

Q.4.1 Demonstrate DOM manipulation using getElementById(), querySelector(), and innerHTML

| Introduction

DOM manipulation allows JavaScript to access and modify HTML elements dynamically.

Three commonly used methods are:

```
D: > BCA_A1 > Ayush_Kumar_Dubey > Untitled-1.html > html
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <title>DOM Manipulation Demo</title>
5
6   <style>
7       body {
8           font-family: Arial, sans-serif;
9           padding: 20px;
10          background: #f1f5f9;
11
12          /* Centering content */
13          display: flex;
14          justify-content: center;
15          align-items: center;
16          height: 100vh;
17          margin: 0;
18      }
19
20     .box {
21         background: white;
22         padding: 20px;
23         width: 350px;
24         border-radius: 8px;
25         box-shadow: 0px 0px 8px rgba(0,0,0,0.2);
26         text-align: center; /* Center text and buttons */
27     }
28
29     button {
30         padding: 10px 16px;
```

* `getElementById()` → Selects an element by its ID

* `querySelector()` → Selects the first matching CSS selector

* `innerHTML` → Changes or inserts HTML content inside an element

These tools make web pages interactive and responsive.

Q.4.1 Demonstrate DOM manipulation using getElementById(), querySelector(), and innerHTML

Explanation of the Methods

The screenshot shows a code editor interface with the following details:

- File:** Untitled-1.html
- Content:**

```
button {
    padding: 10px 16px;
    margin-top: 10px;
    cursor: pointer;
    background: #0066cc;
    color: white;
    border: none;
    border-radius: 5px;
}

button:hover {
    background: #004d99;
}

</style>
</head>

<body>
    <div class="box">
        <h2>DOM Manipulation Example</h2>
        <p id="textArea">This text will change when you press a button.</p>
        <p class="message">Waiting for user action...</p>
        <button onclick="changeText()">Change Text</button>
        <br>
        <button onclick="updateMessage()">Update Message</button>
    </div>
</body>
```
- RUN AND DEBUG:** Run and Debug button is highlighted.
- BREAKPOINTS:** Caught Exceptions and Uncaught Exceptions are listed.
- EVENT LISTENER BREAKPOINTS:** None listed.
- Bottom Bar:** Includes icons for search, file operations, browser tabs, and system status.

A. getElementById()

What it does:

Selects an element by its id attribute
Example:

```
let heading = document.getElementById("title");
```

Actions performed:

Changes text color

Replaces the title text using innerHTML

What it does:

Selects the first element matching a CSS selector
Example:

```
let message = document.querySelector(".msg");
```

Actions performed:

Makes text bold

Changes its displayed message

Q.4.1 Demonstrate DOM manipulation using getElementById(), querySelector(), and innerHTML

Explanation of the Methods

The screenshot shows a code editor interface with a dark theme. The top bar includes 'edit Selection View Go Run ...' and a search bar. The left sidebar has 'RUN AND DEBUG' and 'Run and Debug' selected. A message says 'To customize Run and Debug, open a folder and create a launch.json file.' The main area displays the following code:

```
D: > BCA_A1 > Ayush_Kumar_Dubey > Untitled-1.html > html
59  <script>
60      // Using getElementById()
61      function changeText() {
62          let para = document.getElementById("textArea");
63          para.innerHTML = "Text changed using getElementById() and innerHTML!";
64      }
65
66      // Using querySelector()
67      function updateMessage() {
68          let msg = document.querySelector(".message");
69          msg.innerHTML = "querySelector() updated this message successfully.";
70      }
71  </script>
72
73  </body>
74  </html>
```

The bottom status bar shows 'Ln 74, Col 8 Spaces: 4 UTF-8 CRLF {} HTML'. The taskbar at the bottom includes icons for File, Search, Task View, Start, Taskbar settings, and various pinned applications.

C. innerHTML

What it does:

Changes or insert

document.getElementById("output").innerHTML =
<p>This is dynamic content added using innerHTML.</p>;

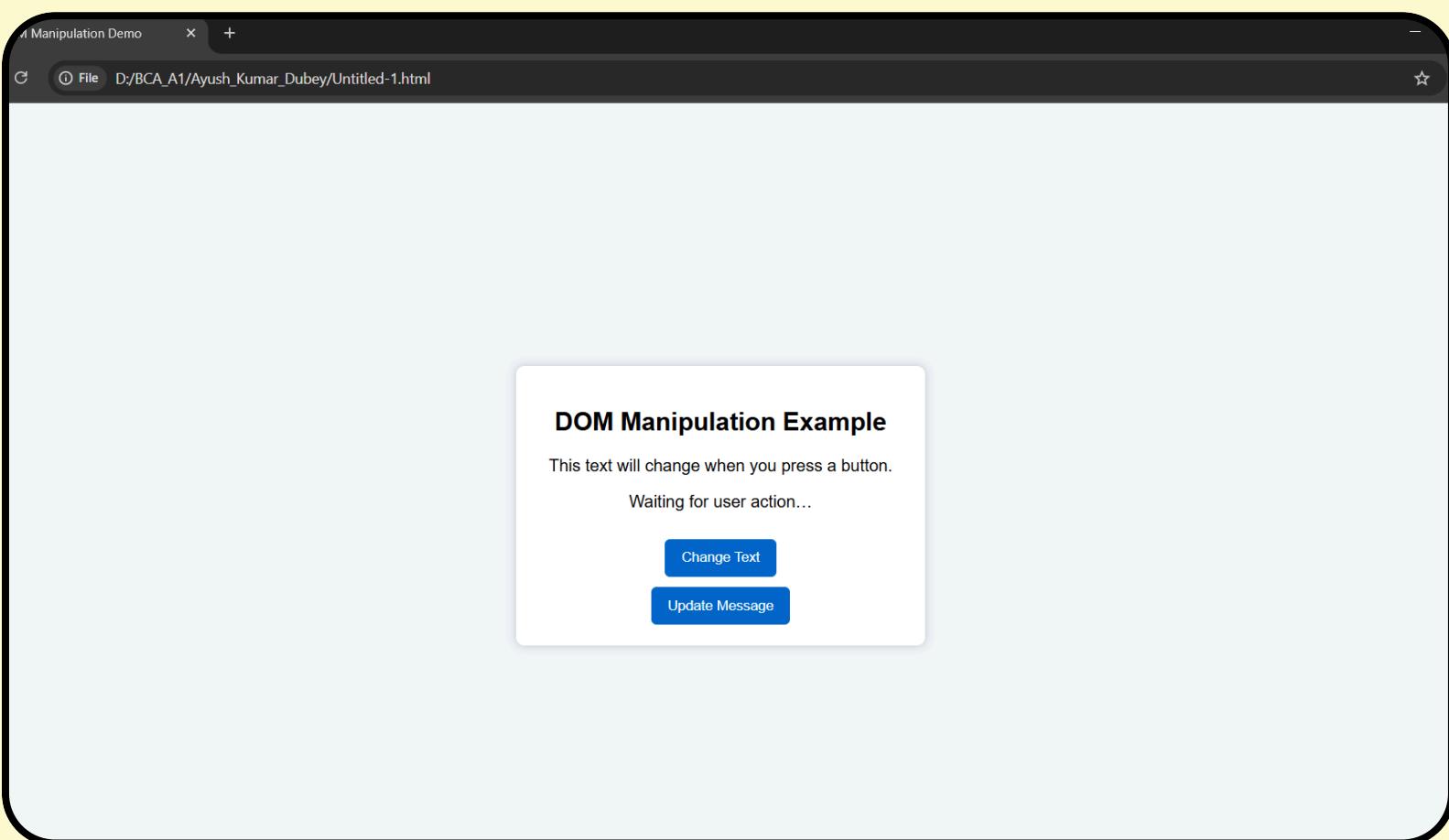
Actions performed:

