LAB ASSIGNMENT-4

1. Explain Array methods in JavaScript. Specifically, demonstrate how push (), pop(), shift(), and unshift() modify an array?

   ANS: **Array Methods in JavaScript**

   Arrays in JavaScript are used to store multiple values in a single variable. JavaScript provides built-in **array methods** to add, remove, and manipulate elements easily.

   Below are commonly used array methods, with special focus on **push()**, **pop()**, **shift()**, and **unshift()**.

---

1. push()

- Adds one or more elements to the **end** of an array.
- Returns the new length of the array.

Example:

```
let fruits = ["Apple", "Banana"];
fruits.push("Orange");

console.log(fruits);
// Output: ["Apple", "Banana", "Orange"]
```

---

2. pop()

- Removes the **last element** from an array.
- Returns the removed element.

Example:

```
let fruits = ["Apple", "Banana", "Orange"];
fruits.pop();

console.log(fruits);
// Output: ["Apple", "Banana"]
```

---

3. shift()

- Removes the **first element** of an array.
- Shifts remaining elements to lower indexes.
- Returns the removed element.

Example:

```
let fruits = ["Apple", "Banana", "Orange"];
fruits.shift();

console.log(fruits);
// Output: ["Banana", "Orange"]
```

---

4. unshift()

- Adds one or more elements to the **beginning** of an array.
- Shifts existing elements to higher indexes.
- Returns the new length of the array.

Example:

```
let fruits = ["Banana", "Orange"];
fruits.unshift("Apple");

console.log(fruits);
// Output: ["Apple", "Banana", "Orange"]
```

Summary Table

Method	**Action**	**Position Affected**
push()	Adds element(s)	End of array
pop()	Removes element	End of array
shift()	Removes element	Beginning of array
unshift()	Adds element(s)	Beginning of array

2.what are promises in JavaScript , and how do async/awaited simplify working with asynchronous code?

ANS: **Promises in JavaScript**

A **Promise** in JavaScript is an object used to handle **asynchronous operations** (tasks that take time to complete, such as fetching data from a server). A promise represents a value that may be available **now, later, or never**.

A Promise has **three states**:

1. **Pending** – The operation is still in progress

2. **Fulfilled** – The operation completed successfully
3. **Rejected** – The operation failed

Example of a Promise:

```
let promise = new Promise((resolve, reject) => {
  let success = true;

  if (success) {
    resolve("Data fetched successfully");
  } else {
    reject("Error fetching data");
  }
});
```

Promises are usually handled using .then() and .catch():

```
promise
  .then(result => console.log(result))
  .catch(error => console.log(error));
```

Async/Await in JavaScript

async and await are modern JavaScript features that make working with promises **simpler and more readable**.

- **async** is used before a function to make it return a promise.
- **await** pauses the execution of the function until the promise is resolved or rejected.

Example using async/await:

```
async function fetchData() {
  try {
    let result = await promise;
    console.log(result);
  } catch (error) {
    console.log(error);
  }
}
```

How async/await Simplify Asynchronous Code

1. **Readable code** – Looks like synchronous code, making it easier to understand.
2. **Better error handling** – Uses try...catch instead of .then() and .catch().

3. **Less chaining** – Avoids long chains of .then() callbacks.
4. **Easier debugging** – Code flow is clearer.

Comparison: Promises vs Async/Await

Feature	**Promises (then/catch)**	**Async/Await**
Syntax	More complex	Cleaner and simpler
Readability	Moderate	High
Error handling	.catch()	try...catch
Code structure	Chained	Linear

---

4. **Describe the concept of Event Delegation and explain the use of addEventListener.**

ANS: **Event Delegation**

Event Delegation is a technique in JavaScript where a **single event listener** is attached to a **parent element** instead of attaching separate listeners to multiple child elements. This works because of **event bubbling**, where an event triggered on a child element propagates (bubbles) up to its parent elements.

Why use Event Delegation?

- Improves performance by reducing the number of event listeners.
- Works well for dynamically added elements.
- Makes code cleaner and easier to manage.

Example of Event Delegation:

```
<ul id="list">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
```

```
</ul>
document.getElementById("list").addEventListener("click", function(event) {
  if (event.target.tagName === "LI") {
    console.log(event.target.innerText);
  }
});
```

Here, clicking any <li> is handled by the single event listener on the <ul>.

---

addEventListener

addEventListener() is a JavaScript method used to attach an event handler to an element. It allows multiple event handlers to be added to the same element without overwriting existing ones.

Syntax:

```
element.addEventListener(event, function, useCapture);
```

- **event** – Type of event (e.g., click, mouseover, keydown)
- **function** – Function to execute when the event occurs
- **useCapture** (optional) – Boolean indicating event capturing or bubbling phase

Example:

```
document.getElementById("btn").addEventListener("click", function() {
  alert("Button clicked!");
});
```

Relationship Between Event Delegation and addEventListener

Event Delegation relies on addEventListener to attach a single listener to a parent element. When a child element triggers an event, addEventListener catches it during the bubbling phase and identifies the actual target using event.target.

Advantages of Event Delegation

- Reduces memory usage
- Handles dynamic elements efficiently
- Simplifies event management