Module 2

Q. What is mean by Event handling in JavaScript explain it with example.

1. Event handling in JavaScript refers to the process of detecting and responding to user interactions or other events that occur on a web page. An **event** can be any action or occurrence that takes place, such as:

- A user clicking a button
- Pressing a key on the keyboard
- Hovering over an element
- Scrolling the page
- Submitting a form
- Loading a page

2. **Event handling** allows JavaScript to react to these events by executing a block of code, known as an **event handler** or **event listener**.

How Event Handling Works:

1. **Event**: An action or occurrence recognized by JavaScript.
2. **Event Listener**: A function or method that waits for a specific event to happen on a specific element.
3. **Event Handler**: A function that defines what happens when the event occurs.

JavaScript provides the ability to add event listeners to elements and define how the browser should respond when a specific event is triggered.

Common JavaScript Events:

- click: Fired when an element is clicked.
- mouseover: Triggered when the user hovers the mouse over an element.
- keydown: Triggered when a key is pressed on the keyboard.
- submit: Triggered when a form is submitted.
- load: Fired when a web page has fully loaded.

Example :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Event Handling in JavaScript</title>
</head>
```

```
<body>
  <h1>Event Handling Example</h1>
  <button id="myButton">Click Me</button>
  <p id="result"></p>
  <script>
    const button = document.getElementById('myButton');
    function buttonClickHandler() {
      document.getElementById('result').innerHTML = "Button was clicked!";
    }
    button.addEventListener('click', buttonClickHandler);
  </script>
</body>
</html>
```

Q. What is the difference between class selector and ID selector.

ID Attribute	**Class Attribute**
Uniquely identifies one element.	Can be applied to multiple elements.
Primarily used for styling or JavaScript.	Also used for styling or JavaScript.
Only one element can have a specific ID.	Multiple elements can share the same class.
Written as id="example".	Written as class="example".
Accessed in CSS with #example selector.	Accessed in CSS with .example selector.
Often used for unique page elements.	Commonly used for styling groups of elements.

Q. Explain built in objects in JavaScript.

In JavaScript, built-in objects are predefined objects that provide functionality and features that you can use without needing to define them yourself. These objects are essential for working with data types, managing control flow, handling errors, and performing various utility functions.

1. Object

- In JavaScript, an object is a collection of related data and functions, known as properties and methods, respectively. Properties are “key: value” pairs that store data, while methods are functions associated with the object that can manipulate its properties. The core object from which all other objects inherit. It can hold any key-value pairs. Used to create and manage objects.

Example : const person = { name: 'Alice', age: 30 };

2. Array

- An array in JavaScript is a data structure used to store multiple values in a single variable. It can hold various data types and allows for dynamic resizing. Elements are accessed by their index, starting from 0. Ideal for working with collections of items.

Example : const numbers = [1, 2, 3, 4];

3. Function

- A function in JavaScript is a reusable block of code that performs a specific task. You define it once, and then you can run (or "call") it whenever you need that task done in your program. Functions are first-class objects that can be stored in variables, passed as arguments, and returned from other functions.

Example :

```
const add = function(x, y) {
  return x + y;
};
```

4. String

- JavaScript String is a sequence of characters, typically used to represent text. It is enclosed in single or double quotes and supports various methods for text manipulation. Used for manipulating text data.

Example : const greeting = 'Hello, World!';

5. Number

- JavaScript numbers are primitive data types and, unlike other programming languages, you don't need to declare different numeric types like int, float, etc. JavaScript numbers are always stored in double-precision 64-bit binary format IEEE 754. This format stores numbers in 64 bits. Used for mathematical operations.

Example : const age = 25;

6. Boolean

- JavaScript Boolean represents true or false values. It's used for logical operations, condition testing, and variable assignments based on conditions. Values like 0, NaN, empty strings, undefined, and null are false; non-empty strings, numbers other than 0, objects, and arrays are true.

Example : const isActive = true;

7. Date

- The JavaScript Date object represents a specific moment in time, measured in milliseconds since the Unix Epoch (January 1, 1970). It's crucial for working with date and time in web applications, providing methods for tasks like date manipulation, formatting, and calculations**.**

Example : const now = new Date();

8. RegExp

- Represents regular expressions for pattern matching in strings. Used for string searching and manipulation.

Example : const regex = /[a-z]/;

9. Error

- Represents runtime errors that occur in the script. Used for error handling and debugging.

Example : const error = new Error('An error occurred');

10. Map

- A collection of keyed data items, similar to an object, but the keys can be of any type. Useful for associating values with unique keys.

Example :

const map = new Map();

map.set('key', 'value');

Q. What are the benefits of using JSON over XML data?

1. Both JSON and XML can be used to receive data from a web server.

2. Both JSON and XML can be fetched with an XMLHttpRequest

3. JSON is Like XML Because

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages

4. JSON is Unlike XML Because

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays

5. XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

Why JSON is Better Than XML

XML is much more difficult to parse than JSON.
JSON is parsed into a ready-to-use JavaScript object.

For AJAX applications, JSON is faster and easier than XML

Using JSON

- Fetch a JSON string
- JSON.Parse the JSON string

JSON Example

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

Using XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables

XML Example

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

Q. Explain how javaScript can hide HTML elements with suitable example.

JavaScript can hide HTML elements using various methods, typically by manipulating the CSS styles of those elements. The most common approach is to set the display or visibility properties of an element to either none or hidden, respectively.

Method 1: Using style.display

The style.display property controls the layout of the element. Setting it to none removes the element from the document flow, effectively hiding it.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Hide Element Example</title>
</head>
<body>
   <h1 id="myElement">Hello, I am visible!</h1>
   <button onclick="hideElement()">Hide Element</button>
   <script>
      function hideElement() {
         const element = document.getElementById('myElement');
         element.style.display = 'none'; // Hides the element
      }
   </script>
</body>
</html>
```

Method 2: Using style.visibility

The style.visibility property can also be used to hide elements, but unlike display: none, it retains the space in the layout. Setting it to hidden makes the element invisible, but the space it occupies is still preserved.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Hide Element Example</title>
</head>
```

```
<body>
  <h1 id="myElement">Hello, I am visible!</h1>
  <button onclick="hideElement()">Hide Element</button>
  <button onclick="showElement()">Show Element</button>
  <script>
    function hideElement() {
      const element = document.getElementById('myElement');
      element.style.visibility = 'hidden'; // Hides the element but keeps space
    }
    function showElement() {
      const element = document.getElementById('myElement');
      element.style.visibility = 'visible'; // Shows the element
    }
  </script>
</body>
</html>
```

Method 3: Using CSS Classes
Another approach is to define CSS classes for hiding elements and toggle those classes using JavaScript. This is a more maintainable way to manage styles.
Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hide Element Example</title>
  <style>
    .hidden {
      display: none; /* Hides the element */
    }
  </style>
</head>
<body>
  <h1 id="myElement">Hello, I am visible!</h1>
  <button onclick="toggleElement()">Toggle Element</button>
  <script>
    function toggleElement() {
      const element = document.getElementById('myElement');
      element.classList.toggle('hidden'); // Toggles the 'hidden' class
    }
  </script>
</body>
</html>
```

Q. Differentiate Between JSON and XML

JSON	XML
It is JavaScript Object Notation	It is Extensible markup language
It is based on JavaScript language.	It is derived from SGML.
It is a way of representing objects.	It is a markup language and uses tag structure to represent data items.
It does not provides any support for namespaces.	It supports namespaces.
It supports array.	It doesn't supports array.
Its files are very easy to read as compared to XML.	Its documents are comparatively difficult to read and interpret.
It doesn't use end tag.	It has start and end tags.
It is less secured.	It is more secured than JSON.
It doesn't supports comments.	It supports comments.
It supports only UTF-8 encoding.	It supports various encoding.

Q. Write a JavaScript code to accept a name and password from user and validate the data as follows:-

1. **Name should not be empty**
2. **Password should not be less than 6 characters**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form Validation Example</title>
  <style>
    /* Simple styling for error messages */
    .error {
      color: red;
    }
  </style>
</head>
<body>

  <h2>User Registration Form</h2>
```

```
<!-- Simple form to accept name and password -->
<form id="userForm" onsubmit="return validateForm()">
  <!-- Name Field -->
  <label for="name">Name:</label>
  <input type="text" id="name" name="name"><br><br>

  <!-- Password Field -->
  <label for="password">Password:</label>
  <input type="password" id="password" name="password"><br><br>

  <!-- Error Message Display -->
  <p id="errorMsg" class="error"></p>

  <!-- Submit Button -->
  <input type="submit" value="Submit">
</form>

<!-- JavaScript for validation -->
<script>
  // Function to validate form input
  function validateForm() {
    // Get the input fields
    var name = document.getElementById('name').value;
    var password = document.getElementById('password').value;
    var errorMsg = document.getElementById('errorMsg');

    // Clear previous error message
    errorMsg.innerHTML = "";

    // Check if the name field is empty
    if (name === "") {
      errorMsg.innerHTML = "Name cannot be empty.";
      return false; // Prevent form submission
    }

    // Check if the password is less than 6 characters
    if (password.length < 6) {
      errorMsg.innerHTML = "Password must be at least 6 characters long.";
      return false; // Prevent form submission
    }

    // If both validations pass, form will submit
    return true;
  }
  </script>

</body>
</html>
```

Q. Write a JavaScript that reads ten numbers and displays the count of negative numbers, the count of positive numbers and the count of zero from the list.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Count Positive, Negative, and Zero</title>
</head>
<body>

  <h2>Enter 10 Numbers</h2>
  <form onsubmit="return countNumbers()">
    <!-- Create 10 number input fields dynamically -->
    <input type="number" id="num1"> <input type="number" id="num2"> <input
type="number" id="num3">
    <input type="number" id="num4"> <input type="number" id="num5"> <input
type="number" id="num6">
    <input type="number" id="num7"> <input type="number" id="num8"> <input
type="number" id="num9">
    <input type="number" id="num10"><br><br>
    <input type="submit" value="Count">
  </form>

  <p id="result"></p>

  <script>
    function countNumbers() {
      let pos = 0, neg = 0, zero = 0;
      for (let i = 1; i <= 10; i++) {
        const num = parseInt(document.getElementById('num' + i).value);
        num > 0 ? pos++ : num < 0 ? neg++ : zero++;
      }
      document.getElementById('result').innerHTML = `Positive: ${pos}, Negative: ${neg}, Zero: 
${zero}`;
      return false; // Prevent form submission
    }
  </script>

</body>
</html>
```

Q. Write a JavaScript code to check password and confirm password are same or not.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Password Confirmation</title>
</head>
<body>
  <h2>Password Confirmation</h2>

  <!-- Form with Password and Confirm Password fields -->
  <form onsubmit="return checkPassword()">
    <!-- Password field -->
    <label for="password">Password:</label>
    <input type="password" id="password"><br><br>

    <!-- Confirm Password field -->
    <label for="confirmPassword">Confirm Password:</label>
    <input type="password" id="confirmPassword"><br><br>