Inserts a new

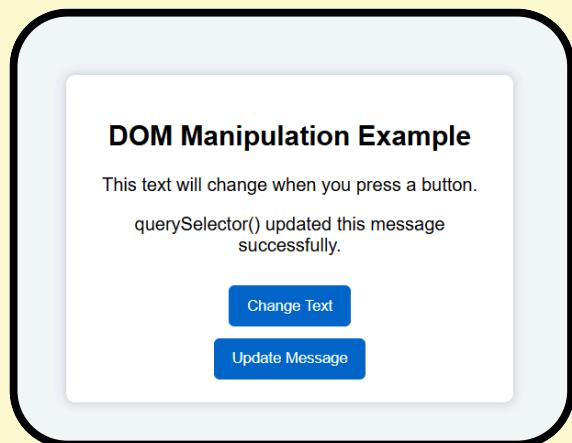
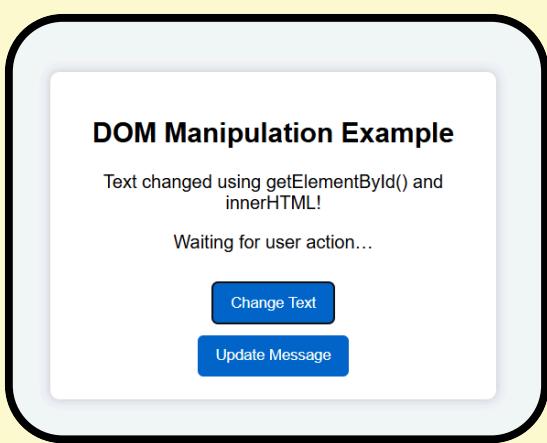
element inside the output container

Q.4.1 Demonstrate DOM manipulation using getElementById(), querySelector(), and innerHTML

OUTPUT



EFFECT AFTER USING BUTTON



Conclusion

Using getElementById(), querySelector(), and innerHTML, JavaScript can easily select, modify, and inject HTML elements. These methods are essential for building interactive web pages, updating content dynamically, and improving user experience.

Q.4.2 Show or hide HTML elements using JavaScript and toggle CSS classes dynamically

Introduction

JavaScript is commonly used to control the visibility of HTML elements and modify their appearance. Two powerful techniques include:



The screenshot shows a code editor interface with a sidebar containing icons for search, outline, timeline, and other development tools. The main area displays the content of a file named QUES4.2.html. The code includes HTML structure, CSS styles, and JavaScript logic to manage element visibility and class toggling.

```
<!DOCTYPE html>
<html>
<head>
    <title>Show/Hide and Toggle Class Example</title>
    <style>
        /* Base style for the text box */
        .box {
            padding: 15px;
            border: 2px solid #333;
            width: 320px;
            margin-top: 15px;
            transition: 0.3s;
        }
        /* Hides the element */
        .hidden {
            display: none;
        }
        /* Highlight style toggled with JavaScript */
        .highlight {
            background-color: yellow;
            border-color: orange;
            color: black;
        }
    </style>
</head>
<body>
    <center>
        <h2>Show / Hide Element & Toggle </h2>
        <button id="toggleBtn">Show / Hide Text</button>
        <button id="colorBtn">Toggle Highlight</button>
        <input id="textBox" class="box" type="text" value="Hello, World!">
    </center>
</body>

