W.T. Assignment No. 3

❖ 2 Marks Questions

Q.1.) Explain data types in JavaScript.

Ans :- JavaScript has **two** main categories of data types:

1. **Primitive Data Types** (immutable, stored by value):
   - **Number** – Represents numeric values (e.g., 42, 3.14).
   - **String** – Represents text (e.g., "Hello", 'World').
   - **Boolean** – Represents true or false.
   - **Undefined** – A variable that has been declared but not assigned a value.
   - **Null** – Represents an intentional absence of value.
   - **BigInt** – For large integers beyond Number limits.
   - **Symbol** – A unique and immutable value, mainly used for object properties.
2. **Non-Primitive (Reference) Data Types** (mutable, stored by reference):
   - **Object** – A collection of key-value pairs (e.g., { name: "John", age: 25 }).
   - **Array** – A special type of object for ordered lists (e.g., [1, 2, 3]).
   - **Function** – A callable block of code.

Q.2.) Write a simple program in JavaScript to validate the email-id.

Ans :- Here's a simple JavaScript program to validate an email ID using a regular expression:

JavaScript Email Validation Program

```
function validateEmail(email) {
    // Regular expression for basic email validation
    let regex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

    if (regex.test(email)) {
        console.log("Valid Email ID");
        return true;
    } else {
        console.log("Invalid Email ID");
        return false;
    }
}

// Example usage
let email1 = "test@example.com";
let email2 = "invalid-email";

validateEmail(email1); // Output: Valid Email ID
validateEmail(email2); // Output: Invalid Email ID
```

Q.3.) How to write function using Java Script? Give Example.

Ans :- In JavaScript, functions are used to execute a block of code when called. There are different ways to define functions:

1. Function Declaration (Regular Function)

```
function greet(name) {
   return "Hello, " + name + "!";
}

console.log(greet("Alice")); // Output: Hello, Alice!
```

Explanation:

- Uses the function keyword.
- Can be called before declaration due to **hoisting**.

2. Function Expression

```
const greet = function(name) {
    return "Hello, " + name + "!";
};

console.log(greet("Bob")); // Output: Hello, Bob!
```

Explanation:

- **Function is assigned to a variable.**
- **Not hoisted like function declarations.**

3. Arrow Function (ES6)

```
const greet = (name) => "Hello, " + name + "!";

console.log(greet("Charlie")); // Output: Hello, Charlie!
```

Explanation:

- **Concise syntax, useful for short functions.**
- **this behaves differently compared to regular functions.**

4. Function with Default Parameter

```
function greet(name = "Guest") {
    return "Hello, " + name + "!";
}

console.log(greet()); // Output: Hello, Guest!
console.log(greet("David")); // Output: Hello, David!
```

Explanation:

- **Uses a default value ("Guest") if no argument is passed.**

5. Function with Multiple Parameters

```
function add(a, b) {
    return a + b;
}

console.log(add(5, 3)); // Output: 8
```

Explanation:

- **Accepts multiple arguments and returns the sum.**

6. Anonymous Function (Used in Callbacks)

```
setTimeout(function() {
    console.log("This runs after 2 seconds");
}, 2000);
```

Explanation:

- **Function without a name, used as a callback.**

7. Immediately Invoked Function Expression (IIFE)

```
(function() {
    console.log("This function runs immediately!");
})();
```

Explanation:

- **Runs immediately without being explicitly called.**

Q.4.) Discuss JavaScript objects in detail with suitable examples.

Ans :- In JavaScript, an **object** is a collection of key-value pairs where **keys** are strings (or Symbols) and **values** can be any data type, including other objects and functions. Objects allow us to store and manage related data efficiently.

