Batch: D-2	**Roll No.: 16010123325**
Experiment / assignment / tutorial No._2___	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Advanced JavaScript Concepts

AIM: Implementation of Advanced JavaScript Concepts.

Objective:

- Understand and implement key advanced JavaScript concepts.
- Learn how the JavaScript engine executes code (execution context, call stack).
- Explore modern ES6+ features like arrow functions, destructuring, and template literals.
- Apply asynchronous programming concepts using Promises and async/await.
- Use closures, callbacks, and higher-order functions effectively.
- Understand and utilize prototypal inheritance and object-oriented JavaScript.

Problem Definition:

JavaScript is a core language for web development, enabling dynamic and interactive user experiences. While many developers are familiar with its basics, mastering advanced JavaScript concepts is crucial for building scalable, performant, and maintainable applications.

This assignment aims to deepen understanding of advanced JavaScript features such as closures, promises, async/await, prototypes, higher-order functions, ES6+ features, and module patterns. Through

hands-on implementation, debugging, and real-world scenarios, students will develop the confidence to write clean, efficient, and modern JavaScript code..

Resources used:

https://javascript.info/

Expected OUTCOME of Experiment:

CO 4: Illustrate the concepts of various front-end, back-end web application development technologies & frameworks using different web development tools.

Books/ Journals/ Websites referred:

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

Pre Lab/ Prior Concepts:

- Basic JavaScript syntax and data types
- Functions: declaration, parameters, return values
- Arrow functions and template literals
- Array methods like map()
- Basic DOM methods (getElementById, createElement, appendChild)
- Event handling (onclick, addEventListener)
- Promises: resolve, reject, .then()

Methodology:

- Convert regular functions to arrow functions.
- Use arrow functions with array methods.
- Create and use callback functions.
- Write and handle Promises using setTimeout to simulate async behavior.
- Manipulate HTML using DOM methods.
- Use events to respond to clicks and keyboard input in real time.

Implementation Details:

1. Arrow Functions

Steps:

1. The function receives a number num.
2. It multiplies num * num.
3. Returns the result (the square).
4. square(11) returns 121.

Steps:

1. The function receives a string name.
2. It returns a string formatted as "Hello, <name>".
3. greet("Shreyans Tatiya") returns "Hello, Shreyans Tatiya".

```
//Problem 1.1 -Function Conversion

const square=(num)=>{
    return num * num;
}

console.log(square(11));


const greet=(name)=>{
    return `Hello, ${name} `;
}

console.log(greet("Shreyans Tatiya"));
```

Output-

```
121
Hello, Shreyans Tatiya


=== Code Execution Successful ===
```

Steps:

1. An array [2, 4, 6, 8] is given.
2. map() goes through each item in the array.
3. The arrow function doubles each item: num * 2.
4. Returns a new array: [4, 8, 12, 16]

```
//Problem 1.2 - Using Arrow Functions with Array Methods

let number=[2,4,6,8];

const doubled=number.map((num)=>{
    return num * 2;
})

console.log(doubled);
```

Output-

```
  node "/Users/shreya/Desktop/CE/TY/FSD/adv_js/arrow_func.js"
● shreya@Shreyas-MacBook-Air-2 FSD % node "/Users/shreya/Desktop/CE/TY/FSD/adv_js/arrow_func.js"
  [ 4, 8, 12, 16 ]
✧ shreya@Shreyas-MacBook-Air-2 FSD %
```

2. Callback Functions

Steps:

1. greetUser receives "Shreyans Tatiya" and greet function.
2. Inside greetUser, it calls greetCallback(name) → i.e., greet("Shreyans Tatiya").
3. greet() logs "Hello, Shreyans Tatiya!"

```
//Problem 2.1 - Execute a Callback
function greet(name){
   console.log(`Hello, ${name}!`);
}

function greetUser(name, greetCallback){
    greetCallback(name);
}

greetUser("Shreyans Tatiya", greet);
```

Output-

```
Hello, Shreyans Tatiya!

=== Code Execution Successful ===
```

Steps:

1. Takes an array (e.g., [1, 2, 3, 4]) and a callback (e.g., num * num).
2. Initializes an empty result array.
3. For each element in the array:
   - Applies the callback.
   - Pushes the result to result.
4. Prints the final array (e.g., [1, 4, 9, 16]).

```
// Problem 2.2 - Custom Array Processor
function processArray(arr, arrayCallback){
    let result = [];
    for(let i = 0; i < arr.length; i++){
        result.push(arrayCallback(arr[i]));
    }
    console.log(result);
}


processArray([1, 2, 3, 4], (num) =>{
    return num*num;
})
```

Output-

```
  node "/Users/shreya/Desktop/CE/TY/FSD/adv_js/callback.js"
● shreya@Shreyas-MacBook-Air-2 FSD % node "/Users/shreya/Desktop/CE/TY/FSD/adv_js/callback.js"
  [ 1, 4, 9, 16 ]
✧ shreya@Shreyas-MacBook-Air-2 FSD %
```

3. Promises

Steps:

1. Returns a Promise.
2. Inside the Promise:
   - A setTimeout waits 2 seconds.
   - Then calls resolve("Data loaded successfully!").
3. .then() waits for the Promise to finish.
4. After 2 seconds, logs "Data loaded successfully!".

```
// Problem 3.1 - Simple Promise Handling
function loadData(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const data = "Data loaded successfully!";
            resolve(data);
        }, 2000);
    });
}

let promise=loadData();

promise.then((res) => {
    console.log(res);
})
```

Output-

```
node "/Users/shreya/Desktop/CE/TY/FSD/adv_js/promises.js"
shreya@Shreyas-MacBook-Air-2 FSD % node "/Users/shreya/Desktop/CE/TY/FSD/adv_js/promises.js"
Data loaded successfully!
shreya@Shreyas-MacBook-Air-2 FSD %
```

Steps:

1. Function receives a URL.
2. Returns a Promise.
3. Inside the Promise:
   - A setTimeout delays for 3 seconds.
   - After that, resolves with "Data fetched from <url>".
4. .then() logs the result when the Promise resolves.

```
// Problem 3.2 - Simulate an API Call

function fakeFetch(url){
    return new Promise((resolve,reject)=>{
        setTimeout(()=>{
            resolve(`Data fetched from ${url}`);
        }, 3000);
    })
}

fakeFetch("https://api.example.com").then(console.log);
```

Output-

```
node "/Users/shreya/Desktop/CE/TY/FSD/adv_js/promises.js"
shreya@Shreyas-MacBook-Air-2 FSD % node "/Users/shreya/Desktop/CE/TY/FSD/adv_js/promises.js"
Data fetched from https://api.example.com
shreya@Shreyas-MacBook-Air-2 FSD %
```

4. DOM Manipulation

Steps:

1. Button is clicked → changeText() is called.
2. Grabs the <p> element by id="demo".
3. Updates its content to "Text updated!".

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JS</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <script src="script.js"></script>
    <h1 class="subheading">DOM MANIPULATION</h1>
    <h4 class="subheading">JavaScript</h4>
    <p id="demo">This is a paragraph.</p>

    <button id="btn" onclick="changeText()">Click Me!</button>

</body>
</html>
```

```javascript
const changeText =()=>{
    const textElement = document.getElementById('demo');
    textElement.innerHTML = "Text updated!";
}
```

Output-

Steps:

1. Button click calls addItem().
2. Gets <ul> by id="myList".
3. Creates a new <li> element.
4. Counts current number of s using list.children.length.
5. Sets text as "Item X", where X = count + 1.
6. Appends <li> to the list.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JS</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <ul id="myList"></ul>
<button onclick="addItem()">Add Item</button>

<script>
function addItem() {
    const list = document.getElementById("myList");
    const newItem = document.createElement("li");

    const itemNumber = list.children.length + 1;
    newItem.textContent = "Item " + itemNumber;

    list.appendChild(newItem);
}
</script>
</body>
</html>
```

Output-

5. Events

Steps:

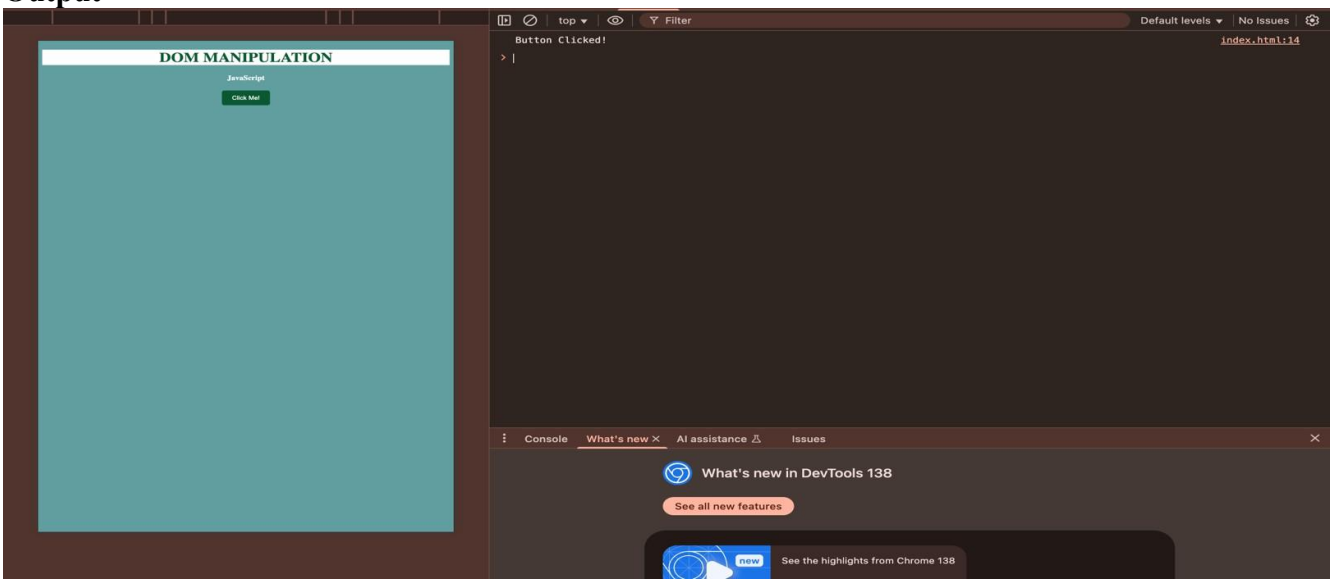
1. JS selects the button by id.
2. Adds a click event listener.
3. When clicked, executes a function.
4. That function logs "Button clicked!" to the console.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JS</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1 class="subheading">DOM MANIPULATION</h1>
    <h4 class="subheading">JavaScript</h4>


    <button id="btn" onclick="console.log('Button Clicked!')">Click Me!</button>

</body>
</html>
```

Output-



Steps:

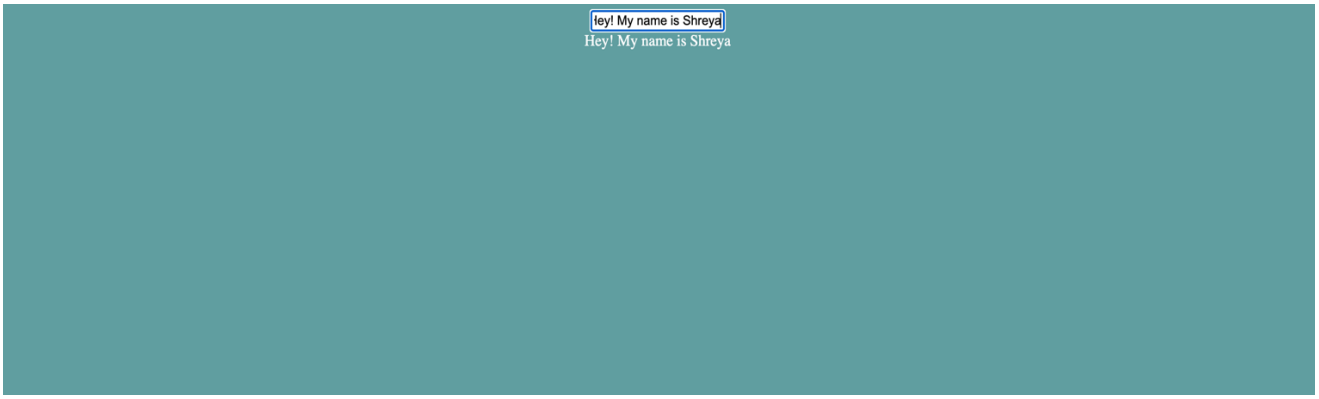
1. User types in the input field.
2. input event is triggered on each keystroke.
3. JS captures inputField.value.
4. Sets displayText.innerText to match the input in real time.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JS</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
   <input type="text" id="inputField" placeholder="Type something..." />
    <div id="displayText"></div>

    <script>
        const inputField = document.getElementById("inputField");
        const displayText = document.getElementById("displayText");

        inputField.addEventListener("input", function() {
            displayText.textContent = inputField.value;
        });
    </script>
</body>
</html>
```

Output-



Conclusion:

The above experiment highlights advanced Javascript functionalities like Events, Callbacks, DOM manipulation which are important for implementing real functionalities in applications.