```

Changing style properties (like display) →

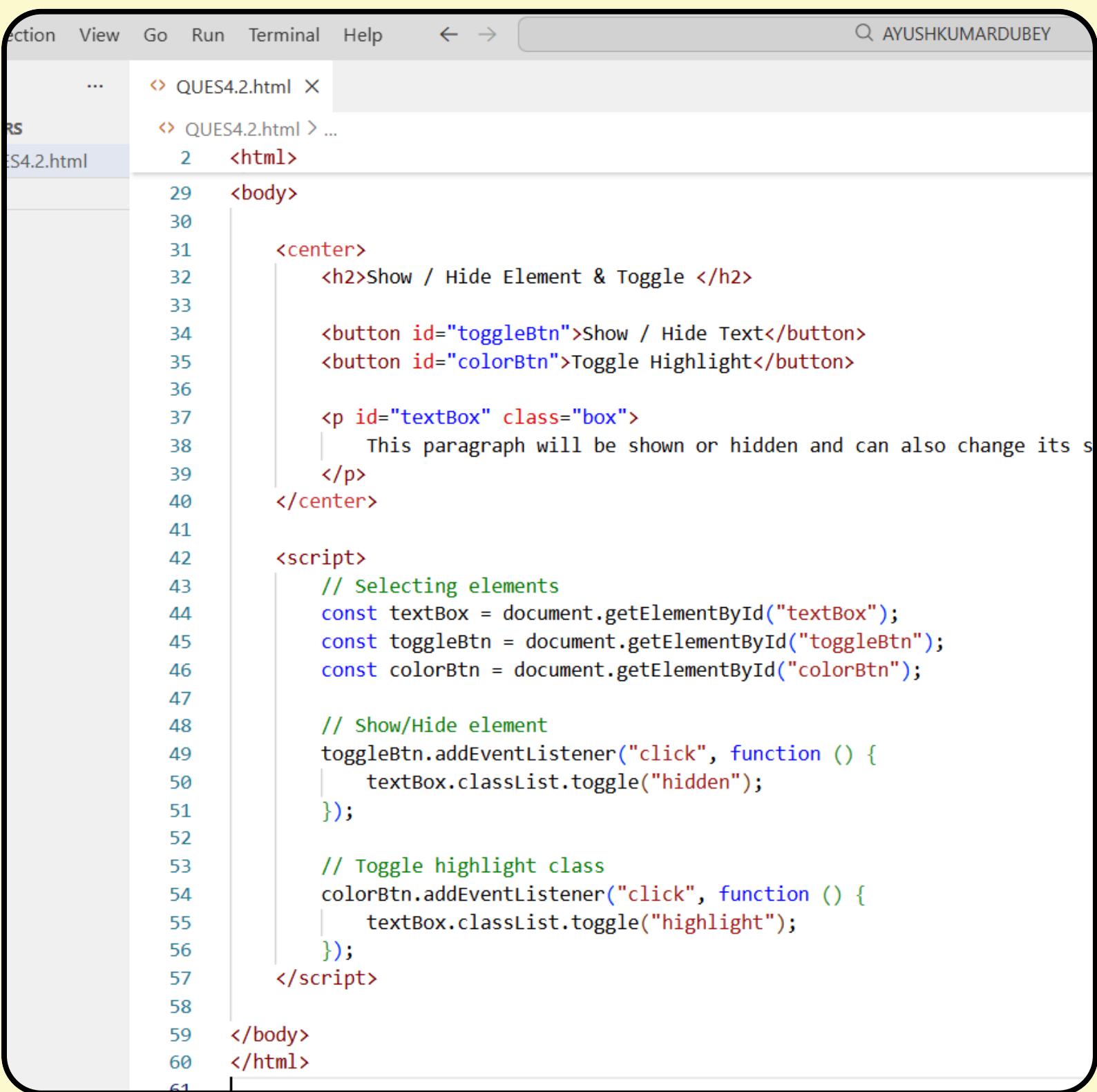
Toggling CSS classes →

hides or shows content

applies or removes predefined styles dynamically

Q.4.2 Show or hide HTML elements using JavaScript and toggle CSS classes dynamically

These features are essential for building interactive UI components such as dropdown menus, pop-ups, alerts, and navigation bars.



The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files named "QUES4.2.html" and "QUES4.2.html".
- Search Bar:** Contains the text "AYUSHKUMARDUBEY".
- Code Area:** Displays the following code:

```
<html>
  <body>
    <center>
      <h2>Show / Hide Element & Toggle </h2>
      <button id="toggleBtn">Show / Hide Text</button>
      <button id="colorBtn">Toggle Highlight</button>

      <p id="textBox" class="box">
        This paragraph will be shown or hidden and can also change its color.
      </p>
    </center>

    <script>
      // Selecting elements
      const textBox = document.getElementById("textBox");
      const toggleBtn = document.getElementById("toggleBtn");
      const colorBtn = document.getElementById("colorBtn");

      // Show/Hide element
      toggleBtn.addEventListener("click", function () {
        textBox.classList.toggle("hidden");
      });

      // Toggle highlight class
      colorBtn.addEventListener("click", function () {
        textBox.classList.toggle("highlight");
      });
    </script>
  </body>
</html>
```

The code demonstrates how to use JavaScript to manipulate HTML elements. It includes two buttons: "Show / Hide Text" and "Toggle Highlight". The "Show / Hide Text" button uses the `classList.toggle` method to switch between the "hidden" and "visible" states of the paragraph. The "Toggle Highlight" button uses the same method to switch between the "highlight" and "normal" states of the paragraph, which is visually represented by a color change.

Q.4.2 Show or hide HTML elements using JavaScript and toggle CSS classes dynamically

OUTPUT

Show / Hide Element & Toggle

Show / Hide Text Toggle Highlight

This paragraph will be shown or hidden and can also change its style dynamically.

Add A Subheading NO TEXT BOX HIDDEN WITH SHOW/HIDE TEXT

Show / Hide Element & Toggle

Show / Hide Text Toggle Highlight

USING TOGGLE BUTTON

Show / Hide Element & Toggle

Show / Hide Text Toggle Highlight

This paragraph will be shown or hidden and can also change its style dynamically.

EXAMPLE OF TOGGLE BUTTON

Show / Hide Element & Toggle

Show / Hide Text Toggle Highlight

This paragraph will be shown or hidden and can also change its style dynamically.

Conclusion

JavaScript provides flexible methods to control visibility and styling of HTML elements. By using;

`style.display`

`classList.toggle()`

Custom CSS classes

Q.4.3 Add interactivity using event listeners (`addEventListener`, `removeEventListener`) for mouse/keyboard events

INTRODUCTION

Event listeners allow JavaScript to respond to user actions such as clicking, typing, moving the mouse, or pressing keys.

```
File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 AYUSHKUMARD

EXPLORER ...
OPEN EDITORS
  QUES4.2.html
  QUES4.3.html
OUTLINE
TIMELINE

QS QUES4.3.html X
QS QUES4.3.html > html > body > script

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Event Listener Example</title>
5
6    <style>
7      .box {
8        width: 300px;
9        padding: 15px;
10       margin-top: 20px;
11       border: 2px solid black;
12       transition: 0.3s;
13     }
14
15    .active {
16      background-color: lightgreen;
17      border-color: darkgreen;
18    }
19  </style>
20 </head>
21 <body>
22
23   <center>
24     <h2>Interactivity Using Event Listeners</h2>
25
26     <button id="clickBtn">Click Me (Mouse Event)</button>
27     <button id="removeBtn">Remove Click Event</button>
28
29     <p class="box" id="textBox">Press any key on your keyboard!</p>
30   </center>
31
32   <script>
33     const clickBtn = document.getElementById("clickBtn");
34     const removeBtn = document.getElementById("removeBtn");
35     const textBox = document.getElementById("textBox");
36
37     // Mouse Event: Function to run on button click

```

`addEventListener()` →

`removeEventListener()` →

Attach an event

Detach an event

WORKS FOR MOUSE, KEYBOARD,

AND OTHER BROWSER EVENTS

Q.4.3 Add interactivity using event listeners (addEventListener, removeEventListener) for mouse/keyboard events

The screenshot shows a code editor interface with the following details:

- Toolbar:** Includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar labeled "AYUSHKU".
- Explorer:** Shows "QUES4.2.html" and "QUES4.3.html" in the "OPEN EDITORS" list.
- Outline and Timeline:** Shows "OUTLINE" and "TIMELINE" sections.
- Editor Area:** Displays the following code:

```
<html>
<body>
    <center>
        <p class="box" id="textBox">Press any key on your keyboard</p>
    </center>

    <script>
        const clickBtn = document.getElementById("clickBtn");
        const removeBtn = document.getElementById("removeBtn");
        const textBox = document.getElementById("textBox");

        // Mouse Event: Function to run on button click
        function showMessage() {
            alert("Button clicked! (Mouse Event Triggered)");
        }

        // Add mouse event listener
        clickBtn.addEventListener("click", showMessage);

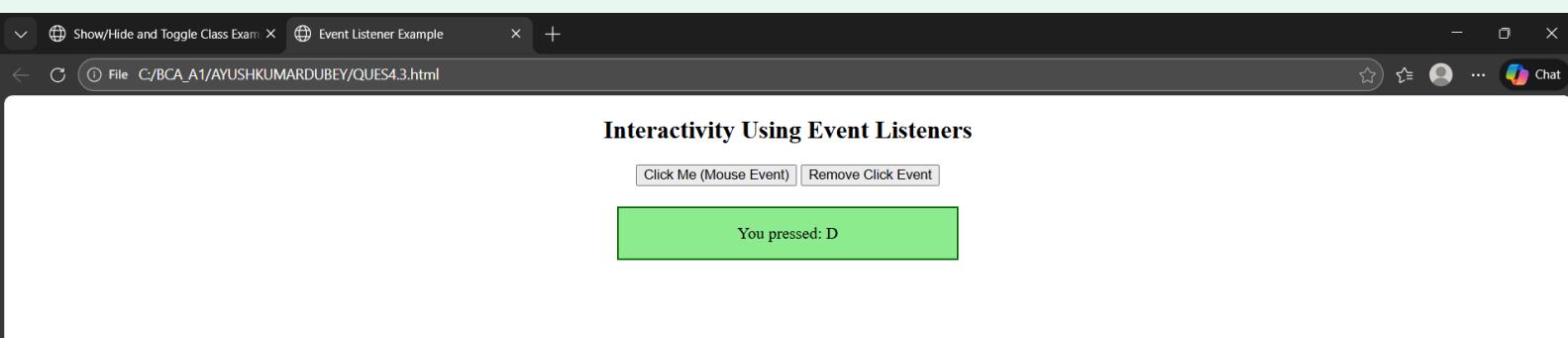
        // Remove mouse event listener
        removeBtn.addEventListener("click", function () {
            clickBtn.removeEventListener("click", showMessage);
            alert("Click event listener removed!");
        });

        // Keyboard Event: Runs when any key is pressed
        document.addEventListener("keydown", function (event) {
            textBox.classList.toggle("active");
            textBox.textContent = "You pressed: " + event.key;
        });
    </script>

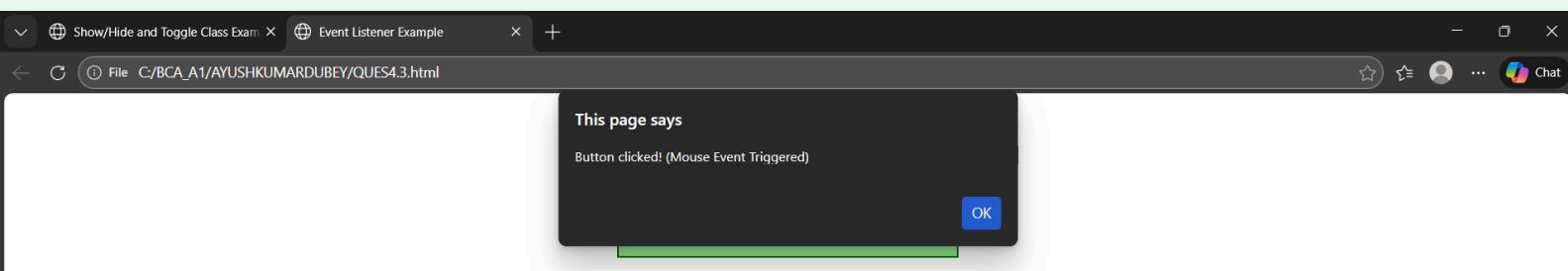
</body>
</html>
```

Q.4.3 Add interactivity using event listeners (addEventListener, removeEventListener) for mouse/keyboard events

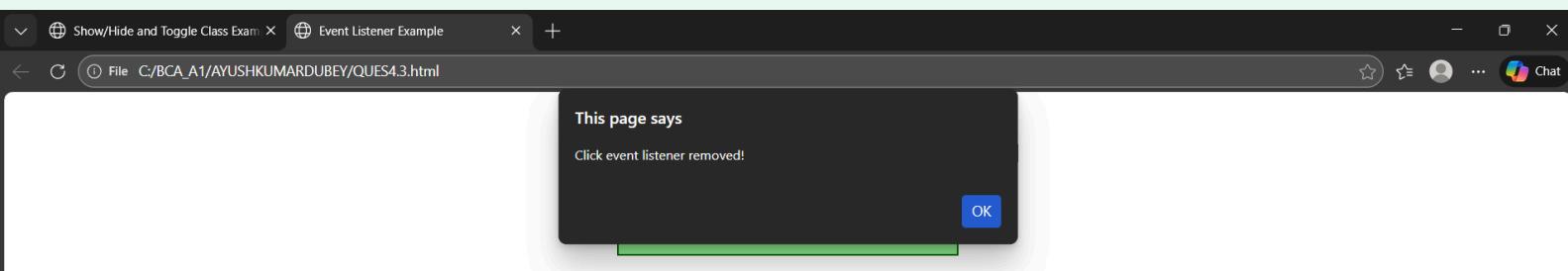
OUTPUT



BUTTON CLICKED



Listener removed



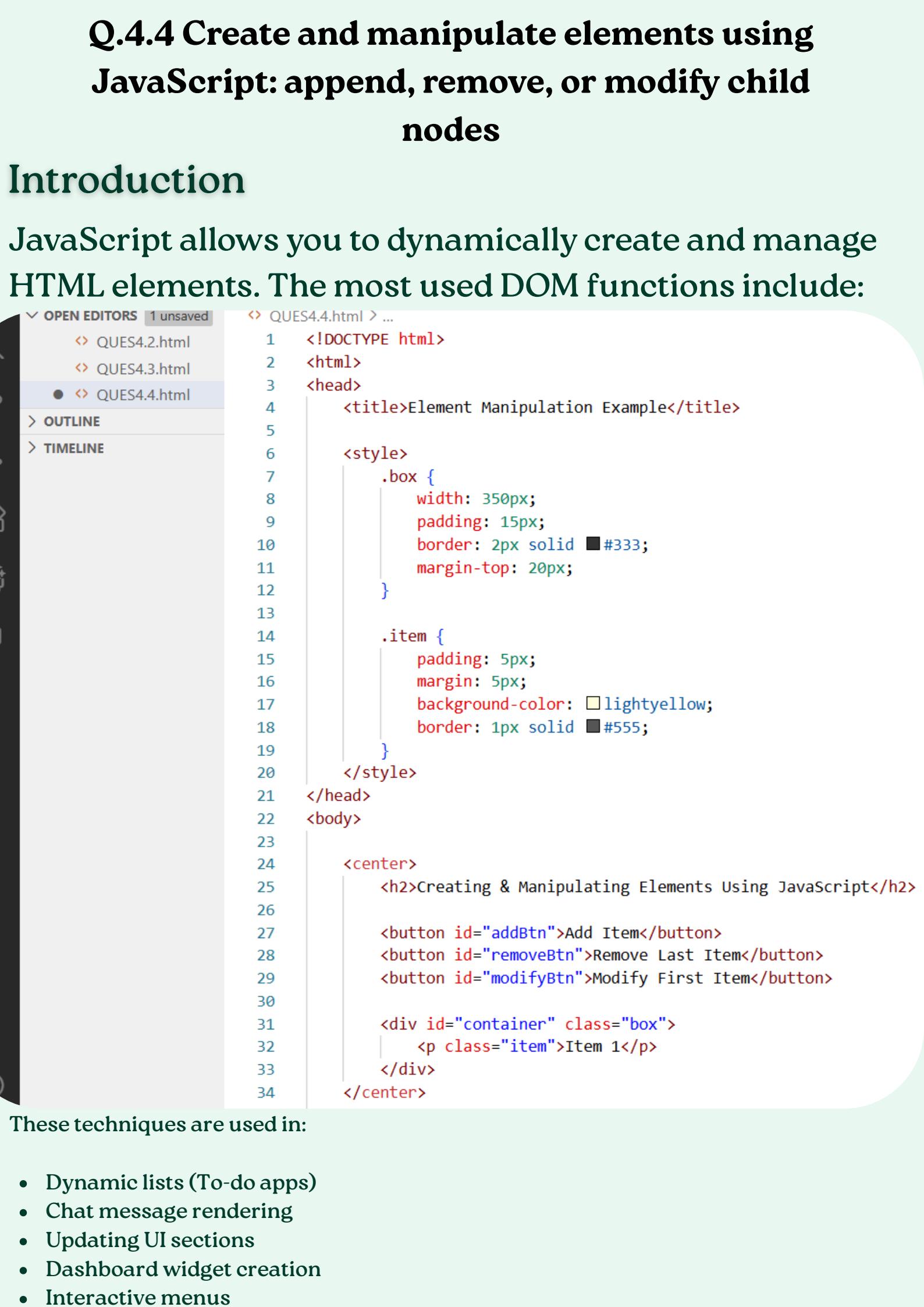
CONCLUSION

- Event listeners are essential for building interactive front-end experiences.**
- Using addEventListener() and removeEventListener(), developers can:**

Q.4.4 Create and manipulate elements using JavaScript: append, remove, or modify child nodes

Introduction

JavaScript allows you to dynamically create and manage HTML elements. The most used DOM functions include:



The screenshot shows a code editor interface with the following details:

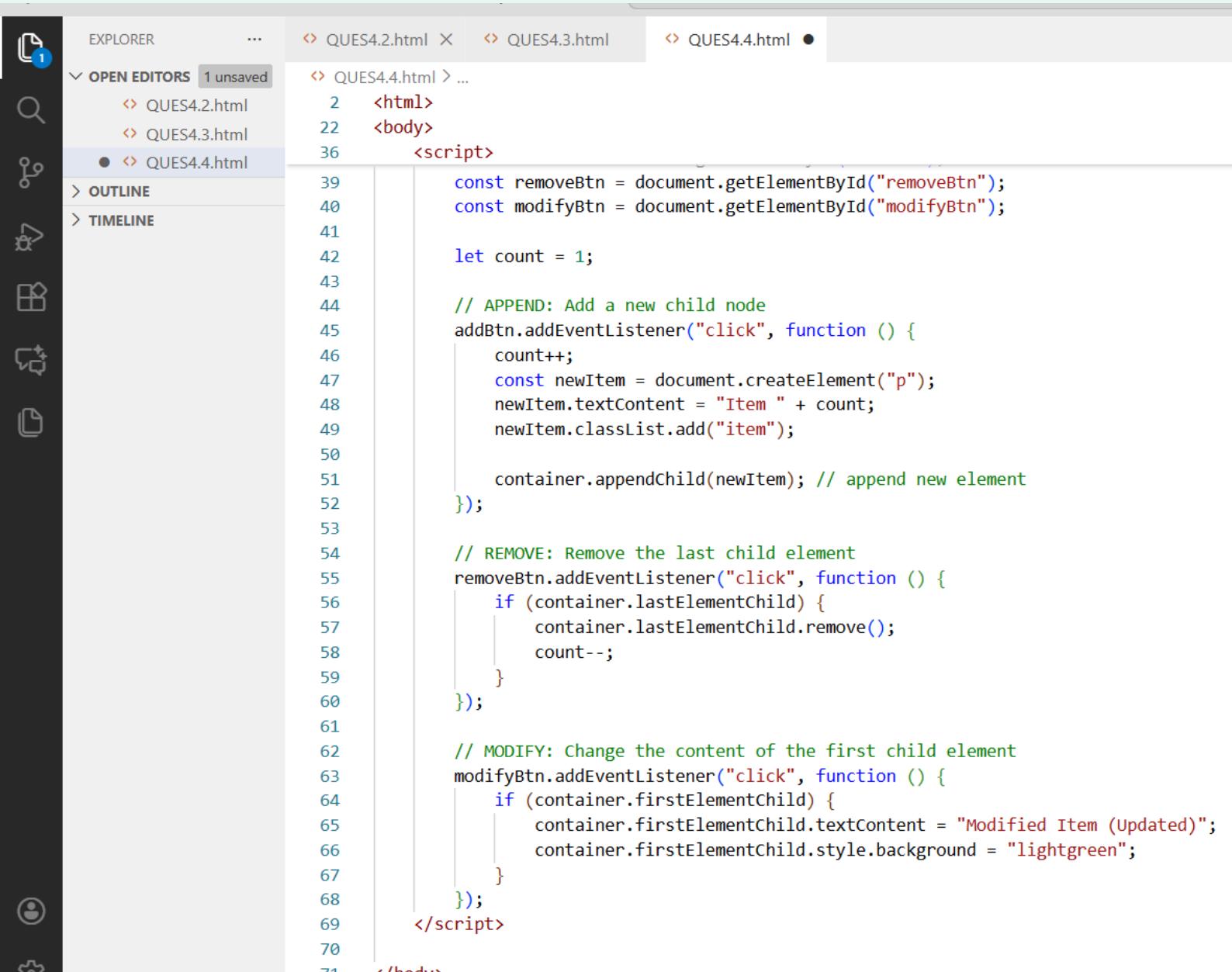
- OPEN EDITORS:** 1 unsaved
- QUES4.4.html** is the active file.
- Outline:** Shows the structure of the document.
- Timeline:** Shows the history of changes.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Element Manipulation Example</title>
5
6   <style>
7     .box {
8       width: 350px;
9       padding: 15px;
10      border: 2px solid #333;
11      margin-top: 20px;
12    }
13
14    .item {
15      padding: 5px;
16      margin: 5px;
17      background-color: lightyellow;
18      border: 1px solid #555;
19    }
20  </style>
21 </head>
22 <body>
23
24   <center>
25     <h2>Creating & Manipulating Elements Using JavaScript</h2>
26
27     <button id="addBtn">Add Item</button>
28     <button id="removeBtn">Remove Last Item</button>
29     <button id="modifyBtn">Modify First Item</button>
30
31     <div id="container" class="box">
32       <p class="item">Item 1</p>
33     </div>
34   </center>
```

These techniques are used in:

- Dynamic lists (To-do apps)
- Chat message rendering
- Updating UI sections
- Dashboard widget creation
- Interactive menus

Q.4.4 Create and manipulate elements using JavaScript: append, remove, or modify child nodes



```
const removeBtn = document.getElementById("removeBtn");
const modifyBtn = document.getElementById("modifyBtn");

let count = 1;

// APPEND: Add a new child node
addBtn.addEventListener("click", function () {
    count++;
    const newItem = document.createElement("p");
    newItem.textContent = "Item " + count;
    newItem.classList.add("item");

    container.appendChild(newItem); // append new element
});

// REMOVE: Remove the last child element
removeBtn.addEventListener("click", function () {
    if (container.lastElementChild) {
        container.lastElementChild.remove();
        count--;
    }
});

// MODIFY: Change the content of the first child element
modifyBtn.addEventListener("click", function () {
    if (container.firstElementChild) {
        container.firstElementChild.textContent = "Modified Item (Updated)";
        container.firstElementChild.style.backgroundColor = "lightgreen";
    }
});
```

JavaScript's DOM manipulation methods allow developers to create, insert, update, and remove HTML elements dynamically. Understanding `createElement`, `appendChild`, `removeChild`, and `innerHTML` is essential for developing interactive and responsive web pages.

Q.4.4 Create and manipulate elements using JavaScript: append, remove, or modify child nodes



Show/Hide and Toggle Class Exam | Event Listener Example | Element Manipulation Example | +

File C:/BCA_A1/AYUSHKUMARDUBEY/QUES4.4.html

Creating & Manipulating Elements Using JavaScript

Add Item Remove Last Item Modify First Item

Item 1

Item 2

ADDING ITEM

Show/Hide and Toggle Class Exam | Event Listener Example | Element Manipulation Example | +

File C:/BCA_A1/AYUSHKUMARDUBEY/QUES4.4.html

Creating & Manipulating Elements Using JavaScript

Add Item Remove Last Item Modify First Item

Item 1

Item 2

Item 3

Item 4

Item 5

ITEM 5 REMOVED

Show/Hide and Toggle Class Exam | Event Listener Example | Element Manipulation Example | +

File C:/BCA_A1/AYUSHKUMARDUBEY/QUES4.4.html

Creating & Manipulating Elements Using JavaScript

