16
Data processed c        index.html:22
Data saved c            index.html:28
Saved Data c            index.html:39
Data fetched p          index.html:47
Data processed p        index.html:55
Data saved p            index.html:63
Saved Data p            index.html:72
Data fetched p          index.html:47
Data processed p        index.html:55
Data saved p            index.html:63
Saved Data p            index.html:79
```

2) Filter unique array members

Let arr be an array.

Create a function unique(arr) that should return an array with unique items of arr

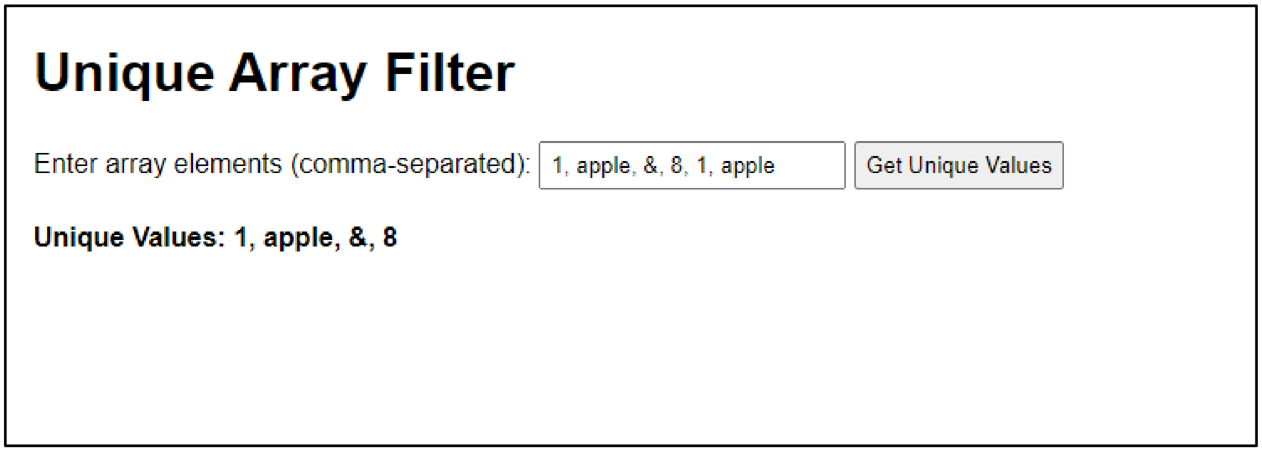
Code:

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>

    <meta charset="UTF-8">

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

    <title>Unique Array Filter</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            margin: 20px;

        }

        input,

        button {

            margin-bottom: 10px;

            padding: 5px;

        }


        #result {

            margin-top: 10px;

            font-weight: bold;

        }

    </style>
```

```
</head>


<body>
    <h1>Unique Array Filter</h1>
    <label for="arrayInput">Enter array elements (comma-separated):</label>
    <input type="text" id="arrayInput" placeholder="Enter values">
    <button onclick="filterUnique()">Get Unique Values</button>
    <div id="result"></div>


    <script>
        function unique(arr) {
            return Array.from(new Set(arr));
        }


        function filterUnique() {
            const input = document.getElementById('arrayInput').value;
            const array = input.split(',').map(item => item.trim()).filter(item => item !== '');



            const uniqueValues = unique(array);
            document.getElementById('result').textContent = `Unique Values:
${uniqueValues.join(', ')}`;

        }

    </script>

</body>


</html>
```