Creating an Object

1. **Using Object Literal (Most Common Method)**

```
let person = {
    name: "John",
    age: 30,
    isMarried: false
};
```

2. Using the new Object() Constructor

```
let person = new Object();
person.name = "John";
person.age = 30;
person.isMarried = false;
```

Accessing Object Properties

- **Dot Notation** (Preferred)

  ```javascript
  console.log(person.name); // Output: John
  ```

- **Bracket Notation** (Used for dynamic keys)

  ```javascript
  console.log(person["age"]); // Output: 30
  ```

Adding & Modifying Properties

```javascript
person.city = "New York"; // Adding a new property
person.age = 31; // Modifying an existing property
console.log(person);
```

Deleting a Property

```javascript
delete person.isMarried;
console.log(person);
```

Object Methods (Functions Inside Objects)

```javascript
let car = {
    brand: "Toyota",
    model: "Corolla",
    start: function() {
        return "Car started";
    }
};

console.log(car.start()); // Output: Car started
```

Looping Through an Object

Using `for...in` loop:

```javascript
for (let key in person) {
    console.log(key + ": " + person[key]);
}
```

Nested Objects

```javascript
let student = {
    name: "Alice",
    marks: {
        math: 90,
        science: 85
    }
};

console.log(student.marks.math); // Output: 90
```

Q.5. Write a Java script to convert temperature from Celsius to Fahrenheit of the given number.

Ans :- Here's a simple JavaScript program to convert temperature from **Celsius** to **Fahrenheit**:

```
function celsiusToFahrenheit(celsius) {
    let fahrenheit = (celsius * 9/5) + 32;
    return fahrenheit;
}

// Example usage
let celsius = 25;
console.log(`${celsius}°C is equal to ${celsiusToFahrenheit(celsius)}°F`);
```

Formula Used:

$$^\circ F = (^\circ C \times \frac{9}{5}) + 32$$

Example Output:

```
25°C is equal to 77°F
```

Q.6.) Write a Java script program to create Popup box, alert and confirm box.

Ans :- Here's a JavaScript program to create **Popup Box, Alert Box, and Confirm Box**:

```javascript
// 1. Alert Box
alert("This is an Alert Box!");

// 2. Prompt Box (Takes user input)
let name = prompt("Enter your name:");
if (name) {
    alert("Hello, " + name + "!");
}

// 3. Confirm Box (Asks for confirmation)
let confirmAction = confirm("Do you want to proceed?");
if (confirmAction) {
    alert("You clicked OK!");
} else {
    alert("You clicked Cancel!");
}
```

Explanation:

1. alert() → Displays a simple message popup.
2. prompt() → Takes user input and returns it as a string.
3. confirm() → Asks the user for confirmation (OK returns true, Cancel returns false).

Q.7.) Explain JavaScript - HTML DOM Methods for accessing html elements.

Ans :- In JavaScript, **DOM (Document Object Model) methods** are used to access and manipulate HTML elements. Here are the commonly used methods:

1. `getElementById()` (Access by ID)

- Selects an element with a specific `id`.
- **Example:**

javascript Copy Edit

```javascript
let element = document.getElementById("myElement");
element.style.color = "red";
```

html Copy Edit

```html
<p id="myElement">Hello, World!</p>
```

2. `getElementsByClassName()` (Access by Class)

- Returns a collection (HTMLCollection) of elements with a given class.
- **Example:**

javascript Copy Edit

```javascript
let elements = document.getElementsByClassName("myClass");
elements[0].style.color = "blue"; // Changing first element
```

html Copy Edit

```html
<p class="myClass">First Paragraph</p>
<p class="myClass">Second Paragraph</p>
```

3. `getElementsByTagName()` (Access by Tag Name)

- Returns all elements with the given tag name.
- **Example:**

javascript Copy Edit

```javascript
let paragraphs = document.getElementsByTagName("p");
paragraphs[0].style.fontSize = "20px"; // Modifies first `<p>`
```

html Copy Edit

```html
<p>First Paragraph</p>
<p>Second Paragraph</p>
```

4. `querySelector()` (Access First Matching Element)