Add Item Remove Last Item Modify First Item

Item 1

Item 2

Item 3

Item 4

ITEMS MODIFIED

Show/Hide and Toggle Class Exam | Event Listener Example | Element Manipulation Example | +

File C:/BCA_A1/AYUSHKUMARDUBEY/QUES4.4.html

Creating & Manipulating Elements Using JavaScript

Add Item Remove Last Item Modify First Item

Modified Item (Updated)

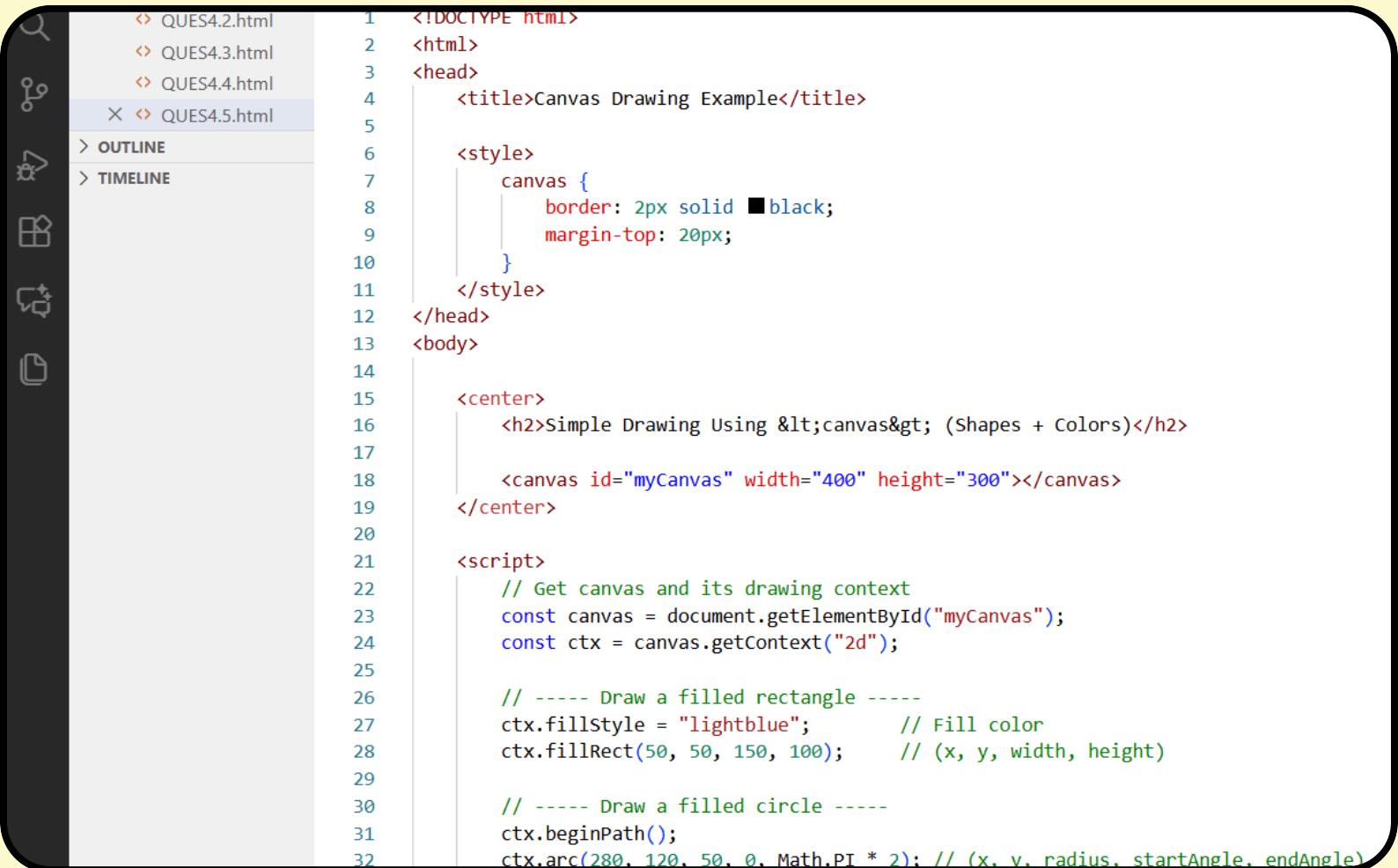
Item 2

Item 3

Item 4

Q.4.5 Create a simple drawing using the `<canvas>` element: draw shapes and fill colors

Explanation of the Methods



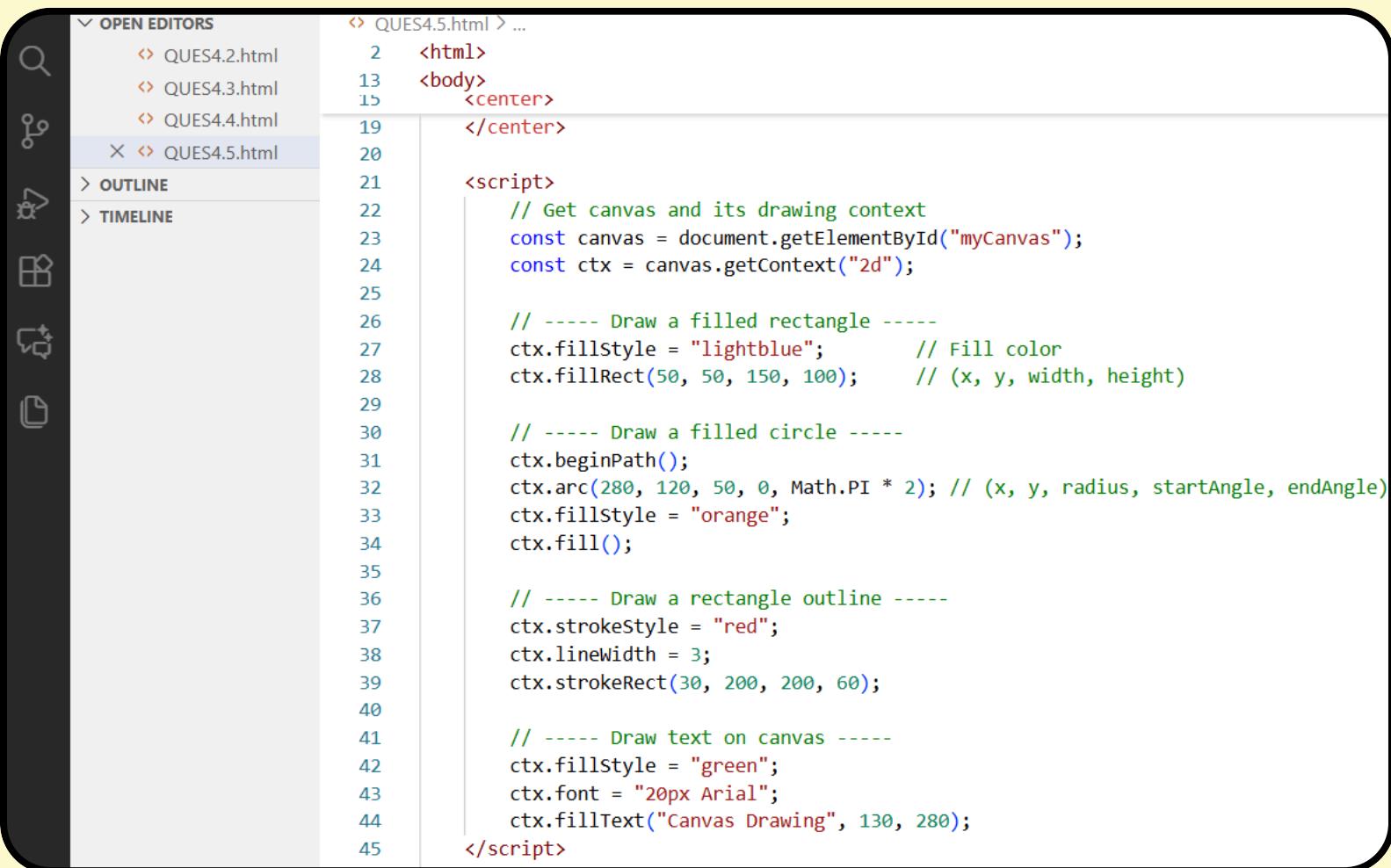
The screenshot shows a code editor interface with a sidebar containing icons for search, file, and navigation. The sidebar also lists other files: QUES4.2.html, QUES4.3.html, QUES4.4.html, and QUES4.5.html. The current file is QUES4.5.html, which is marked with a red X icon.

The main area displays the following code:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Canvas Drawing Example</title>
5
6      <style>
7          canvas {
8              border: 2px solid black;
9              margin-top: 20px;
10         }
11     </style>
12 </head>
13 <body>
14
15     <center>
16         <h2>Simple Drawing Using &lt;canvas&gt; (Shapes + Colors)</h2>
17
18         <canvas id="myCanvas" width="400" height="300"></canvas>
19     </center>
20
21     <script>
22         // Get canvas and its drawing context
23         const canvas = document.getElementById("myCanvas");
24         const ctx = canvas.getContext("2d");
25
26         // ----- Draw a filled rectangle -----
27         ctx.fillStyle = "lightblue";           // Fill color
28         ctx.fillRect(50, 50, 150, 100);       // (x, y, width, height)
29
30         // ----- Draw a filled circle -----
31         ctx.beginPath();
32         ctx.arc(280, 120, 50, 0, Math.PI * 2); // (x, y, radius, startAngle, endAngle)
```

The HTML `<canvas>` element allows you to draw graphics using JavaScript. With the canvas 2D drawing context, you can create shapes, color fills, lines, text, and even animations.

