|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Web Programming Laboratory 116U40L501** | **Semester:** | **V** |
| **Date of Performance:** | **12/08/2024** | **Batch No:** | **A1** |
| **Faculty Name:** | **Prof. Madhura Pednekar** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** |  |

**Experiment No: 3**

**Title: Develop and demonstrate JavaScript with POP-UP boxes and functions.**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| To demonstrate the functionalities of JavaScript using HTML and CSS. |

|  |
| --- |
| **COs to be achieved:** |
| **CO1:** Understand the fundamentals of web development technologies, including HTML, CSS, and Bootstrap.  **CO2:** Design responsive web pages using HTML, CSS, and Bootstrap, improving user interfaces |

## **Apparatus / Software tools used:**

|  |
| --- |
| **Problem Statement** |
| **a) Input**: Click on Display Date button using onclick( ) function  **Output:** Display date in the textbox  **b) Input:** A number n obtained using prompt  **Output:** Factorial of n number using alert   |  | | --- | | c) Implement a static webpage for the Retail Store management system using JavaScript with POP-UP boxes | |

|  |
| --- |
| **Javascript Basic Concepts Learned With Syntax:** |
| 1. Event Handling   Event handling in JavaScript refers to the process of capturing and responding to user actions, such as clicks, key presses, or form submissions. Events can be tied to specific elements on a webpage, allowing the developer to define custom behavior when an event occurs. For example, attaching a function to a button’s click event enables dynamic interactions on the page.   1. Functions   Functions are blocks of reusable code that perform specific tasks. They can be defined once and called multiple times, which helps in reducing code duplication and improving modularity. Functions can accept inputs (parameters) and return outputs, making them versatile tools for handling logic within a program.   1. DOM Manipulation   The Document Object Model (DOM) is an interface that allows JavaScript to interact with and modify the structure, content, and styling of a webpage. Through DOM manipulation, developers can dynamically change elements, such as updating text, altering styles, or adding/removing elements based on user actions.   1. User Interaction with Prompt, Alert, and Confirm   JavaScript provides built-in methods to interact with the user through dialog boxes:   * Prompt allows for input collection. * Alert is used to display messages. * Confirm can be used to ask the user to make a choice (e.g., OK or Cancel).   These dialogs are useful for gathering information or providing feedback directly to the user. |

|  |
| --- |
| **Code:** |
| **a) Input**: Click on Display Date button using onclick( ) function  **Output:** Display date in the textbox  **b) Input:** A number n obtained using prompt  **Output:** Factorial of n number using alert  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Minimal JS Example</title>  <style>  body {  font-family: Arial, sans-serif;  margin: 0;  padding: 20px;  display: flex;  flex-direction: column;  align-items: center;  justify-content: center;  height: 100vh;  background-color: #f5f5f5;  }  button, input {  margin: 10px 0;  padding: 8px 12px;  font-size: 14px;  }  input {  width: 200px;  text-align: center;  }  </style>  </head>  <body>  <button onclick="displayDate()">Display Date</button>  <input type="text" id="dateBox" readonly>  <button onclick="calculateFactorial()">Calculate Factorial</button>  <script>  function displayDate() {  document.getElementById('dateBox').value = new Date().toLocaleString();  }  function calculateFactorial() {  const n = prompt("Enter a number:");  if (n !== null && !isNaN(n) && n > 0) {  let factorial = 1;  for (let i = 1; i <= n; i++) {  factorial \*= i;  }  alert(`Factorial of ${n} is ${factorial}`);  } else {  alert("Please enter a valid positive number.");  }  }  </script>  </body>  </html>     |  | | --- | | **c) Implement a static webpage for the Retail Store management system using JavaScript with POP-UP boxes**  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Retail Store Management System</title>  <style>  body {  font-family: Arial, sans-serif;  margin: 0;  padding: 20px;  background-color: #f0f0f0;  display: flex;  flex-direction: column;  align-items: center;  }  h1 {  color: #333;  text-align: center;  margin-bottom: 20px;  }  .button-container {  display: flex;  flex-wrap: wrap;  justify-content: center;  gap: 10px;  max-width: 600px;  }  button {  padding: 10px 20px;  font-size: 16px;  cursor: pointer;  background-color: #5bc0c4;  color: white;  border: none;  border-radius: 5px;  transition: background-color 0.3s ease;  }  button:hover {  background-color: #45a049;  }  </style>  </head>  <body>  <h1>Retail Store Management System</h1>  <div class="button-container">  <button onclick="addProduct()">Add Product</button>  <button onclick="removeProduct()">Remove Product</button>  <button onclick="updateStock()">Update Stock</button>  <button onclick="checkPrice()">Check Price</button>  <button onclick="displayInventory()">Display Inventory</button>  </div>  <script>  // JavaScript code remains the same as in the previous example  let inventory = [];  function addProduct() {  let name = prompt("Enter product name:");  let price = parseFloat(prompt("Enter product price:"));  let stock = parseInt(prompt("Enter initial stock:"));  if (name && !isNaN(price) && !isNaN(stock)) {  inventory.push({ name, price, stock });  alert("Product added successfully!");  } else {  alert("Invalid input. Product not added.");  }  }  function removeProduct() {  let name = prompt("Enter product name to remove:");  let index = inventory.findIndex(item => item.name.toLowerCase() === name.toLowerCase());    if (index !== -1) {  inventory.splice(index, 1);  alert("Product removed successfully!");  } else {  alert("Product not found.");  }  }  function updateStock() {  let name = prompt("Enter product name to update stock:");  let product = inventory.find(item => item.name.toLowerCase() === name.toLowerCase());    if (product) {  let newStock = parseInt(prompt("Enter new stock amount:"));  if (!isNaN(newStock)) {  product.stock = newStock;  alert("Stock updated successfully!");  } else {  alert("Invalid stock amount.");  }  } else {  alert("Product not found.");  }  }  function checkPrice() {  let name = prompt("Enter product name to check price:");  let product = inventory.find(item => item.name.toLowerCase() === name.toLowerCase());    if (product) {  alert(`Price of ${product.name}: $${product.price.toFixed(2)}`);  } else {  alert("Product not found.");  }  }  function displayInventory() {  if (inventory.length === 0) {  alert("Inventory is empty.");  } else {  let inventoryList = inventory.map(item =>  `${item.name} - Price: $${item.price.toFixed(2)}, Stock: ${item.stock}`  ).join("\n");  alert("Current Inventory:\n\n" + inventoryList);  }  }  </script>  </body>  </html>   | |

