BATCH: T5

PL-3 ASSIGNMENT NO. 4

**Study of Java Script and DOM.**

NAME: Priti Sunil Desai

PRN NO. : 22510062

Perform following problem statements for DOM using Javascript

**Problem Statement 0: Basics of DOM**

• What is the DOM?

The **Document Object Model** (**DOM**) connects web pages to scripts or programming languages by representing the structure of a document—such as the HTML representing a web page—in memory. Usually it refers to JavaScript, even though modelling HTML, SVG, or XML documents as objects are not part of the core JavaScript language.

The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree. With them, you can change the document's structure, style, or content.

Nodes can also have event handlers attached to them. Once an event is triggered, the event handlers get executed.

• What is DOM Tree Structure? Elaborate its elements with example (you may create DOM tree structure of previously created web pages)

The **DOM Tree Structure** is a hierarchical representation of a web page. It models the document as a tree, where:

* 1.The root node represents the document itself (typically the <html> tag).
* 2.Element nodes represent HTML tags.
* 3.Text nodes represent the text content within elements.
* 4.Attribute nodes represent the attributes of elements (like class, id, etc.).

Each node in the DOM can have child nodes, sibling nodes, and a parent node, forming a tree-like structure.

**Example of a DOM Tree Structure**

HTML CODE=>

<html lang="en">

<head>

<title>My Document</title>

</head>

<body>

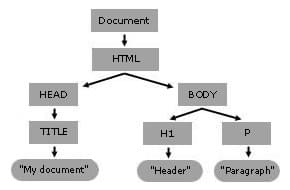
<h1>Header</h1>

<p>Paragraph</p>

</body>

</html>

TREE STRUCTURE=>



• Give examples for following:

1) Accessing the DOM

Accessing elements within the DOM allows you to retrieve nodes to read or manipulate their content.

let heading = document.querySelector('h1'); // Accesses the first <h1> element

2) Manipulating the DOM

Manipulating the DOM involves changing the structure, content, or style of the elements.

heading.textContent = 'Hello, World!'; // Changes the text inside the <h1> element

heading.style.color = 'blue'; // Changes the color of the <h1> element to blue

3) Event Handling

Event handling allows you to respond to user actions like clicks, keypresses, etc.

heading.addEventListener('click', function()

{

alert('Heading clicked!');

});

4) Traversing the DOM

DOM traversal refers to moving between nodes (parents, children, siblings) in the DOM tree.

let paragraph = heading.nextElementSibling; // Accesses the next sibling element after <h1>

• What are Performance Considerations while implementing the DOM and can DOM supports all browsers?

**Minimize DOM Access**:

* **Accessing the DOM is slow**: Each time you access or manipulate the DOM, the browser may need to re-render parts of the page. This can be resource-intensive.
* **Cache DOM elements**: Instead of repeatedly accessing the same DOM element, store it in a variable.

**Use Virtual DOM**:

* **Leverage libraries like React**: The Virtual DOM is an abstraction that allows efficient updates by minimizing direct manipulations to the actual DOM, reducing the number of reflows and repaints.

**Limit the Use of Expensive CSS Properties**:

* **Some CSS properties are more expensive**: Properties like box-shadow, border-radius, or transform can be costly to render. Use them judiciously, especially on elements that are animated or change frequently.

Browser Compatibility of the DOM

The DOM is a standard API supported by all modern browsers, including Chrome, Firefox, Safari, Edge, and others. However, there are a few considerations:

1 While core methods and properties (like getElementById, querySelector, innerHTML, etc.) are supported across all modern browsers, some newer methods or properties (like classList, closest, etc.) may have limited support in older browsers.

2 Use feature detection to ensure compatibility.

3 Different browsers may implement the DOM API slightly differently, leading to inconsistencies. Testing across multiple browsers is crucial to ensure consistent behavior.

• Elaborate Common Methods and Properties of DOM

The DOM provides a wide range of methods and properties that allow developers to interact with and manipulate web pages. Here's an overview of some of the most commonly used methods and properties:

getElementById(id): Returns the element with the specified id attribute.

getElementsByClassName(className): Returns a collection of all elements that have the specified class name. The result is a live HTMLCollection (a collection of elements that updates automatically if the document changes).

getElementsByTagName(tagName): Returns a collection of all elements with the specified tag name (e.g., div, p, etc.).

querySelector(selector): Returns the first element that matches a specified CSS selector. It allows for more complex queries than getElementById or getElementsByClassName.

querySelectorAll(selector): Returns a static NodeList of all elements that match the specified CSS selector(s). Unlike getElementsByClassName, it doesn't automatically update when the document changes.

**Problem Statement 1: DOM selector methods**

**•** Here, the existing code expects the variables 'buttonElem' and 'inputElem' to represent the button and input elements in the example UI. Assign the respective elements to the variables. In this case, the two elements do not have unique identifiers - like for example an id. Instead they are direct descendents of a div element with id 'wrapper'. Use an appropriate selector method! Click the button to verify that the code is working.

• In this scenario, we are looking for a list of elements gathered in one variable - rather than only one element. Assign the list items in the view to the variable 'listItems' by using an appropriate selector method. Once you have completed the code below, verify it by hovering over the list items until all items have the value 'ON'.



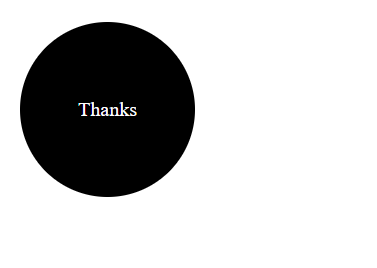
**Problem Statement 2: Events and user interactions**

• The Javascript function handleText fills the input field with the words Hello World. But, there is no code to execute this function. Complete the existing code below such that the function is called when the button is clicked. Verify by clicking the button.

• The Javascript function changeText changes the text inside the circle. But again, there is no code to execute this function. Complete the existing code below such that the function is called when the cursor moves onto the circle. Verify that your code works by hovering over the circle.

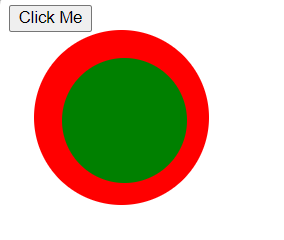
• In this scenario we want the color of the circle to change depending on the type of cursor movement. Use the function toggleColor to turn the circle orange when the cursor moves onto it. Reuse the same function to turn it black when the cursor leaves it. The tricky part is that you have to call toggleColor with different values for the parameter isEntering. Verify that your code is working by hovering the circle with the mouse cursor and leaving it again.





**Problem Statement 3: DOM manipulation with JavaScript**

• Remove element from the DOM. Create t2 circles red and green and a button clickme. Place them such a way that red circle hides the green circle. Add the function removeRedCircle to remove the circle with id red from the DOM when clicked on clickme button. Make sure that you really remove the element instead of just hiding it.





**Problem Statement 4: DOM fundamentals**

• Create JavaScript code to interact with the displayed HTML elements. Create a checkbox and a button. Once you click the button, the checkbox should be checked.



• Create 3 textboxes and a button. First 2 checkboxes contain first name and last name respectively. When the button is clicked, combine the names of the first two input fields. Insert the full name in the third input field (textbox). Check if your code still works if you change the first or last name.

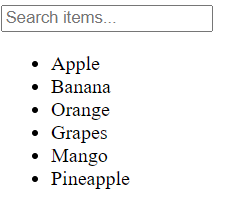
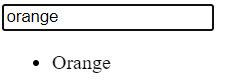




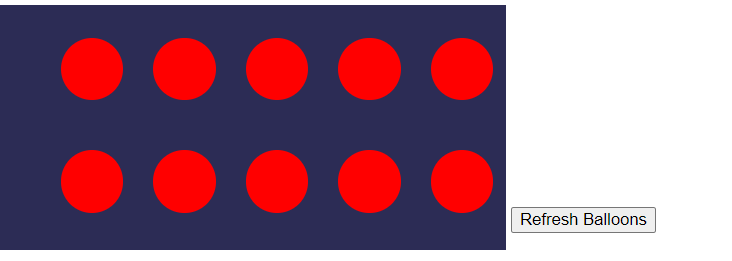
• Create three buttons. One button displays value of 0. Other two buttons are for increment and reset. By clicking increment button each time, increase the value of the button by 1. By clicking the reset button set the value of button to 0. Confirm your code by clicking the buttons.

• create a dynamic input filter with JavaScript. Type a search term in the input field. The displayed items in the list should match your search term. The rest of the list elements should be hidden.

• Create 10 balloons as shown below. Every time you hover over a balloon, it should become invisible. Your goal is to pop all the balloons one after the other. Create a refresh button. After clicking refresh button it will again display all the balloons.





**Problem Statement 5: Recursive functions**

• Create a function move that moves the button 1px to the left or the right. It is recursive because it calls itself again and again. This keeps the button moving. Extend the JavaScript code. Once you click the button, it should stop moving. When you click it again, it should move again.