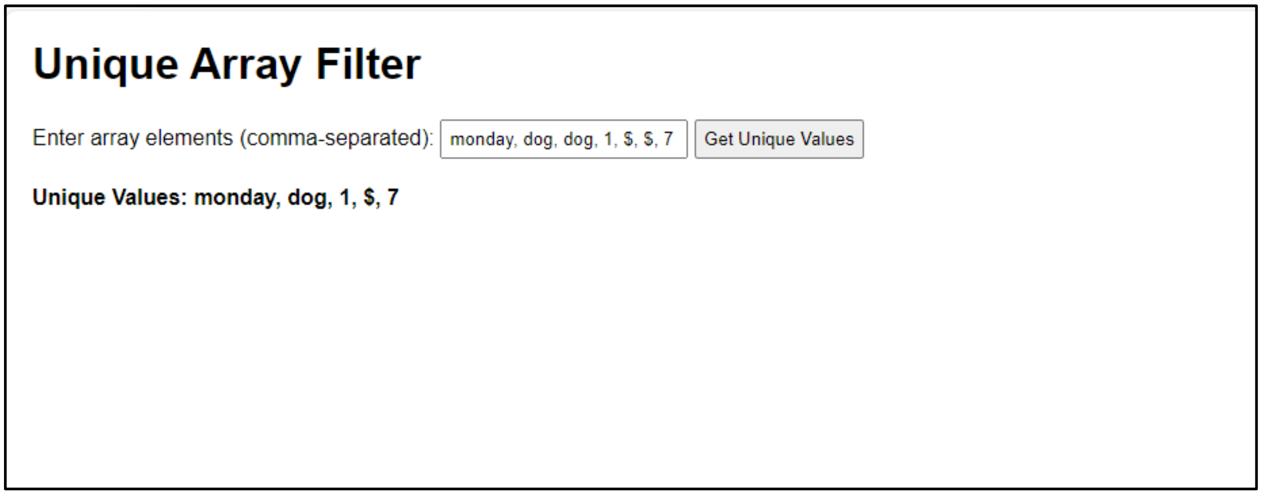
Output Screenshot:



Unique Array Filter

Enter array elements (comma-separated): 1, apple, &, 8, 1, apple

Unique Values: 1, apple, &, 8


Unique Array Filter

Enter array elements (comma-separated): monday, dog, dog, 1, $, $, 7 Get Unique Values

Unique Values: monday, dog, 1, $, 7

3) Filter anagrams

Anagrams are words that have the same number of same letters, but in different order.

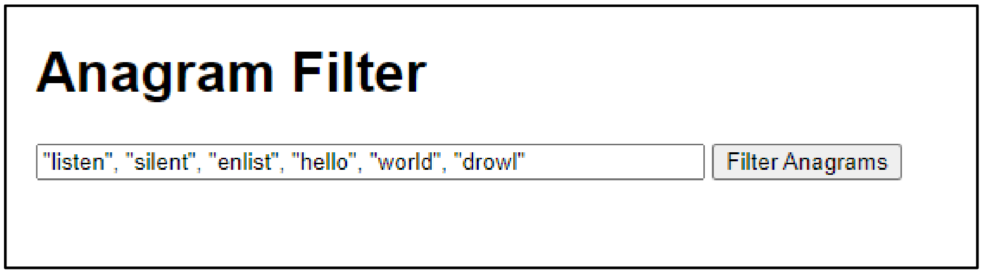
Code :

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Anagram Filter</title>

    <style>

        body {

            font-family: Arial, sans-serif;
            margin: 20px;
        }
        #output {
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <h1>Anagram Filter</h1>
    <input type="text" id="inputWords" placeholder="Enter words separated by commas"
size="50">
    <button onclick="filterAndDisplayAnagrams()">Filter Anagrams</button>
    <div id="output"></div>

    <script>
        // Function to sort the characters in a word
        function sortWord(word) {
            return word.split('').sort().join('');
        }
```

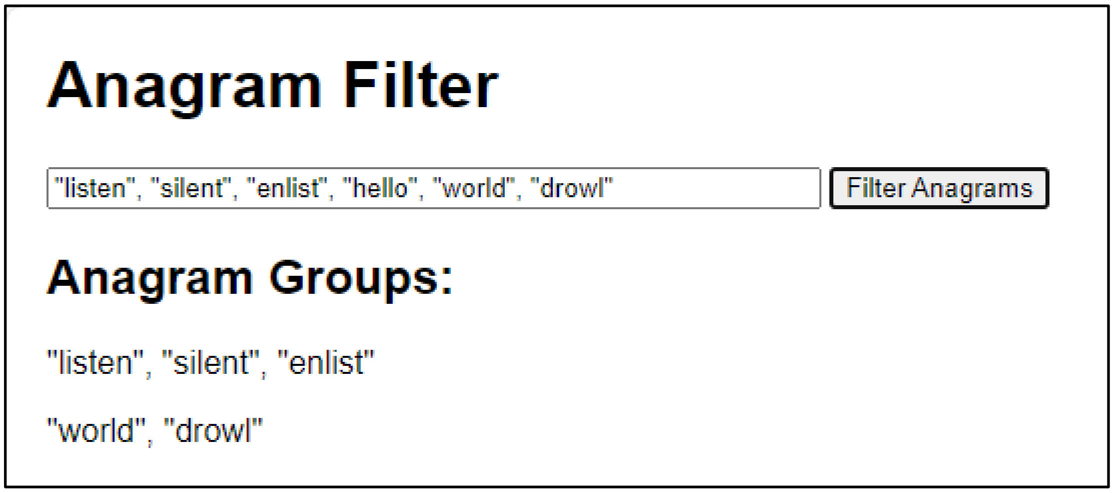
```
function filterAnagrams(words) {
    const anagrams = {};

