Day 3: DOM Manipulation in JavaScript

1. Understanding the Document Object Model (DOM)

The **DOM (Document Object Model)** represents a webpage as a tree structure where each element (tags, attributes, text) is a node.

DOM Tree Structure Example

Consider the following HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>DOM Example</title>
</head>
<body>
    <h1 id="heading">Hello, World!</h1>
    <p class="content">Welcome to JavaScript DOM.</p>
    <button id="changeText">Click Me</button>
</body>
</html>
```

This HTML gets represented as a tree where <html> is the root, containing <head> and <body>, which further contain elements.

---

2. Selecting Elements

JavaScript provides various methods to **select** elements.

Using getElementById

Selects a single element by its ID.

```
let heading = document.getElementById("heading");
console.log(heading.innerHTML);  // Output: Hello, World!
```

Using getElementsByClassName

Selects multiple elements with the same class.

```
let paragraphs = document.getElementsByClassName("content");
console.log(paragraphs[0].textContent);  // Output: Welcome to JavaScript DOM.
```

Using querySelector and querySelectorAll

- querySelector(): Selects the first matching element.
- querySelectorAll(): Selects all matching elements.

```
let firstParagraph = document.querySelector(".content");
let allParagraphs = document.querySelectorAll(".content");
console.log(firstParagraph.innerHTML);  // Output: Welcome to JavaScript DOM.
console.log(allParagraphs.length);      // Output: Number of matching elements
```

---

3. Manipulating Elements

Once selected, we can **modify** elements.

Changing Content

```
heading.innerHTML = "Welcome to DOM Manipulation!";
```

Modifying Attributes

```
heading.style.color = "blue";
heading.classList.add("highlight");
```

Creating and Removing Elements

```
let newElement = document.createElement("p");
newElement.textContent = "This is a new paragraph.";
document.body.appendChild(newElement);  // Adds to the body


document.body.removeChild(newElement);  // Removes from the body
```

4. Event Handling

Adding Event Listeners

```
let button = document.getElementById("changeText");
button.addEventListener("click", function() {
    heading.textContent = "Button Clicked!";
});
```

Common Events

- click → When an element is clicked.
- mouseover → When the mouse hovers over an element.
- mouseout → When the mouse leaves an element.
- keydown → When a key is pressed.
- keyup → When a key is released.

Example: Mouse Events

```
heading.addEventListener("mouseover", function() {
    heading.style.color = "red";
});
heading.addEventListener("mouseout", function() {
    heading.style.color = "black";
});
```

Example: Keyboard Events

```
document.addEventListener("keydown", function(event) {
    console.log("Key Pressed: " + event.key);
});
```

Project-Based Example: Interactive To-Do List App

Project Overview

We will create a **simple To-Do List app** using JavaScript DOM manipulation and event handling. The app will allow users to:

- **Add** tasks to a list.
- **Remove** tasks from the list.
- **Mark tasks as completed**.

Step 1: HTML Structure

Create a file index.html with the following structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>To-Do List</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            background-color: #f5f5f5;
        }
        #taskList {
            list-style-type: none;
            padding: 0;
        }
        li {
            background: white;
            padding: 10px;
```

```
            margin: 5px;
            display: flex;
            justify-content: space-between;
            align-items: center;
            border-radius: 5px;
            box-shadow: 0px 0px 5px gray;
        }
        .completed {
            text-decoration: line-through;
            color: gray;
        }
    </style>
</head>
<body>

    <h1>To-Do List</h1>
    <input type="text" id="taskInput" placeholder="Enter a task">
    <button id="addTask">Add Task</button>

    <ul id="taskList"></ul>

    <script src="script.js"></script>

</body>
</html>
```

Step 2: JavaScript (script.js)

```
// Select elements
let taskInput = document.getElementById("taskInput");
let addTaskButton = document.getElementById("addTask");
let taskList = document.getElementById("taskList");

// Function to add a new task
function addTask() {
    let taskText = taskInput.value.trim();

    if (taskText === "") {
        alert("Please enter a task!");
        return;
    }

    // Create a new list item
    let li = document.createElement("li");
    li.textContent = taskText;

    // Create complete button
    let completeButton = document.createElement("button");
    completeButton.textContent = "✔";
    completeButton.style.marginLeft = "10px";
    completeButton.addEventListener("click", function() {
        li.classList.toggle("completed");
    });

    // Create delete button
```

```
    let deleteButton = document.createElement("button");
    deleteButton.textContent = "❌";
    deleteButton.style.marginLeft = "10px";
    deleteButton.addEventListener("click", function() {
        taskList.removeChild(li);
    });

    // Append buttons to the list item
    li.appendChild(completeButton);
    li.appendChild(deleteButton);

    // Append list item to the task list
    taskList.appendChild(li);

    // Clear input field
    taskInput.value = "";
}

// Event listener for the add button
addTaskButton.addEventListener("click", addTask);

// Allow adding tasks by pressing "Enter"
taskInput.addEventListener("keypress", function(event) {
    if (event.key === "Enter") {
        addTask();
    }
});
```

Step 3: Explanation of Concepts Used

1. Selecting Elements

- document.getElementById("taskInput") → Selects the input field.
- document.getElementById("addTask") → Selects the "Add Task" button.
- document.getElementById("taskList") → Selects the task list <ul>.

2. Adding Event Listeners

- addTaskButton.addEventListener("click", addTask); → Calls the addTask function when the "Add Task" button is clicked.
- taskInput.addEventListener("keypress", function(event) {...}) → Calls addTask when the **Enter key** is pressed.

3. Creating and Appending Elements

- document.createElement("li") → Creates a new <li> element.
- document.createElement("button") → Creates "✔" and "✖" buttons for marking tasks as complete and deleting them.
- taskList.appendChild(li) → Adds the new task to the <ul>.

4. Handling Click Events

- Clicking "✔" toggles the completed class to **strike through** the text.
- Clicking "✖" removes the task from the list using taskList.removeChild(li);.

Final Output

When you run this project:

1. You can type a task and press **"Add Task"** or hit **Enter**.
2. The task will appear in the list.
3. Clicking "✔" will mark the task as **completed**.
4. Clicking "✖" will **delete** the task.

Enhancements and Further Learning

- Store tasks in **local storage** so they persist after refreshing.
- Add **animations** when tasks are added or removed.
- Use **Drag and Drop** to rearrange tasks.

This project-based example **reinforces DOM manipulation concepts** while building something practical.

The symbols ✔ **(Check Mark) and** ✖ **(Cross Mark)** are **Unicode characters (Emoji)**. You can use these directly in JavaScript or HTML because modern browsers support Unicode characters.

Where Did I Get These Symbols?

I used standard Unicode symbols from the **Emoji & Symbol Character Set**. You can find them in:

- **Emojipedia**: https://emojipedia.org/
- **Unicode Character Table**: https://unicode-table.com/
- **Copy-Paste Emoji Websites**

Alternative Approach: Using Unicode Codes

Instead of directly using ✔ and ✖, you can use their Unicode escape sequences:

deleteButton.textContent = "\u274C"; // ✖ Cross Mark

completeButton.textContent = "\u2705"; // ✔ Check Mark

- \u274C → ✖ (Cross Mark)

- \u2705 → ✔ (Check Mark)

Alternative Approach: Using Icons (Font Awesome)

If you prefer using **icons**, you can use **Font Awesome**:

html

 <!-- Check Mark -->

 <!-- Cross Mark -->

But for a **pure JavaScript approach**, Unicode characters are **simple and effective**.