**Web Technologies Laboratory 03**

**Name: Aditya N. Patil**

**Panel:** A

**Roll No.:** PA-30

**Batch:** A2

**PRN No.:** 1032211573



**Aim: Client-side Form Validations using JavaScript, DOM real-time update, JQuery to develop Ajax based applications.**

**Aim:** Write a program to perform following form validations using JavaScript:

a) All fields mandatory,

b) Phone number, Email Address, Zip code Validation etc.

Include JavaScript to access and manipulate Document Object Model (DOM) objects in an HTML web page.

Include JQuery to develop to develop your application as an Ajax based application.

**Objectives:**

1. To understand what is form validation.
2. To learn basic functioning of DOM objects.
3. To learn how to apply various techniques to implement it.

**Theory:**

1. Different types of form validations.

Form validation is an essential part of web development and user experience. It ensures that user-submitted data is accurate, complete, and secure. There are various types of form validations you can implement in web applications, depending on your specific needs. Here are some common types of form validations:

* **Required Field Validation:** This type of validation ensures that users fill out essential fields in a form, such as name, email, or password. It prevents the submission of blank or incomplete forms.
* **Email Validation:** Checks if the input in an email field follows a valid email format. It typically verifies that the email contains an "@" symbol and a valid domain.
* **Password Validation:** Ensures that passwords meet certain criteria, such as a minimum length, the inclusion of uppercase and lowercase letters, numbers, and special characters.
* **Numeric Validation:** Used for fields that should only contain numbers, such as phone numbers or credit card numbers. It checks if the input consists of digits and may enforce a specific format.
* **Date Validation:** Validates date inputs to ensure they follow a particular format (e.g., MM/DD/YYYY) and may also check for valid dates (e.g., February 30 is not a valid date).
* **URL Validation:** Verifies that the input in a URL field is a valid web address with the correct format, including the "http://" or "https://" prefix.
* **Regular Expression (Regex) Validation:** Allows you to define custom validation patterns using regular expressions. This is useful for complex validation requirements, like validating phone numbers with specific formats or custom input patterns.
* **File Upload Validation:** Ensures that uploaded files meet specific criteria, such as file type (e.g., only allow image files) and size limitations.
* **Character Limit Validation:** Limits the number of characters a user can input in a field, such as a comment box or a tweet.
* **Custom Validation Messages:** Provides user-friendly error messages to guide users when their input doesn't meet the required criteria, making it easier for them to correct mistakes.
* **Confirmation Fields:** Used for password confirmation or email confirmation fields. Ensures that the user enters the same information in both fields to prevent typos or mistakes.
* **Captcha and reCAPTCHA:** Helps prevent automated submissions by requiring users to solve puzzles or prove they are not bots.
* **Server-Side Validation:** While most validation occurs on the client side using JavaScript, it's crucial to perform additional validation on the server to prevent tampering or malicious input. This includes database checks and security validations.
* **Cross-Field Validation:** Validates data across multiple fields. For example, **ensuring that a start date is earlier than an end date in a date range picker.**
* **Database Validation:** Checks if the input data already exists in a database, preventing duplicates or enforcing unique values.
* **Geographical Validation:** Verifies that location-based inputs, such as postal codes or addresses, are valid for a specific region or country.
* **Time Validation:** Ensures that time inputs, such as appointment times or event start times, adhere to a specified format and logic.

1. HTML Document Object Model.

The HTML Document Object Model, often referred to as the HTML DOM, is a programming interface and representation of an HTML document's structure. It provides a structured way to access, manipulate, and interact with HTML documents using programming languages like JavaScript. The HTML DOM represents an HTML document as a tree-like structure of objects, where each object corresponds to an HTML element, attribute, or text content within the document.

Here are some key points and concepts related to the HTML DOM:

* **Hierarchy:** The HTML DOM represents the document as a tree structure with a single root node called the "document" object. The document object contains all other elements and nodes as its children. Each HTML element, such as headings, paragraphs, images, forms, and links, is represented as a node in the tree.
* **Nodes:** There are several types of nodes in the HTML DOM, including:
* **Element Nodes**: Represent HTML elements (e.g., <div>, <p>, <a>).
* **Text Nodes:** Represent the text content within elements.
* **Attribute Nodes:** Represent attributes of HTML elements.
* **Comment Nodes:** Represent comments within the HTML code.
* **Document Nodes:** Represent the document itself.
* **Accessing Elements:** You can access elements in the HTML DOM using JavaScript. Common methods for accessing elements include getElementById, getElementsByClassName, getElementsByTagName, and more.
* **Manipulation:** You can modify the content, attributes, and structure of HTML elements using the DOM. For instance, you can change the text of an element, add or remove elements, modify attributes, and respond to user interactions.
* **Traversal:** You can navigate the DOM tree by moving from one node to another. Common traversal methods include accessing parent nodes, child nodes, and sibling nodes. This is useful for moving around the document and targeting specific elements.
* **Events:** The DOM allows you to attach event listeners to elements to respond to user interactions like clicks, mouse movements, and keyboard inputs. This enables you to create interactive web applications.
* **Dynamic Updates:** One of the most significant advantages of the HTML DOM is its ability to dynamically update the content of web pages without requiring a full page reload. This is commonly used in modern web applications to create responsive and interactive user interfaces.
* **Cross-Browser Compatibility:** The HTML DOM is supported by all major web browsers, making it a reliable way to interact with and manipulate web pages across different platforms.

1. What is JQuery? Write various JQuery Selectors.

jQuery is a fast, small, and feature-rich JavaScript library. It simplifies various tasks related to HTML document traversal, manipulation, event handling, animation, and AJAX (Asynchronous JavaScript and XML) interactions. jQuery is designed to make client-side scripting on web pages easier and more efficient by providing a concise and consistent API for interacting with the Document Object Model (DOM).

One of jQuery's core features is its support for selectors. jQuery selectors allow you to efficiently select and manipulate HTML elements from a web page. Here are some common jQuery selectors:

* **Element Selector (element):** Selects all HTML elements with the specified element name. For example, to select all <p> elements on a page, you can use:

$("p")

* **ID Selector (#id):** Selects an element with the specified id attribute. For example:

$("#myId")

* **Class Selector (.class):** Selects elements with the specified class attribute. For example:

$(".myClass")

* **Attribute Selector ([attribute]):** Selects elements with a specific attribute. For example, to select all elements with a data-toggle attribute:

$("[data-toggle]")

* **Attribute Value Selector ([attribute=value]):** Selects elements with a specific attribute and attribute value. For example, to select all <input> elements with a type attribute set to "text":

$("input[type='text']")

* **Multiple Selector (selector1, selector2, ...):** Allows you to combine multiple selectors to select elements that match any of the specified conditions. For example:

$("h1, h2, h3")

* **Descendant Selector (ancestor descendant):** Selects all descendants of the specified ancestor element. For example, to select all <span> elements within a <div> with the ID "myDiv":

$("#myDiv span")

* **Child Selector (parent > child):** Selects all immediate child elements that are direct descendants of a specified parent element. For example:

$("ul > li")

* **Siblings Selector (prev ~ siblings):** Selects all elements that are siblings of a specified element and have the same parent. For example, to select all <p> elements that are siblings of an <h1> element:

$("h1 ~ p")

* **Filter Selector (selector:filter):** Allows you to apply filters to further refine your selection. For example, to select even rows in a table with the class "table":

$(".table tr:even")

* **First Selector (:first):** Selects the first element in a set of matched elements. For example:

$("li:first")

* **Last Selector (:last):** Selects the last element in a set of matched elements. For example:

$("li:last")

**FAQ:**

1. Write 3 reasons why Form validations are important.

Form validations are crucial for web applications and websites for several important reasons:

1. **Data Accuracy and Integrity:** Form validations help ensure that the data submitted by users is accurate and meets the expected format or criteria. Without validations, users might enter incorrect or incomplete information, leading to data inconsistencies and errors in the system. Validations prevent these issues by enforcing data correctness.
2. **Enhanced User Experience:** Validations contribute significantly to a positive user experience. When users encounter clear and informative validation messages, they can quickly correct errors and understand the required data format. This reduces frustration and confusion, making the interaction with the website more user-friendly.
3. **Security and Data Integrity:** Form validations play a vital role in web application security. They help prevent malicious input, such as SQL injection, cross-site scripting (XSS), and other types of attacks. Properly validated forms ensure that only safe and expected data is processed, reducing the risk of security vulnerabilities and data breaches.
4. Give an example of how to modify an attribute value using DOM.