- Selects the first element that matches a CSS selector.
- **Example:**

javascript Copy Edit

```javascript
let element = document.querySelector(".myClass");
element.style.backgroundColor = "yellow";
```

html Copy Edit

```html
<p class="myClass">Hello</p>
```

5. `querySelectorAll()` (Access All Matching Elements)

- Returns a **NodeList** of all matching elements.
- **Example:**

javascript Copy Edit

```javascript
let elements = document.querySelectorAll(".myClass");
elements.forEach(el => el.style.border = "2px solid black");
```

html Copy Edit

```html
<p class="myClass">One</p>
<p class="myClass">Two</p>
```

Q.8.) Write a program using JavaScript DOM Validate Numeric Input.

Ans :- Here's a **JavaScript DOM program** to validate if the user enters a numeric value in an input field.

JavaScript Numeric Input Validation:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Numeric Input Validation</title>
</head>
<body>

    <label for="numInput">Enter a number:</label>
    <input type="text" id="numInput">
    <button onclick="validateInput()">Validate</button>
    <p id="message"></p>
    <script>
        function validateInput() {
            let input = document.getElementById("numInput").value;
            let message = document.getElementById("message");

            if (!isNaN(input) && input.trim() !== "") {
                message.style.color = "green";
                message.textContent = "Valid numeric input!";
            } else {
                message.style.color = "red";
                message.textContent = "Invalid input! Please enter a number.";
            }
        }
    </script>

</body>
</html>
```

Q.9.) What is Ajax? How AJAX Works?

Ans :- **AJAX (Asynchronous JavaScript and XML)** is a technique that allows web pages to send and receive data from a server asynchronously **without reloading** the page. It enhances user experience by making web applications more dynamic and responsive.

How AJAX Works?

AJAX follows these steps:

1. **User Action** → A user triggers an event (e.g., button click).
2. **Create XMLHttpRequest** → A request is sent to the server using JavaScript.
3. **Server Processes Request** → The server processes and sends a response (data).
4. **Update Web Page** → JavaScript updates the page dynamically without reloading.

Example: Simple AJAX Request

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "https://api.example.com/data", true);
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
        console.log(xhr.responseText); // Process response
    }
};
xhr.send();
```

Benefits of AJAX:

✓ Faster web pages (no full page reload).
✓ Improves user experience.
✓ Reduces server load.

Q.10.) Explain XMLHttpRequest Object Properties. (any 2)

Ans :- The **XMLHttpRequest** object is used in AJAX to send and receive data from a server. Here are **two important properties**:

1. `readyState` (Request State)

- Represents the state of the request.
- Possible values:

Value	State	Description
`0`	UNSENT	Request not initialized
`1`	OPENED	Server connection established
`2`	HEADERS_RECEIVED	Request received
`3`	LOADING	Processing request
`4`	DONE	Request finished and response is ready

Example Usage:

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "https://api.example.com/data", true);
xhr.onreadystatechange = function() {
    console.log("Ready State:", xhr.readyState);
};
xhr.send();
```

2. `status` (HTTP Response Status Code)

- Returns the HTTP status of the response.
- Common values:

Code	Meaning
`200`	OK (Success)
`404`	Not Found
`500`	Server Error

Example Usage:

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "https://api.example.com/data", true);
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4) {
        console.log("Status Code:", xhr.status);
    }
};
xhr.send();
```

Q.11.) Write a program web page can fetch information from an text file with AJAX.

Ans :- Here's a simple **AJAX** program that fetches data from a **text file** and displays it on a web page.

HTML + JavaScript (AJAX) to Fetch Data from a Text File

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>AJAX Fetch Text File</title>
</head>
<body>

    <h2>Fetch Data from a Text File using AJAX</h2>
    <button onclick="loadText()">Fetch Data</button>
    <p id="output"></p>

     <script>
        function loadText() {
            let xhr = new XMLHttpRequest();
            xhr.open("GET", "data.txt", true);

            xhr.onreadystatechange = function() {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    document.getElementById("output").innerText = xhr.responseText;
                }
            };

            xhr.send();
        }
     </script>

</body>
</html>
```

- **Steps to Run the Code:**

1. **Create a text file (data.txt)** with sample content:

```
Hello! This is data fetched using AJAX.
```

2. **Save the HTML file and the text file in the same directory.**
3. **Open the HTML file in a browser** and click the **"Fetch Data"** button.
4. The content of data.txt will be displayed inside the <p> tag.

- **How It Works?**

1. XMLHttpRequest() creates an AJAX request.
2. open("GET", "data.txt", true) prepares a request for data.txt.
3. onreadystatechange checks when data is received (readyState === 4 && status === 200).
4. xhr.responseText retrieves and displays the text file content.

❖ **5 Marks Questions**

Q.1.) Design a webpage to Display XML Data in an HTML Table.

Ans :- Here's a **webpage** that fetches and displays **XML data** inside an **HTML table using AJAX**.

Step 1: Create an XML File (data.xml)

Save this XML file in the same directory as the HTML file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<employees>
    <employee>
        <name>John Doe</name>
        <position>Software Engineer</position>
        <salary>70000</salary>
    </employee>
    <employee>
        <name>Jane Smith</name>
        <position>Project Manager</position>
        <salary>90000</salary>
    </employee>
    <employee>
        <name>Robert Brown</name>
        <position>UI/UX Designer</position>
        <salary>75000</salary>
    </employee>
</employees>
```

Step 2: Create an HTML File (index.html)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Display XML Data in Table</title>
    <style>
        table {
            width: 50%;
            border-collapse: collapse;
            margin-top: 20px;
        }
        th, td {
            border: 1px solid black;
            padding: 10px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
        }
    </style>
</head>
<body>
```

```
 <h2>Employee Data from XML</h2>
 <button onclick="loadXML()">Load Data</button>
 <table id="employeeTable">
     <tr>
         <th>Name</th>
         <th>Position</th>
         <th>Salary ($)</th>
     </tr>
 </table>

 <script>
     function loadXML() {
         let xhr = new XMLHttpRequest();
         xhr.open("GET", "data.xml", true);

            xhr.onreadystatechange = function () {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    let xmlDoc = xhr.responseXML;
                    let table = document.getElementById("employeeTable");

                    let employees = xmlDoc.getElementsByTagName("employee");

                    for (let i = 0; i < employees.length; i++) {
                        let row = table.insertRow();

                        let name = employees[i].getElementsByTagName("name")[0].textContent;
                        let position = employees[i].getElementsByTagName("position")[0].textContent;
                        let salary = employees[i].getElementsByTagName("salary")[0].textContent;

                        row.insertCell(0).textContent = name;
                        row.insertCell(1).textContent = position;
                        row.insertCell(2).textContent = salary;
                    }
                }
            };
            xhr.send();
        }
    </script>

</body>
</html>
```

Q.2.) Explain the logic of using GET or POST in open() of XMLHttpRequest object

Ans :- **Using GET or POST in open() of XMLHttpRequest**

The open() method of the XMLHttpRequest object is used to **initialize a request**. The first parameter specifies the HTTP **method**, which can be either **GET or POST**.

1. GET Method

- **Used to request data from a server** (read-only operations).
- **Parameters are sent in the URL** as query strings.
- **Faster** because it caches responses and has less overhead.
- **Not secure** for sending sensitive data since parameters are visible in the URL.

Example of GET Request:

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "data.txt", true); // Requesting a text file
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
        console.log(xhr.responseText); // Logs the response
    }
};
xhr.send(); // No data in the body for GET requests
```

Example URL with Query String:

```
https://example.com/data?name=John&age=30
```

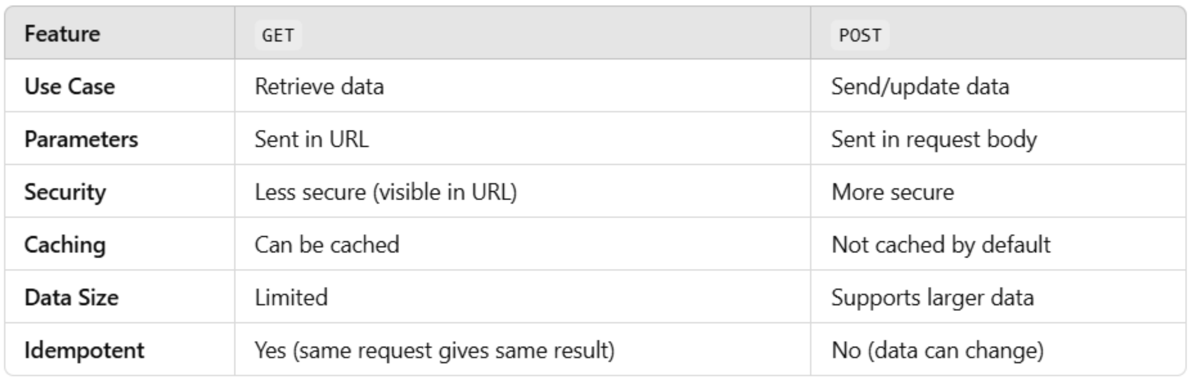
2. POST Method

- **Used to send data to the server** (write/update operations).
- **Data is sent in the request body**, making it more secure.
- **Not cached** by default, so it's better for frequently changing data.
- **Allows larger payloads** compared to GET.

Example of POST Request:

```
let xhr = new XMLHttpRequest();
xhr.open("POST", "submit.php", true);
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
        console.log(xhr.responseText);
    }
};
xhr.send("name=John&age=30"); // Data sent in the request body
```

Comparison Table:



Feature	`GET`	`POST`
Use Case	Retrieve data	Send/update data
Parameters	Sent in URL	Sent in request body
Security	Less secure (visible in URL)	More secure
Caching	Can be cached	Not cached by default
Data Size	Limited	Supports larger data
Idempotent	Yes (same request gives same result)	No (data can change)

Q.3.) What do you mean by asynchronous requests? Explain how AJAX handles it.

Ans :- An **asynchronous request** allows a web page to communicate with a server **without blocking** the execution of other tasks. This means the web page can continue running **without waiting** for the server response.

How AJAX Handles Asynchronous Requests?

AJAX (**Asynchronous JavaScript and XML**) uses the **XMLHttpRequest** object to send and receive data asynchronously.

1. **Creates an XMLHttpRequest object**
2. **Opens a connection (open(method, URL, async))**
3. **Sends a request to the server (send())**
4. **Listens for a response (onreadystatechange)**
5. **Updates the webpage dynamically**

By setting the third parameter of open() to true, AJAX enables asynchronous processing:

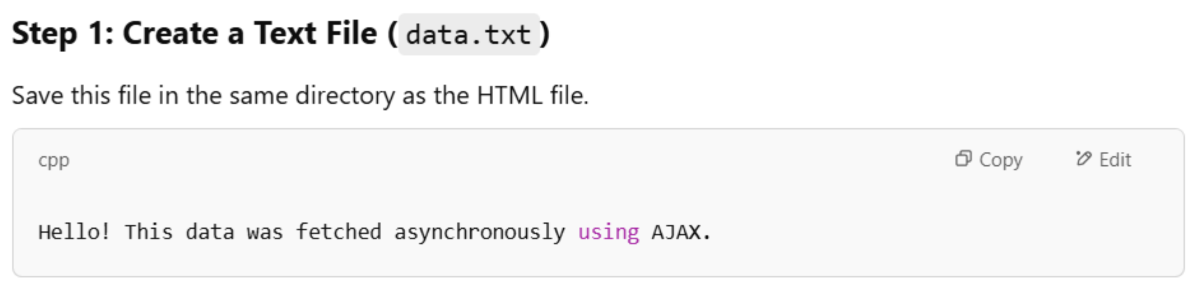
```
let xhr = new XMLHttpRequest();
xhr.open("GET", "data.txt", true); // 'true' makes it asynchronous

xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
        console.log(xhr.responseText); // Processes response without blocking UI
    }
};

xhr.send();
```

Why Use Asynchronous Requests?

✅ **Non-blocking** → The webpage remains responsive.
✅ **Improves Performance** → Other tasks can execute while waiting for a response.
✅ **Better User Experience** → No page reloads, smoother interactions.

Q.4.) Design a webpage to handle asynchronous requests using AJAX on button click event.

Ans :- Here's a simple **webpage** that handles **asynchronous requests using AJAX** when a button is clicked. It fetches data from a **text file** and displays it dynamically without reloading the page.



Step 1: Create a Text File (`data.txt`)

Save this file in the same directory as the HTML file.

```cpp
Hello! This data was fetched asynchronously using AJAX.
```

Step 2: Create an HTML File (index.html)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX Asynchronous Request</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
        button {
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
        }
        p {
            margin-top: 20px;
            font-size: 18px;
            color: blue;
        }
    </style>
</head>
<body>

    <h2>AJAX Asynchronous Request Example</h2>
    <button onclick="fetchData()">Fetch Data</button>
    <p id="output"></p>

    <script>
        function fetchData() {
            let xhr = new XMLHttpRequest();
            xhr.open("GET", "data.txt", true); // 'true' makes it asynchronous

            xhr.onreadystatechange = function() {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    document.getElementById("output").textContent = xhr.responseText;
                }
            };

            xhr.send(); // Sends the request
        }
    </script>

</body>
</html>
```

How It Works?

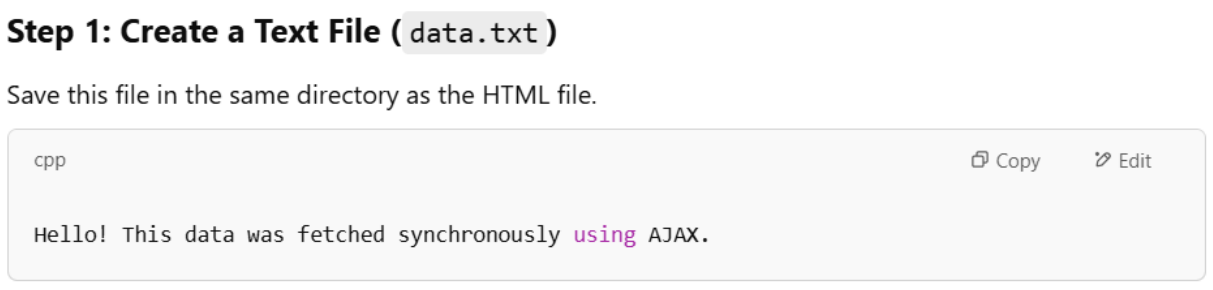
1. **User clicks the "Fetch Data" button**

2. **AJAX (XMLHttpRequest) sends an asynchronous request** to fetch data.txt
3. **onreadystatechange listens for the response**
4. **When data is received, it updates the webpage dynamically** without refreshing

Q.5.) Design a webpage to handle synchronous requests using AJAX on button click event.

Ans :- **Webpage to Handle Synchronous Requests Using AJAX**

By default, **AJAX is asynchronous**, but you can make a request **synchronous** by setting the third parameter of open() to **false**. This means the request will **block execution** until a response is received.



Step 1: Create a Text File (`data.txt`)

Save this file in the same directory as the HTML file.

```cpp
Hello! This data was fetched synchronously using AJAX.
```

Step 2: Create an HTML File (index.html)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Synchronous AJAX Request</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
        button {
            padding: 10px 20px;
            font-size: 16px;
            cursor: pointer;
        }
        p {
            margin-top: 20px;
            font-size: 18px;
            color: green;
        }
    </style>
</head>
<body>

     <h2>Synchronous AJAX Request Example</h2>
     <button onclick="fetchData()">Fetch Data</button>
     <p id="output"></p>

     <script>
         function fetchData() {
             let xhr = new XMLHttpRequest();
             xhr.open("GET", "data.txt", false); // 'false' makes it synchronous

             xhr.send(); // Sends the request (execution is blocked until response)

             if (xhr.status === 200) {
                 document.getElementById("output").textContent = xhr.responseText;
             } else {
                 document.getElementById("output").textContent = "Error fetching data.";
             }
         }
     </script>

 </body>
 </html>
```

How It Works?

1. **User clicks the "Fetch Data" button**

2. **AJAX (XMLHttpRequest) sends a synchronous request to fetch data.txt**
3. **Execution stops until the response is received**
4. **When data is received, it updates the webpage**

Key Difference Between Synchronous & Asynchronous AJAX

Feature	Synchronous (Blocking)	Asynchronous (Non-Blocking)
Execution	Blocks execution until response is received	Continues execution without waiting
User Experience	Slower, freezes UI until response arrives	Faster, allows smooth user interactions
`open()` Method	`xhr.open("GET", "url", false);`	`xhr.open("GET", "url", true);`

Q.6.) Write an AJAX application to perform simple arithmetic operation

Ans :- **AJAX Application for Simple Arithmetic Operations**

This **AJAX-based application** allows users to **perform arithmetic operations (Addition, Subtraction, Multiplication, and Division)** without reloading the page. It sends data to a server-side script (calculate.php) and retrieves the result asynchronously.

Step 1: Create an HTML File (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX Arithmetic Operations</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
        input, select, button {
            font-size: 16px;
            padding: 10px;
            margin: 5px;
        }
        #result {
            font-size: 18px;
            color: green;
            margin-top: 20px;
        }
    </style>
</head>
<body>
```

```
<h2>AJAX Arithmetic Calculator</h2>

<input type="number" id="num1" placeholder="Enter first number">
<select id="operator">
    <option value="add">+</option>
    <option value="subtract">-</option>
    <option value="multiply">×</option>
    <option value="divide">÷</option>
</select>
<input type="number" id="num2" placeholder="Enter second number">
<button onclick="calculate()">Calculate</button>

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

```
<script>
    function calculate() {
        let num1 = document.getElementById("num1").value;
        let num2 = document.getElementById("num2").value;
        let operator = document.getElementById("operator").value;

        if (num1 === "" || num2 === "") {
            document.getElementById("result").textContent = "Please enter both numbers!";
            return;
        }
```

```
<script>
    function calculate() {
        let num1 = document.getElementById("num1").value;
        let num2 = document.getElementById("num2").value;
        let operator = document.getElementById("operator").value;

        if (num1 === "" || num2 === "") {
            document.getElementById("result").textContent = "Please enter both numbers!";
            return;
        }
```

```
          }
      </script>

   </body>
   </html>
```

Step 2: Create a PHP File (calculate.php)

This script processes the AJAX request and returns the calculated result.

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $num1 = floatval($_POST["num1"]);
    $num2 = floatval($_POST["num2"]);
    $operator = $_POST["operator"];
    $result = "";
switch ($operator) {
    case "add":
        $result = $num1 + $num2;
        break;
    case "subtract":
        $result = $num1 - $num2;
        break;
    case "multiply":
        $result = $num1 * $num2;
        break;
    case "divide":
        if ($num2 != 0) {
            $result = $num1 / $num2;
        } else {
            $result = "Error! Division by zero.";

            }
            break;
        default:
            $result = "Invalid operation";
      }

      echo $result;
  }
  ?>
```

How It Works?

1. The user enters two numbers and selects an arithmetic operation.
2. Clicking the **"Calculate"** button triggers the **AJAX request**.
3. The data (numbers and operator) is sent to calculate.php via **POST**.
4. The PHP script processes the request and returns the **result**.
5. The **result is displayed dynamically** without reloading the page.