|  |
| --- |
| **Stepwise-Procedure / Algorithm:** |
| To create the provided JavaScript examples, the following steps were taken:   1. Set Up the HTML Structure:   Create the basic HTML structure, including the <!DOCTYPE html>, <html>, <head>, and <body> tags. Include necessary metadata like charset and viewport.   1. Design the User Interface (UI):   Add buttons and input fields in the HTML to interact with the user. Apply CSS styles for layout, alignment, and aesthetics.   1. Define Event Handlers:   Write JavaScript functions that respond to user actions (e.g., clicking a button).  Link these functions to buttons using the onclick attribute.   1. Implement Core Logic:   Inside the JavaScript functions, add logic for specific tasks (e.g., displaying the date, calculating factorial, managing inventory). Use DOM manipulation to update the page content based on user interaction.   1. Test User Interactions:   Test the buttons and prompts to ensure they work as expected, handling valid and invalid inputs. Adjust the code for any errors or edge cases. |

|  |
| --- |
| **Output** |
|  |
| **Link of the Generated website** |
| It’s a static html webpage. |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| **1 What are the possible ways to create objects in JavaScript?**   * Object Literal: The simplest way to create an object. Example: let obj = { key: "value" }; * Constructor Function: Defines a function and uses the new keyword to create objects.   function MyObject() {  this.key = "value";  }  let obj = new MyObject();     * ES6 Class: Uses the class syntax introduced in ES6. Example:   class MyObject {  constructor() {  this.key = "value";  }  }  let obj = new MyObject();    **2 What is the Difference between == and === operators**  == (Abstract Equality): Compares two values for equality after performing type conversion if necessary. For example, 5 == '5' returns true because the string '5' is converted to a number before comparison.  === (Strict Equality): Compares two values for equality without performing type conversion. The types of the values must be the same for it to return true. For example, 5 === '5' returns false because one is a number and the other is a string.  **3 What is the difference between let and var?**   1. Scope:  * var is function-scoped, meaning it is accessible within the entire function where it is declared, even before its declaration due to hoisting. * let is block-scoped, meaning it is only accessible within the block (e.g., inside an if or for loop) where it is declared.  1. Hoisting:  * Variables declared with var are hoisted to the top of their scope and initialized with undefined. * Variables declared with let are hoisted but not initialized, leading to a ReferenceError if accessed before declaration.  1. Redeclaration:  * var allows redeclaration within the same scope. * let does not allow redeclaration within the same scope, preventing accidental overwrites. |

|  |
| --- |
| **Conclusion:** |
| The experiment effectively demonstrated the core functionalities of JavaScript, including the use of pop-up boxes such as alert(), confirm(), and prompt(), integrated with HTML and styled using CSS. By implementing these functions, the experiment highlighted how JavaScript can dynamically interact with users and manipulate the webpage, confirming its essential role in enhancing web interactivity and user experience. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |