Full Stack Development – Lab Programs
(AD2604-1)

Program 4: Interactive DOM Manipulation and Event Handling in JavaScript

Aim: To understand and implement DOM manipulation and event handling in JavaScript by dynamically updating HTML elements based on user input and interactions.

Explanation:

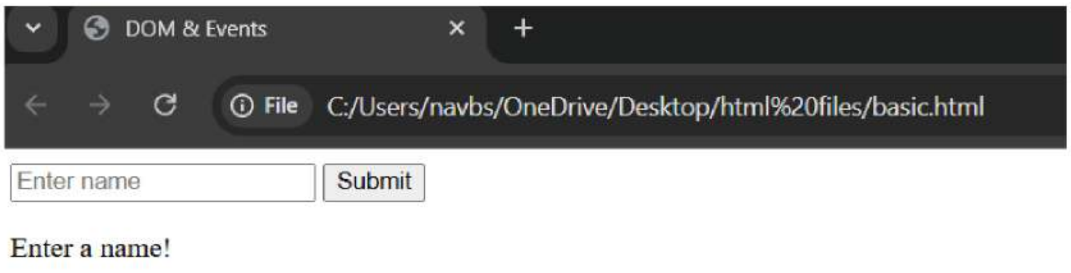
- DOM (Document Object Model) is like a tree structure of a webpage.
- When a browser loads a webpage, it creates a structured version of the HTML.
- The DOM allows JavaScript to dynamically access, modify, and interact with webpage content, structure, and styles, making websites interactive and responsive.
- This program demonstrates how JavaScript interacts with the DOM, listens for user events like typing and clicking, and dynamically updates webpage content.
- The program will create an input box, a button, and a message display. The user can type their name and click the button to see a greeting.
- When the page fully loads, a message "Page Loaded!" is logged in the console.
- The script selects three elements:
  - The input box where the user types their name.
  - The button that the user clicks to submit their name.
  - The paragraph (<p>) where the greeting message will be displayed.
- When the button is clicked:
  - If the user has typed a name, it displays "Hello, [name]!".
  - If the input is empty, it shows "Enter a name!".
- As the user types in the input box:
  - The text updates instantly and shows "Typing: [name]" in real-time.
- Execution order when the page loads:
  - The page loads, and "Page Loaded!" is logged in the console.
  - The user starts typing, and the message updates dynamically.
  - The user clicks the button, and a greeting is displayed based on the input.

Code:

```
<html lang="en">
<head>
  <title>DOM & Events</title> <!-- Page title displayed on the browser tab -->
</head>
<body>
  <input id="name" placeholder="Enter name">  <!-- Creates a text box where user can type -->
  <button id="btn">Submit</button>          <!-- Creates a button labeled "Submit" -->
  <p id="msg"></p>                          <!-- Creates an empty paragraph -->
  <script>
    // Waits for the entire webpage (HTML) to load before running JavaScript
```

```
        document.addEventListener("DOMContentLoaded", () => console.log("Page Loaded!"));
        // Find the input field, button, and message display area from the page
        const nameInput = document.getElementById("name");  // Selects the text input
        const btn = document.getElementById("btn");         // Selects the button
        const msg = document.getElementById("msg");  // Selects the paragraph where messages appear
        // Listens for a click event on the submit button
        btn.addEventListener("click", () => {
          // Checks if the user has typed something in the input box
          // If the input is not empty, show "Hello, [name]!"
          // Otherwise, ask the user to enter a name
          msg.textContent = nameInput.value ? `Hello, ${nameInput.value}!` : "Enter a name!";
        });
        // Listens for a key press event inside the text input box
        nameInput.addEventListener("keyup", () => {
          // Updates the <p> message in real-time to show what the user is typing
          msg.textContent = `Typing: ${nameInput.value}`;
        });
    </script>
</body>
</html>
```

Output:



Enter a name!

Program 5: Understanding Mutation Observers and JavaScript Async Execution

Aim: To demonstrate the use of Mutation Observers to detect DOM changes and explore the execution order of synchronous code, microtasks, and macrotasks in JavaScript's event loop.