    words.forEach(word => {
        const sortedWord = sortWord(word);
        if (anagrams[sortedWord]) {
            anagrams[sortedWord].push(word);
        } else {
            anagrams[sortedWord] = [word];
        }
    });

    const result = [];
    for (const key in anagrams) {
        if (anagrams[key].length > 1) {
            result.push(anagrams[key]);
        }
    }

    return result;
```

```
        }

        function filterAndDisplayAnagrams() {
            const input = document.getElementById('inputWords').value;
            const words = input.split(',').map(word => word.trim());
            const anagramGroups = filterAnagrams(words);

            const outputDiv = document.getElementById('output');
            outputDiv.innerHTML = '<h2>Anagram Groups:</h2>';
            if (anagramGroups.length > 0) {
                anagramGroups.forEach(group => {
                    outputDiv.innerHTML += `<p>${group.join(', ')}</p>`;
                });
            } else {
                outputDiv.innerHTML += '<p>No anagrams found.</p>';
            }
        }
    </script>
</body>
</html>
```

Output Screenshots :


Anagram Filter

"listen", "silent", "enlist", "hello", "world", "drowl"


Anagram Filter

"listen", "silent", "enlist", "hello", "world", "drowl"

Anagram Groups:

"listen", "silent", "enlist"

"world", "drowl"

4) Iterable keys

We’d like to get an array of map.keys() in a variable and then apply array-specific methods to it, e.g. .push.

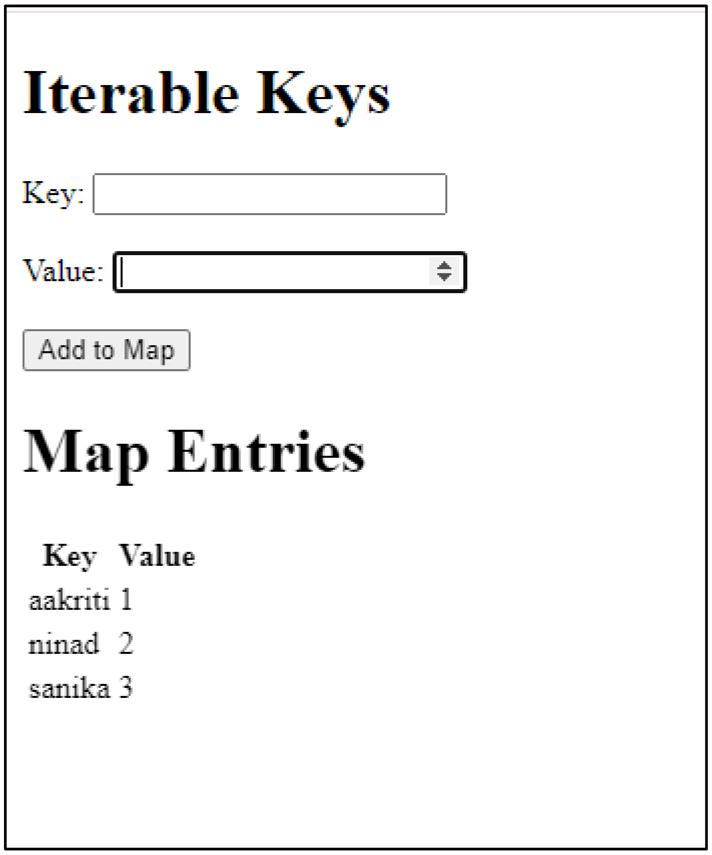
Code:

```
<!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>

    <h1>Iterable Keys</h1>

    <form id="map-form">

        <div class="input-group">

            <label for="key-input">Key:</label>

            <input type="text" id="key-input" required>

        </div>

        <br>

        <div class="input-group">

            <label for="value-input">Value:</label>

            <input type="number" id="value-input" required>

        </div>

        <br>

        <button type="submit">Add to Map</button>

    </form>

    <h1>Map Entries</h1>

    <table id="map-table">

        <thead>

            <tr>
            <th>Key</th>

            <th>Value</th>

        </tr>

    </thead>

    <tbody id="map-rows">

    </tbody>

</table>

<script>

    const map = new Map();

    document.getElementById('map-form').addEventListener('submit', (event) => {

        event.preventDefault();


        const key = document.getElementById('key-input').value;

        const value = parseInt(document.getElementById('value-input').value, 10);


        if (key && !isNaN(value)) {

            map.set(key, value);

            document.getElementById('key-input').value = '';

            document.getElementById('value-input').value = '';

        updateTable();
    } else {
        alert('Please enter valid key and value.');
    }
});

function updateTable() {
    const rowsContainer = document.getElementById('map-rows');
    rowsContainer.innerHTML = '';
    map.forEach((value, key) => {
        const row = document.createElement('tr');
        const keyCell = document.createElement('td');
        keyCell.textContent = key;
        const valueCell = document.createElement('td');
        valueCell.textContent = value;
        row.appendChild(keyCell);
        row.appendChild(valueCell);
        rowsContainer.appendChild(row);
    });
}
```

```
        document.getElementById('show-keys').addEventListener('click', () => {
          updateTable();
        });
      </script>
</body>
</html>
```

Output Screenshot:



Iterable Keys

Key:

Value:

Map Entries

Key	Value
aakriti	1
ninad	2
sanika	3

5) Fetch users from GitHub

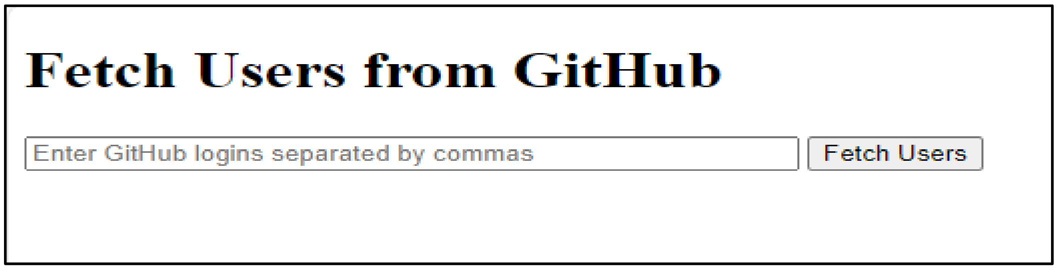
Create an async function getUsers(names), that gets an array of GitHub logins, fetches the users from GitHub and returns an array of GitHub users.

Code:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Fetch Users from GitHub</title>

</head>

<body>

    <h1>Fetch Users from GitHub</h1>

    <input type="text" id="githubLogins" placeholder="Enter GitHub logins separated by
commas" size="50">

    <button onclick="fetchGitHubUsers()">Fetch Users</button>

    <div id="githubOutput"></div>


    <script>

        async function getUsers(names) {

            const requests = names.map(name =>
fetch(`https://api.github.com/users/${name}`));
```

```
        const responses = await Promise.all(requests);

        const users = await Promise.all(responses.map(response => response.json()));

        return users;

    }


    async function fetchGitHubUsers() {

        const input = document.getElementById('githubLogins').value;

        const names = input.split(',').map(name => name.trim());

        const users = await getUsers(names);


        const outputDiv = document.getElementById('githubOutput');

        outputDiv.innerHTML = '<h2>GitHub Users:</h2>';

        users.forEach(user => {

            if (user.message !== 'Not Found') {

                outputDiv.innerHTML += `

                    <div>

                        <img src="${user.avatar_url}" alt="${user.login}" width="50" height="50">

                        <p><strong>${user.login}</strong></p>

                        <p>${user.name || ''}</p>
                        <p>${user.bio || ''}</p>
                    </div>
                    <hr>
                `;
            } else {
                outputDiv.innerHTML += `<p>User ${user.login} not found.</p><hr>`;
            }
        });
    }
    </script>
    </body>
    </html>
```

Output Screenshots :


Fetch Users from GitHub

Steps for execution:

1. Create HTML file
2. Write java script logic for the given question
3. Test and debug if required
4. Check console.

Conclusion:

Learned advanced javascript functions.