Batch: A1	**Roll No.: 16010123012**
Experiment / assignment / tutorial No.: 02	
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:

Methodology:
Problem Statements
1. Arrow Functions
Problem 1.1 – Function Conversion
Convert the following traditional functions into arrow functions:
- A function `square(num)` that returns the square of a number.
- A function `greet(name)` that returns a string "Hello, <name>".
Problem 1.2 – Using Arrow Functions with Array Methods
Given an array of numbers:
let numbers = [2, 4, 6, 8];
Use the `map()` method with an arrow function to return a new array containing each number doubled.

2. Callback Functions
Problem 2.1 – Execute a Callback
Write a function `greetUser(name, callback)` that:
- Prints "Welcome, <name>!".
- Then invokes the `callback()` function passed as an argument.
Problem 2.2 – Custom Array Processor
Write a function `processArray(arr, callback)` that:
- Iterates through the array.
- Applies the callback to each element.
- Returns a new array of processed elements.
Example:

```
processArray([1, 2, 3], function(num) {
  return num * 10;
});
```

Expected output: [10, 20, 30]

3. Promises

Problem 3.1 – Simple Promise Handling
Create a `loadData()` function that:
- Returns a Promise which resolves after 2 seconds.
- The resolved value should be the string "Data Loaded".
- Use `.then()` to display the result in the console.
Problem 3.2 – Simulate an API Call
Implement a function `fakeFetch(url)` that:
- Returns a Promise.
- Uses `setTimeout()` to simulate a 3-second delay.
- Resolves with "Data fetched from <url>".
Test with:
fakeFetch("https://api.example.com").then(console.log);

4. DOM Manipulation
(A) Problem 4.1 – Change HTML Content
Given the following HTML structure:
This is a paragraph.
<button onclick="changeText()">Click Me</button>
Write the `changeText()` function so that when the button is clicked, the paragraph text changes to "Text updated!"
(B)Problem 4.2 – Dynamic List Items
Given:
<ul id="myList">
<button onclick="addItem()">Add Item</button>
Write the `addItem()` function so that each button click:
- Adds a new `<li>` to the list.
- The new item should be labeled `Item X` where X is the item number.

5. Events
(A) Problem 5.1 – Button Click Event
Create a button:
<button id="btnClick">Click Me</button>
Attach a JavaScript event listener to:
- Display "Button clicked!" in the console every time the button is clicked.
(B)Problem 5.2 – Keyboard Input Display
Create the following UI:
<input type="text" id="inputField" />

Write JavaScript to:
- Display the current text entered in the input field inside the .
- Update in real-time as the user types.

Implementation Details:

```
// Problem 1.1 – Function Conversion
let num = Math.round(Math.random() * 10);
let name = "Aaryan";
const square = (num) => num * num;
console.log(square(num));
const greet = (name) => `Hello, ${name}`;
console.log(greet(name));

// Problem 1.2 – Using Arrow Functions with Array Methods
let numbers = [2, 4, 6, 8];
let doubled = numbers.map((num) => num * 2);
console.log(doubled);
```

```
PS D:\KJSCE\BTech\TY\Sem V\MERN\Code> node arrow_functions.js
4
Hello, Aaryan
[ 4, 8, 12, 16 ]
```

```
// Problem 2.1 – Execute a Callback
function greetUser(name, callback) {
  console.log(`Welcome, ${name}!`);
  callback();
}

greetUser("Aaryan", function() {
  console.log("Callback executed!");
});

// Problem 2.2 – Custom Array Processor
function processArray(arr, callback) {
  let result = [];
  for(let i = 0; i < arr.length; i++) {
    result.push(callback(arr[i]));
  }
  return result;
}

processArray([1, 2, 3], function(num) {
  console.log(num * 10);
});
```

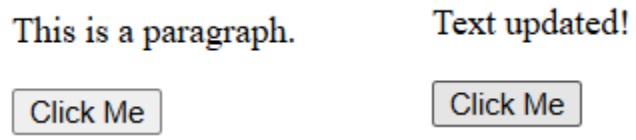
```
PS D:\KJSCE\BTech\TY\Sem V\MERN\Code> node callback_functions.js
Welcome, Aaryan!
Callback executed!
10
20
30
```

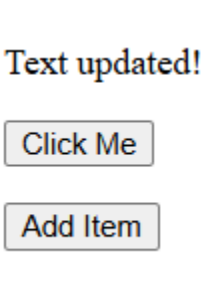
```
// Problem 3.1 – Simple Promise Handling
function loadData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Data Loaded");
    }, 2000);
  });
}

loadData().then((result) => {
  console.log(result);
});

// Problem 3.2 – Simulate an API Call
function fakeFetch(url){
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(`Data fetched from ${url}`);
    }, 3000);
  });
}

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

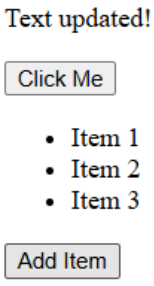
```
PS D:\KJSCE\BTech\TY\Sem V\MERN\Code> node promises.js
Data Loaded
Data fetched from https://api.example.com
```

```
<!-- Problem 4.1 – Change HTML Content -->
<!-- Problem 4.2 – Dynamic List Items -->

<!DOCTYPE html>
<html lang="en">
```

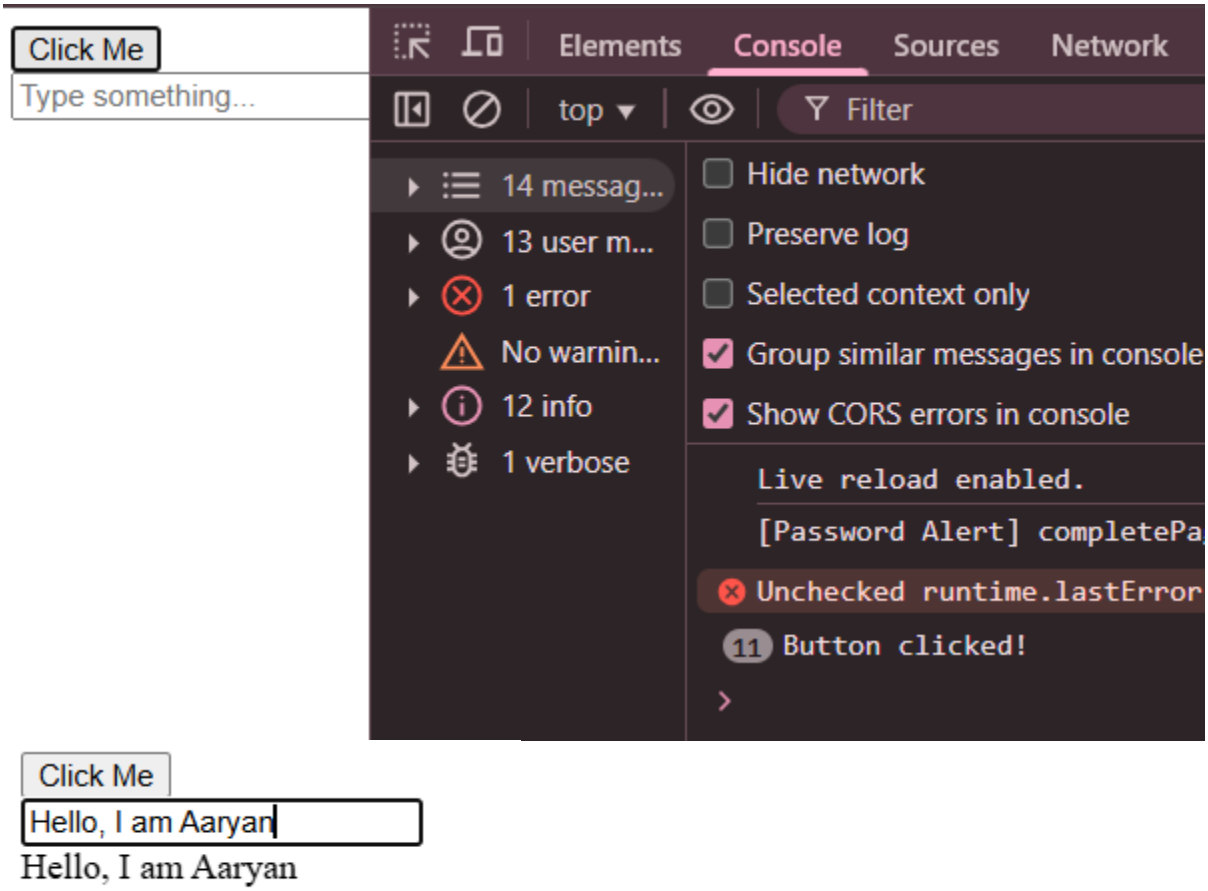
```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DOM Manipulation</title>
</head>
<body>
  <p id="demo">This is a paragraph.</p>
  <button onclick="changeText()">Click Me</button>

  <ul id="myList"></ul>
  <button onclick="addItem()">Add Item</button>

  <script>
    function changeText() {
      document.getElementById("demo").innerHTML = "Text updated!";
    }

    let itemCount = 0;
    function addItem() {
      itemCount++;
      const li = document.createElement("li");
      li.textContent = "Item " + itemCount;
      document.getElementById("myList").appendChild(li);
    }
  </script>
</body>
</html>
```







```
<!-- Problem 5.1 – Button Click Event -->
<!-- Problem 5.2 – Keyboard Input Display -->

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Events</title>
</head>
<body>
  <button id="btnClick">Click Me</button>
  <input type="text" id="inputField" placeholder="Type something..." />
  <div id="displayText"></div>

  <script>
    document.getElementById("btnClick").addEventListener("click", function () {
      console.log("Button clicked!");
    });

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



Conclusion:

I have completed this experiment. Through this experiment, I successfully implemented and explored various advanced JavaScript concepts including arrow functions, callbacks, promises, async/await, DOM manipulation, and event handling. This helped deepen my understanding of modern ES6+ features, asynchronous programming, and interactive UI development.