DOM Manipulation, Event Handling, Exception Handling, Regular Expressions & Form Validation

PART 1: DOM MANIPULATION

1. What is the DOM?

The Document Object Model (DOM) is a programming interface provided by web browsers that represents an HTML or XML document as a hierarchical tree of nodes and objects. It allows programming languages like JavaScript to dynamically interact with, modify, create, and delete web page content, structure, and styles.

2. Core DOM Methods for Selecting and Modifying Elements

- document.getElementById(id): Selects a single element by its unique ID attribute.
- document.querySelector(selector): Returns the first element that matches a specified CSS selector (e.g., '.class', '#id', 'tag').
- document.querySelectorAll(selector): Returns a static NodeList of all elements matching the specified CSS selector.
- element.textContent vs element.innerHTML: textContent sets or gets the raw text content inside an element (safe from XSS), while innerHTML parses and renders HTML markup strings.

3. Program: Dynamic DOM Manipulation

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>DOM Manipulation Demo</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 40px; }
    .highlight { background-color: #fffa65; padding: 10px; }
  </style>
</head>
<body>
  <h1 id="title">Original Title</h1>
  <p class="desc">This is a paragraph.</p>
  <button id="btn">Change Content & Style</button>
  <script>
    const btn = document.getElementById('btn');
    const title = document.getElementById('title');
    const paragraph = document.querySelector('.desc');

    btn.addEventListener('click', () => {
      title.textContent = 'Updated DOM Title!';
```

```
        paragraph.innerHTML = '<strong>This text has been dynamically modified with innerHTML.</strong>';
        paragraph.classList.add('highlight');
      });
  </script>
</body>
</html>
```

PART 2: EVENT HANDLING

1. What is Event Handling?
Event handling refers to the mechanism by which JavaScript detects and responds to user interactions or browser actions (such as clicks, keyboard input, form submissions, page loading, mouse hovering) using event listeners.

2. Key Event Concepts
- addEventListener(event, callback): The standard method to attach an event listener to an element without overwriting existing handlers.
- Event Object: Passed automatically to the callback function, containing details about the event (e.g., event.target, event.clientX).
- Event Bubbling and Capture: Events bubble up from the innermost target element to the outer ancestors. Event delegation uses bubbling to handle events on parent containers efficiently.

3. Program: Interactive Event Handling & Delegation

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Event Handling Demo</title>
  <style>
    ul { list-style: none; padding: 0; }
    li { padding: 8px; margin: 4px 0; background: #e0f7fa; cursor: pointer; }
  </style>
</head>
<body>
  <h2>Click an item to remove it (Event Delegation)</h2>
  <ul id="item-list">
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>
  <script>
    const list = document.getElementById('item-list');
    list.addEventListener('click', (event) => {
```

```
        if (event.target.tagName === 'LI') {
            alert(`Deleting: ${event.target.textContent}`);
            event.target.remove();
        }
    });
  </script>
</body>
</html>
```

PART 3: EXCEPTION HANDLING

1. What is Exception Handling?
Exception handling is the process of gracefully intercepting, managing, and recovering from runtime errors (exceptions) so that the program does not crash unexpectedly and can provide meaningful feedback to users or logs.

2. The try...catch...finally Construct
- try: Encloses code that may potentially throw an error.
- catch (error): Captures the thrown exception object, allowing execution to handle the error condition.
- finally: Executes code unconditionally after try and catch blocks finish, regardless of whether an error occurred (useful for cleanup).

3. Program: Custom Exception Handling

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Exception Handling Demo</title>
</head>
<body>
  <h2>Check Console for Output</h2>
  <script>
    function divideNumbers(a, b) {
      try {
        if (b === 0) {
          throw new Error('Division by zero is not allowed.');
        }
        let result = a / b;
        console.log(`Result: ${result}`);
      } catch (error) {
        console.error(`Caught Exception: ${error.message}`);
      } finally {
        console.log('Execution completed (cleanup operations).');
      }
```

```
    }
    divideNumbers(10, 2);
    divideNumbers(5, 0);
  </script>
</body>
</html>
```

PART 4: REGULAR EXPRESSIONS

1. What are Regular Expressions (Regex)?
Regular expressions are sequences of characters that define search patterns. In JavaScript, they are used for pattern matching, string validation, searching, and text replacement operations.

2. Core Syntax & Methods
- Syntax: Defined using forward slashes (e.g., /pattern/flags) or the RegExp constructor.
- Anchors: ^ (start of string), $ (end of string).
- Quantifiers: * (0 or more), + (1 or more), ? (0 or 1), {n,m} (range).
- Character Classes: \d (digit), \w (word character), \s (whitespace), [a-z] (character set).
- Methods: regex.test(str) returns true/false; str.match(regex) returns matched arrays.

3. Program: Regular Expression Validation

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Regex Demo</title>
</head>
<body>
  <script>
    const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
    const testEmails = ['user@example.com', 'invalid-email', 'test.name@domain.co.in'];

    testEmails.forEach(email => {
      if (emailRegex.test(email)) {
        console.log(`VALID: ${email}`);
      } else {
        console.log(`INVALID: ${email}`);
      }
    });
  </script>
</body>
</html>
```

PART 5: FORM VALIDATION

1. What is Form Validation?

Form validation is the process of ensuring that user input submitted through an HTML form meets specific formatting, completeness, and security criteria before it is sent to a server or processed by client scripts.

2. Client-Side Validation Approaches

- HTML5 Built-in Validation: Using attributes like required, minlength, pattern, and type='email'.
- JavaScript Programmatic Validation: Intercepting the form's submit event using e.preventDefault(), checking conditions, displaying error messages dynamically, and providing immediate user feedback.

3. Program: Complete JavaScript Form Validation

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Form Validation Demo</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 40px; }
    .form-group { margin-bottom: 15px; }
    label { display: block; font-weight: bold; }
    input { padding: 8px; width: 300px; }
    .error { color: red; font-size: 14px; }
    .success { color: green; font-size: 14px; }
  </style>
</head>
<body>
  <h2>Registration Form Validation</h2>
  <form id="signupForm">
    <div class="form-group">
      <label for="username">Username:</label>
      <input type="text" id="username">
      <div id="userError" class="error"></div>
    </div>
    <div class="form-group">
      <label for="emailInput">Email:</label>
      <input type="text" id="emailInput">
      <div id="emailError" class="error"></div>
    </div>
    <button type="submit">Register</button>
    <div id="successMsg" class="success"></div>
  </form>
```

```
  <script>
    const form = document.getElementById('signupForm');
    form.addEventListener('submit', (e) => {
      e.preventDefault();
      let isValid = true;

      const username = document.getElementById('username').value.trim();
      const email = document.getElementById('emailInput').value.trim();

      const userError = document.getElementById('userError');
      const emailError = document.getElementById('emailError');
      const successMsg = document.getElementById('successMsg');

      userError.textContent = '';
      emailError.textContent = '';
      successMsg.textContent = '';

      if (username.length < 3) {
        userError.textContent = 'Username must be at least 3 characters long.';
        isValid = false;
      }

      const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
      if (!emailRegex.test(email)) {
        emailError.textContent = 'Please enter a valid email address.';
        isValid = false;
      }

      if (isValid) {
        successMsg.textContent = 'Form submitted successfully!';
      }
    });
  </script>
</body>
</html>
```