

**Batch - T5****Practical No. - 4****Title – Study of JavaScript and DOM****Student Name - Sharaneshwar Bharat Punjal****Student PRN - 23520011****Perform following problem statements for DOM using Javascript****Problem Statement 0: Basics of DOM****1. What is the DOM?**

The DOM (Document Object Model) is a programming interface for web documents. It represents the structure of a webpage as a tree of objects, where each object corresponds to a part of the document (like elements, attributes, or text).

**2. 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 the elements in an HTML document. It starts from the root (<html> tag) and branches out to include every element on the page, forming a tree-like structure.

Example of a DOM Tree: For a simple HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Page</title>
  </head>
  <body>
    <h1>Welcome</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

The DOM tree structure would look like:

- html
  - head
    - title
  - body
    - h1
    - p

**3. Give examples for following:****a. Accessing the DOM**

Accessing the DOM involves retrieving specific elements or groups of elements from a webpage using JavaScript. This is typically done to manipulate or interact with the elements.

Example - To access the <h1> element

```
const header = document.querySelector('h1');
```

**b. Manipulating the DOM**

Manipulating the DOM refers to modifying the structure, content, or styling of elements on a webpage using JavaScript. This allows for dynamic changes to the page after it has loaded.

Example: Change the text of <h1>

```
header.textContent = 'Hello, World!';
```

**c. Event Handling**

Event Handling involves responding to user interactions or other events on a webpage. JavaScript allows you to add "event listeners" to elements that trigger functions when specific events occur, like clicks, hovers, or keystrokes.

Example: Add a click event to <h1>

```
header.addEventListener('click', () => {  
  alert('Header clicked!');  
});
```

**d. Traversing the DOM**

Traversing the DOM means navigating through the DOM tree to move from one element to another. This is useful when you need to find related elements or move through a series of nodes.

Example: Get the parent of the <h1>

```
const parentElement = header.parentElement;
```

**4. What are Performance Considerations while implementing the DOM and can DOM supports all browsers?**

When working with the DOM, consider:

Minimizing reflows/repaints: Frequent changes to the DOM can cause the page to reflow, which can slow down performance.

Batch updates: Apply multiple DOM changes in a single operation to improve performance.

Caching selectors: Accessing the same DOM element multiple times can be slow; cache it in a variable.

The DOM is widely supported across all modern browsers, though there can be differences in implementation for older browsers.

## 5. Elaborate Common Methods and Properties of DOM

The DOM (Document Object Model) provides a variety of methods and properties that allow developers to interact with and manipulate HTML documents programmatically using JavaScript. Here's an overview of some of the most common methods and properties:

a) `getElementById()`: Selects an element by its unique id attribute.

Example: `const element = document.getElementById('header');`

b) `getElementsByClassName()`: Selects all elements that share a specific class name and returns a live `HTMLCollection`.

Example: `const items = document.getElementsByClassName('menu-item');`

c) `getElementsByTagName()`: Selects all elements with a specific tag name and returns a live `HTMLCollection`.

Example: `const paragraphs = document.getElementsByTagName('p');`

d) `querySelector()`: Selects the first element that matches a specified CSS selector.

Example: `const firstItem = document.querySelector('.menu-item');`

e) `querySelectorAll()`: Selects all elements that match a specified CSS selector and returns a static `NodeList`.

Example: `const allItems = document.querySelectorAll('.menu-item');`

f) `innerHTML`: Gets or sets the HTML content inside an element.

Example:

```
const container = document.getElementById('container');
```

```
container.innerHTML = '<p>New content</p>';
```

g) `textContent`: Gets or sets the text content of an element, ignoring HTML tags.

Example:

```
const header = document.getElementById('header');
```

```
header.textContent = 'New Header Text';
```

h) `className`: Gets or sets the value of the class attribute of an element.

Example:

```
const element = document.getElementById('element');
```

```
element.className = 'new-class';
```

i) `classList`: Provides methods to add, remove, and toggle CSS classes on an element.

Example:

```
element.classList.add('new-class');
```

```
element.classList.remove('old-class');
```

```
element.classList.toggle('active');
```

**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.

reset

View

HTML

Javascript

OFF

Click Me

```
<div id="wrapper">
  <input type="text" value="OFF" readonly/>
  <button type="button">Click Me</button>
</div>
```

```
// assign the correct elements to the variables
const buttonElem =
const inputElem =

buttonElem.addEventListener('click', () => {
  const oldText = inputElem.value;
  return inputElem.value = oldText === "ON" ? "OFF" : "ON";
});
```

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'.

reset

View

HTML

Javascript

OFF

OFF

OFF

OFF

OFF

OFF

```
<ul id="list">
  <li>OFF</li>
  <li>OFF</li>
  <li>OFF</li>
  <li>OFF</li>
  <li>OFF</li>
  <li>OFF</li>
</ul>
```

```
// assign the correct elements to the variable
const listItems =

const handleHover = (event) => {
  return event.target.innerText = 'ON';
};
if(listItems.length > 1) {
  listItems.forEach(item => item.addEventListener('mouseover', handleHover));
}
```

**Screenshots:**

**Problem Statement 1**

**Problem Statement 1**

**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.

reset

View

```
<input type="text" id="input" readonly/>
<button type="button" id="button">Click Me</button>
```

HTML

```
const button = document.getElementById('button');
const input = document.getElementById('input');

const handleClick = () => {
  input.value = 'Hello World';
};

// type in your code here
```

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.



```
const element = document.getElementById('element');

const changeText = () => {
  element.innerText = 'Thanks!';
};

// type in your code here
```

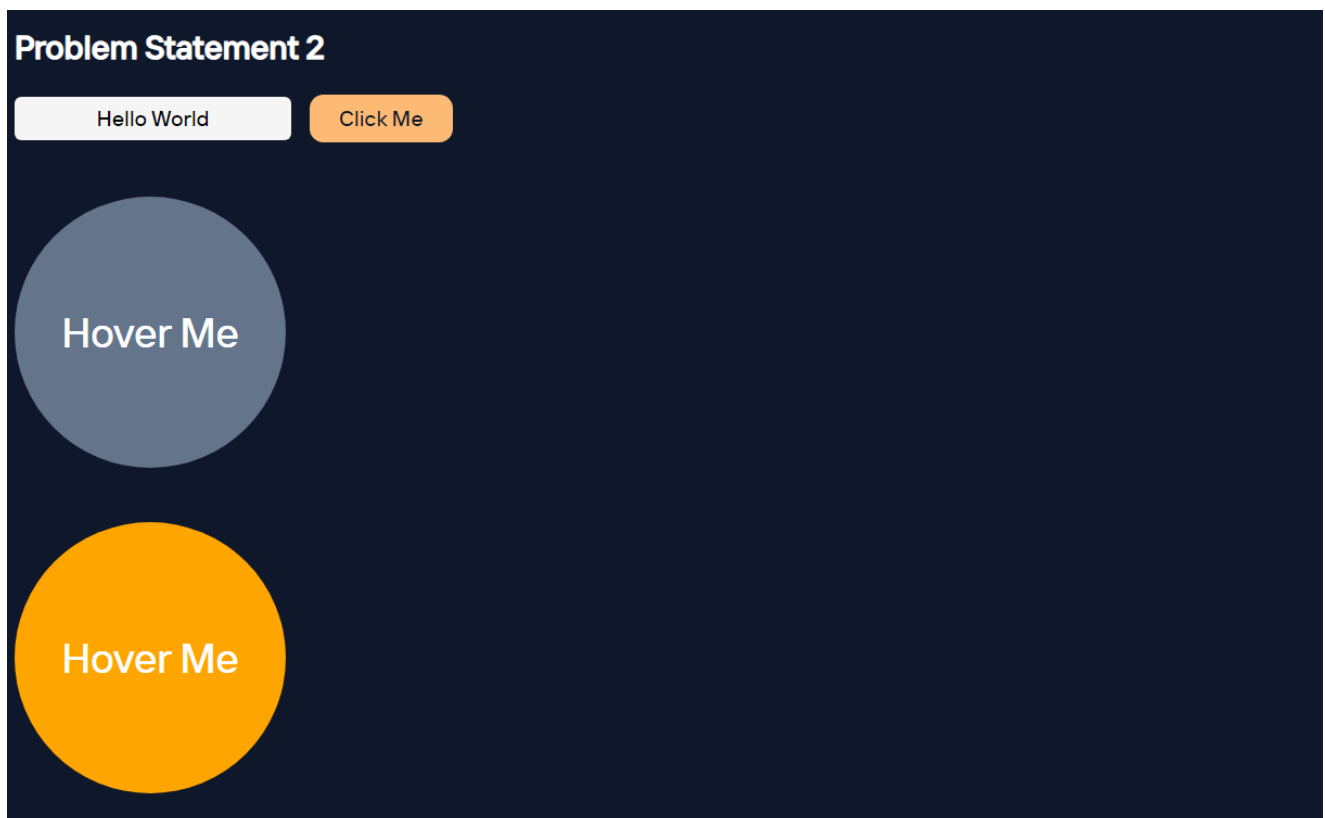
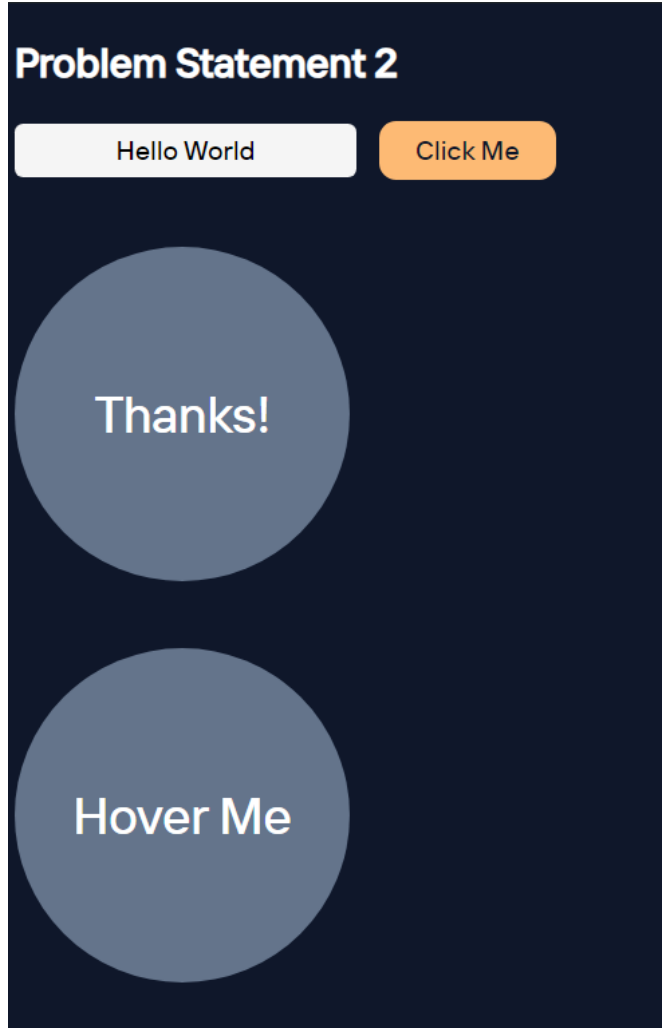
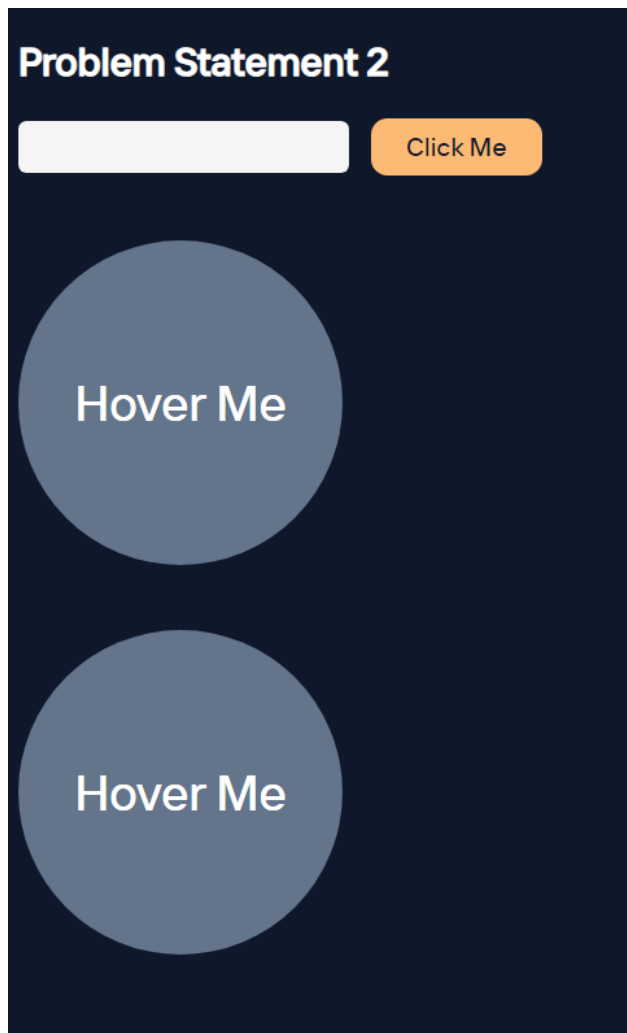
In this scenario we want the color of the circle to change depending on the type of cursor movement. Use the function `toggleColor` to turn the circle orange when the cursor moves onto it. Reuse the same function to turn it black when the cursor leaves it. The tricky part is that you have to call `toggleColor` 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.



```
const element = document.querySelector('#element');

const toggleColor = (isEntering) => {
  element.style.background = isEntering ? 'orange' : 'black';
};
```

**Screenshots:**



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

Remove element from the DOM. Create 2 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.

**Screenshots:****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.

**Screenshots:**

The screenshot shows a web application interface on a dark blue background. At the top, the title "Problem Statement 4" is displayed in white. Below the title, there is a white checkbox and an orange button labeled "Check Checkbox". Further down, there are two white input fields labeled "First Name" and "Last Name", followed by a single white input field labeled "Full Name". Below these fields are two orange buttons: "Combine Names" and "Reset". A search section follows, featuring a white input field labeled "Search Fruit" and a list of fruit names in white buttons: "Apple", "Banana", "Cherry", "Dates", "Grapes", "Mango", and "Orange". At the bottom of the interface, there are ten red circles arranged in two rows of five, representing balloons. An orange button labeled "Refresh" is located at the bottom left of the interface.

**Results:**

## Problem Statement 4

☒

Check Checkbox

Sharaneshwar

Punjal

Sharaneshwar Punjal

Combine Names

Reset

g

Grapes


Mango

Orange

Refresh

**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.

**Problem Statement 5**Click me**Problem Statement 5**Click me