    <!-- Submit button -->
    <input type="submit" value="Submit">

    <!-- Error message display -->
    <p id="message"></p>
  </form>
  <!-- JavaScript code to check if passwords match -->
  <script>
    function checkPassword() {
      const password = document.getElementById("password").value;
      const confirmPassword = document.getElementById("confirmPassword").value;
      const message = document.getElementById("message");
      // Check if passwords match
      if (password === confirmPassword) {
        message.style.color = "green";
        message.innerHTML = "Passwords match!";
        return true;  // Allow form submission
      } else {
        message.style.color = "red";
        message.innerHTML = "Passwords do not match!";
        return false; // Prevent form submission
      }
    }
  </script>
</body>
</html>
```

Q. Write a code to drag an image from outside the box and drop it inside the box.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Drag and Drop Image</title>
    <style>
        #dropZone {
            width: 300px;
            height: 200px;
            border: 2px dashed #ccc;
            display: flex;
            align-items: center;
            justify-content: center;
            margin: 20px auto;
            background-color: #f9f9f9;
        }
        img {
            width: 100px;
            cursor: grab;
        }
    </style>
</head>
<body>
    <h2>Drag and Drop Image</h2>
    <img id="draggableImage" src="https://via.placeholder.com/100" draggable="true"
alt="Draggable Image">
    <div id="dropZone">Drop Here</div>

    <script>
        const img = document.getElementById("draggableImage");
        const dropZone = document.getElementById("dropZone");

        img.addEventListener("dragstart", (event) => {
            event.dataTransfer.setData("text/plain", event.target.src);
        });

        dropZone.addEventListener("dragover", (event) => {
            event.preventDefault();
        });

        dropZone.addEventListener("drop", (event) => {
            event.preventDefault();
            const imageSrc = event.dataTransfer.getData("text/plain");
            const newImg = document.createElement("img");
            newImg.src = imageSrc;
            dropZone.appendChild(newImg);
        });
    </script>
</body>
</html>
```

Q. Create a Form which has following fields 'Name', 'Age', 'Email ID', 'Password'. Using JavaScript validate each field as follows:

- **i. Name Should be between A-z**
- **ii. Age should be between 0-100**
- **iii. Email id should contain '@'**
- **iv. Password should contain 1 Upper case, 1 Digit, 1 Special character and length should be minimum 8.**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Form Validation</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        .error {
            color: red;
            margin-top: 5px;
        }
    </style>
</head>
<body>

    <h2>Registration Form</h2>
```

```
  <form id="registrationForm" onsubmit="return validateForm()">
    <label for="name">Name:</label>
    <input type="text" id="name" required>
    <div id="nameError" class="error"></div>

    <label for="age">Age:</label>
    <input type="number" id="age" required>
    <div id="ageError" class="error"></div>

    <label for="email">Email ID:</label>
    <input type="email" id="email" required>
    <div id="emailError" class="error"></div>

    <label for="password">Password:</label>
    <input type="password" id="password" required>
    <div id="passwordError" class="error"></div>

    <input type="submit" value="Submit">
  </form>

  <script>
    function validateForm() {
      // Clear previous error messages
      document.getElementById("nameError").innerText = "";
      document.getElementById("ageError").innerText = "";
      document.getElementById("emailError").innerText = "";
      document.getElementById("passwordError").innerText = "";

      // Get field values
      const name = document.getElementById("name").value;
      const age = document.getElementById("age").value;
      const email = document.getElementById("email").value;
      const password = document.getElementById("password").value;

      let isValid = true;

      // Validate Name
      const nameRegex = /^[A-Za-z]+$/;
      if (!nameRegex.test(name)) {
        document.getElementById("nameError").innerText = "Name should contain only
letters.";
        isValid = false;
      }

      // Validate Age
      if (age < 0 || age > 100) {
            document.getElementById("ageError").innerText = "Age should be between 0 and
100.";
            isValid = false;
         }

         // Validate Email
         if (!email.includes('@')) {
            document.getElementById("emailError").innerText = "Email ID should contain '@'.";
            isValid = false;
         }

         // Validate Password
         const passwordRegex = /^(?=.*[A-Z])(?=.*\d)(?=.*[\W_]).{8,}$/; // At least one
uppercase, one digit, one special character, and minimum 8 characters
         if (!passwordRegex.test(password)) {
            document.getElementById("passwordError").innerText = "Password must contain at
least 1 uppercase letter, 1 digit, 1 special character, and be at least 8 characters long.";
            isValid = false;
         }

         return isValid; // If all validations are successful, submit the form
      }
   </script>

</body>
</html>
```