Explanation:

- This Program Demonstrates Mutation & Async in JavaScript. Mutation means a change in the webpage's content or structure. JavaScript handles tasks asynchronously using microtasks and macrotasks.

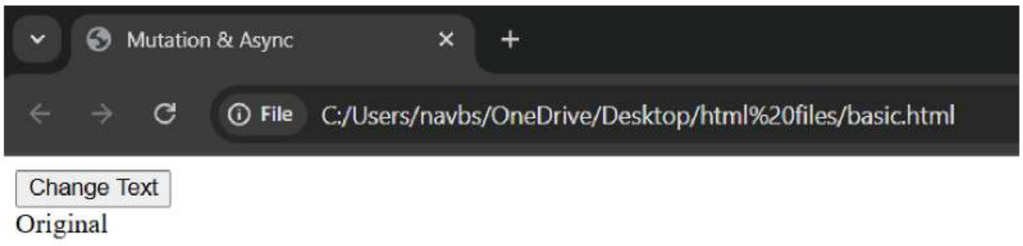
- The webpage has a button and a text box (div). When the button is clicked, the text inside the div changes.
- A MutationObserver is created, a built-in JavaScript object (class) used to watch for changes in the DOM. If the text changes, it logs "Mutation detected" in the console.
- JavaScript tasks are categorized into microtasks and macrotasks based on priority and execution timing in the event loop.
- When the button is clicked:
    - The text inside the div changes to "Updated!".
    - The mutation observer detects the change and logs "Mutation detected: Updated!".
    - A microtask (Promise) is added, which logs "Microtask: Promise".
    - A macrotask (setTimeout) is added, which logs "Macrotask: setTimeout".
- When the script starts running:
    - It logs "Script starts" first because it is normal synchronous code.
    - A microtask (Promise) logs "Microtask: First Promise" before any macrotask.
    - A macrotask (setTimeout) logs "Macrotask: First setTimeout" after microtasks.
- Execution order when the button is clicked:
    - "Mutation detected: Updated!" (MutationObserver detects change).
    - "Microtask: Promise" (Runs before macrotasks).
    - "Macrotask: setTimeout" (Runs after microtasks).
- This program demonstrates how JavaScript handles events, detects changes in the webpage, and executes tasks in a specific order using microtasks and macrotasks.

Code:

```
<html lang="en">
<head>
  <title>Mutation & Async</title> <!-- Page title displayed on the browser tab -->
</head>
<body>
  <button id="change">Change Text</button> <!-- Creates a button labeled "Change Text" -->
  <div id="content">Original</div> <!-- Creates a div (box) with the text "Original" inside -->
  <script>
    // Finds the <div> with id="content"
    const content = document.getElementById("content");
    // Creates a MutationObserver, which watches for changes in the <div>
    const observer = new MutationObserver(() =>
      console.log("Mutation detected:", content.textContent)
    );
    // Starts watching the <div> for text changes.
    observer.observe(content, { childList: true });
    // Add a click event listener to the button
    document.getElementById("change").addEventListener("click", () => {
      content.textContent = "Updated!"; // Change the text inside the content div
      // A microtask is created using a resolved Promise.
```

```
        // Microtasks run immediately after the current JavaScript code finishes.
        Promise.resolve().then(() => console.log("Microtask: Promise"));
        // A macrotask (setTimeout) is scheduled with a 0ms delay.
        // Even with 0ms, macrotasks always run after microtasks.
        setTimeout(() => console.log("Macrotask: setTimeout"), 0);
      });
      // Logs "Script starts" immediately when the script begins
      console.log("Script starts");
      // Even when JavaScript is already executing tasks, new tasks can be added dynamically
      // Another microtask is added to run after the script finishes but before macrotasks
      Promise.resolve().then(() => console.log("Microtask: First Promise"));
      // Another macrotask (setTimeout with 0ms delay) is scheduled
      setTimeout(() => console.log("Macrotask: First setTimeout"), 0);
    </script>
</body>
</html>
```

Output:



Original