Q.4.5 Create a simple drawing using the `<canvas>` element: draw shapes and fill colors

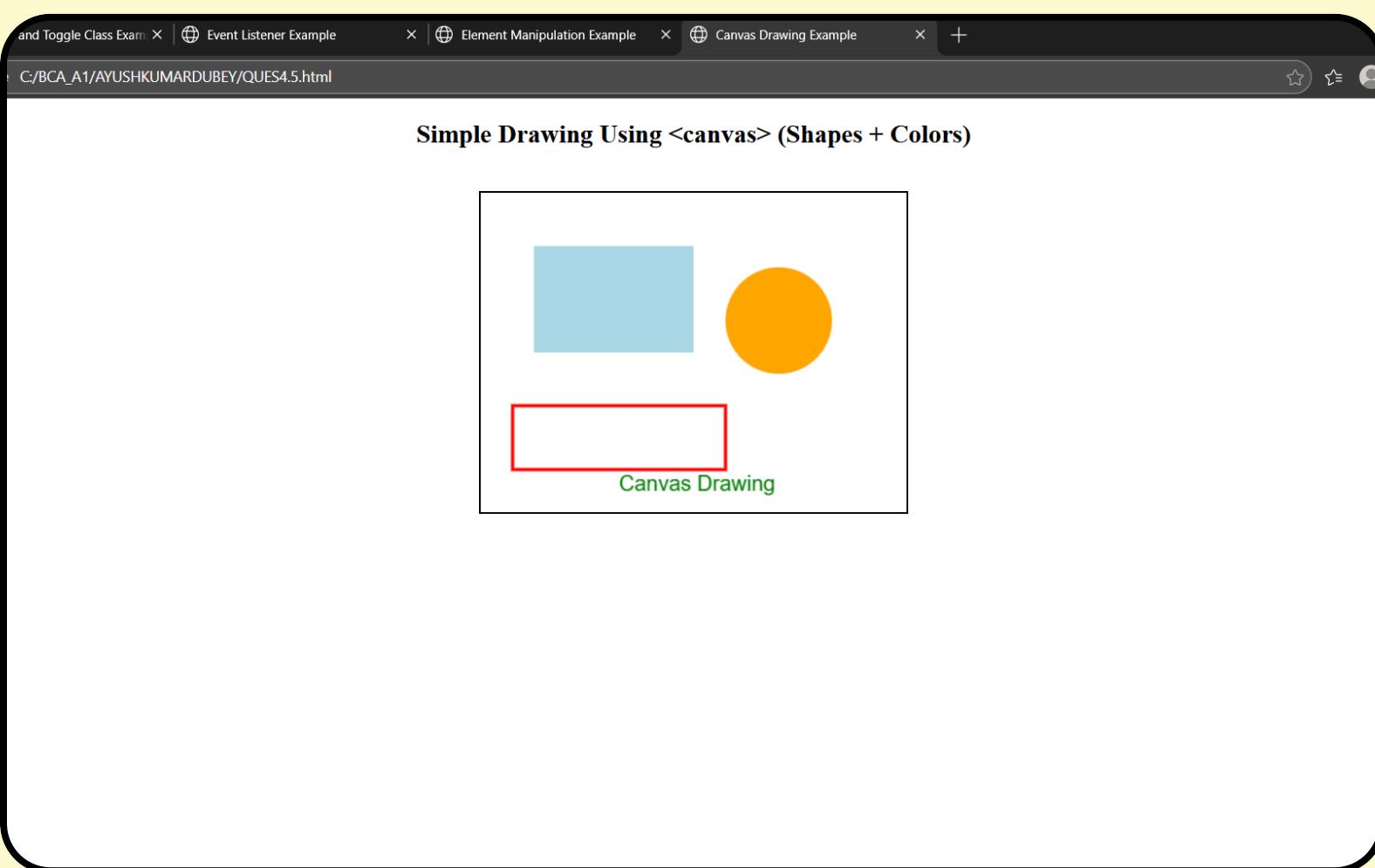


The screenshot shows a code editor interface with a sidebar containing icons for search, file, and project navigation. The main area displays a file named QUES4.5.html. The code within the file is as follows:

```
2  <html>
13 <body>
15 <center>
19   </center>
20
21   <script>
22     // Get canvas and its drawing context
23     const canvas = document.getElementById("myCanvas");
24     const ctx = canvas.getContext("2d");
25
26     // ----- Draw a filled rectangle -----
27     ctx.fillStyle = "lightblue";           // Fill color
28     ctx.fillRect(50, 50, 150, 100);       // (x, y, width, height)
29
30     // ----- Draw a filled circle -----
31     ctx.beginPath();
32     ctx.arc(280, 120, 50, 0, Math.PI * 2); // (x, y, radius, startAngle, endAngle)
33     ctx.fillStyle = "orange";
34     ctx.fill();
35
36     // ----- Draw a rectangle outline -----
37     ctx.strokeStyle = "red";
38     ctx.lineWidth = 3;
39     ctx.strokeRect(30, 200, 200, 60);
40
41     // ----- Draw text on canvas -----
42     ctx.fillStyle = "green";
43     ctx.font = "20px Arial";
44     ctx.fillText("Canvas Drawing", 130, 280);
45   </script>
```

Q.4.5 Create a simple drawing using the `<canvas>` element: draw shapes and fill colors

OUTPUT



Conclusion

Using the `<canvas>` API allows you to draw any shape by combining paths, fills, and strokes.

The essential methods you learned—`fillRect`, `strokeRect`, `beginPath`, `arc`, `lineTo`, `fill`, and `stroke`—form the foundation for more advanced drawings like animations and interactive graphics.

Thank you
for reviewing my assignment.
Your time and guidance are
truly appreciated.

By:-

Ayush Kumar Dubey
RIU:-11741