You can modify an attribute value of an HTML element using the Document Object Model (DOM) in JavaScript. Here's an example of how to do this:

Let's say you have an HTML element with an id attribute that you want to modify:

<!DOCTYPE html>

<html>

<head>

<title>Modify Attribute Example</title>

</head>

<body>

<p id="myParagraph">This is a paragraph with an ID.</p>

<button onclick="modifyAttribute()">Modify Attribute</button>

<script>

function modifyAttribute() {

// Get a reference to the paragraph element by its ID

var paragraph = document.getElementById("myParagraph");

// Modify the 'id' attribute value

paragraph.id = "newParagraphId";

}

</script>

</body>

</html>

In this example:

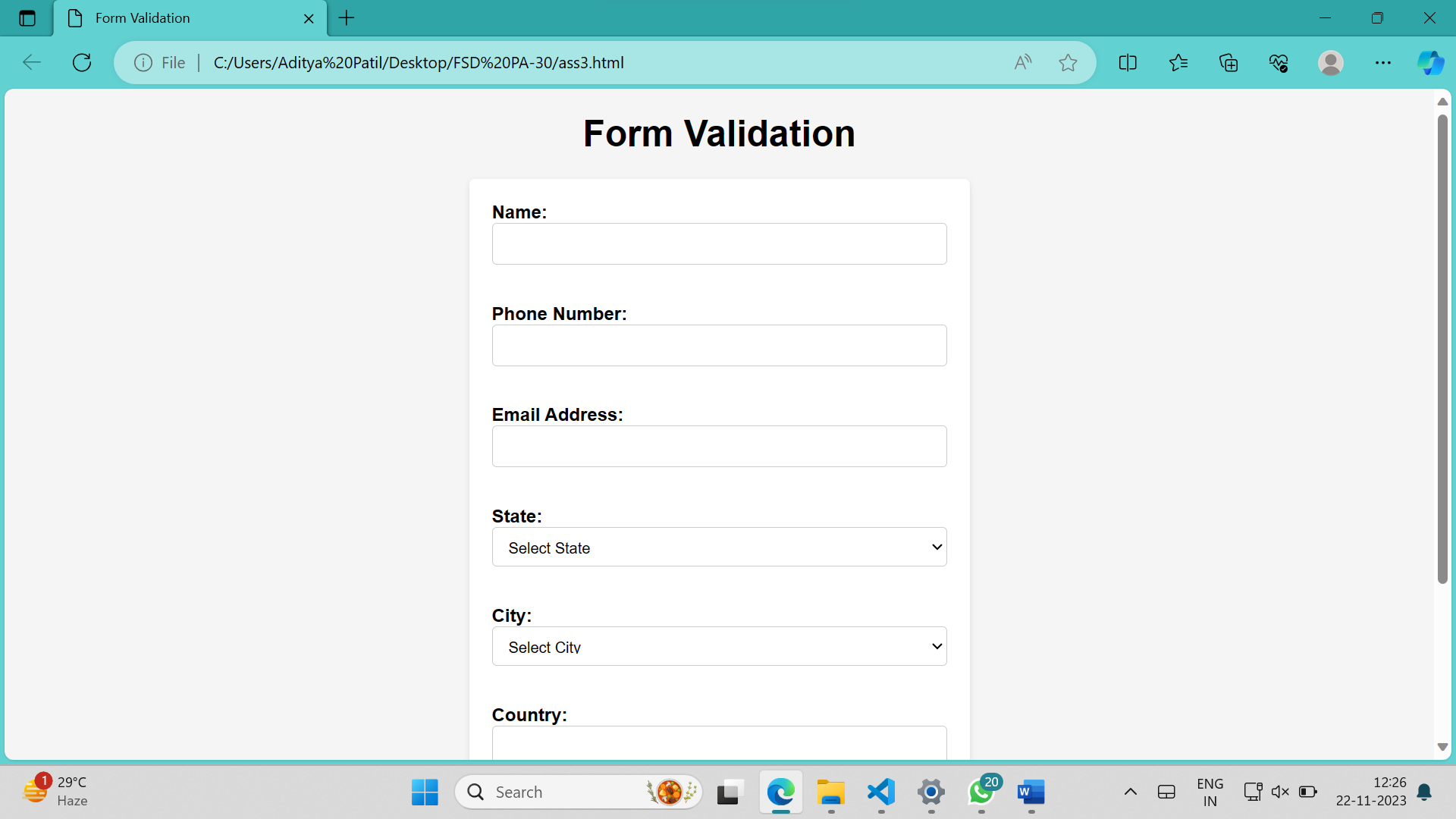
* We have an HTML paragraph element with the id attribute set to "myParagraph."
* There's a button with an onclick attribute that calls a JavaScript function called modifyAttribute() when clicked.
* Inside the modifyAttribute() function:
* We use document.getElementById("myParagraph") to get a reference to the paragraph element with the ID "myParagraph."
* We then modify the id attribute of the paragraph element by setting it to "newParagraphId."

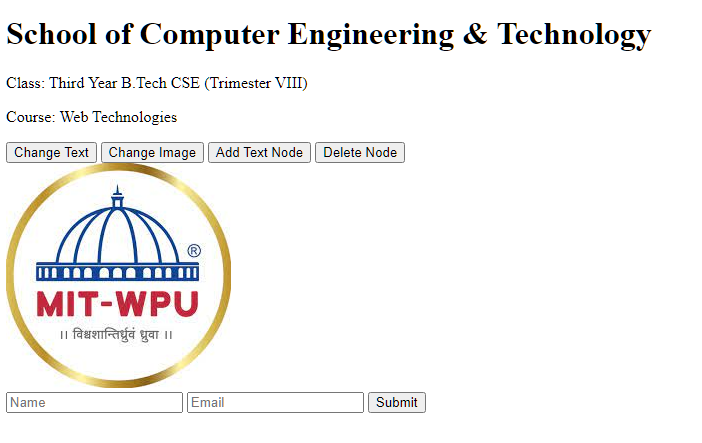
1. What is jQuery Ajax?

jQuery Ajax (short for "Asynchronous JavaScript and XML") is a set of methods and functionalities provided by the jQuery library to simplify making asynchronous HTTP requests to a server from a web page. Ajax allows web applications to send and receive data from the server without requiring a full page reload. This technology has become fundamental in modern web development for building interactive and responsive web applications.

Here are some key features and concepts related to jQuery Ajax:

* **Asynchronous Communication:** With jQuery Ajax, you can send and receive data from the server without blocking the user's interaction with the web page. This asynchronous behavior ensures that the user interface remains responsive while data is being exchanged with the server.
* **HTTP Methods:** jQuery Ajax supports various HTTP methods, including GET, POST, PUT, DELETE, and more. These methods are used to specify the type of request you want to make to the server.
* **Data Serialization:** jQuery simplifies the process of serializing form data or JSON objects and sending them to the server in the appropriate format. It can also parse JSON responses from the server, making it easy to work with structured data.
* **Callbacks:** jQuery Ajax relies heavily on callback functions to handle different stages of the asynchronous request. Common callbacks include success, error, and complete. You can specify these callbacks to perform actions based on the outcome of the request.
* **Cross-Origin Requests:** jQuery Ajax provides options for making cross-origin requests, which are necessary when accessing resources from different domains. This is essential for fetching data from third-party APIs or external servers.
* **Promises:** jQuery Ajax methods return a Promise object, which allows you to use promises and chaining to manage asynchronous operations more elegantly. This is particularly useful when dealing with multiple Ajax requests or complex workflows.
* **Error Handling:** jQuery Ajax includes built-in error handling mechanisms that let you gracefully handle network errors, server errors, and unexpected issues during the request/response process.



****

**Github link-**