Unit – II Javascript

Introduction to Scripting - Core features

Data types and Variables -Operators

Expressions and Statements

Functions - Arrays - Objects

Document Object Model and Event Handling

JSON – Introduction to AJAX.

1. Introduction to Scripting in JavaScript

JavaScript is a lightweight, interpreted, or just-in-time compiled programming language primarily used for web development. It allows developers to create interactive and dynamic web pages.

Why JavaScript?

- Enables client-side scripting, reducing server load.
- Provides interactive web pages (animations, form validation, etc.).
- Can be used for both frontend and backend (Node.js).
- Supported by all modern web browsers.
- We can validate the HTML page and pass Alerts to the users before passing the page to web servers.

2. Core Features of JavaScript

- Lightweight and interpreted
- Event-driven and asynchronous capabilities
- Dynamic typing (variables don't require explicit type declaration)
- Prototype-based Object-Oriented Programming (OOP)
- Cross-platform compatibility

3.Type of logging in JavaScript

- console.log → press Ctrl + Shift + J (Windows) to see the console output

- Alert pop up → alert(“Good Morning”)
- confirm("Are you sure you want to submit?")
- document.write → writing to a HTML page

4. Variable Declaration

JavaScript has three ways to declare variables:

1. **var** – can be re-declared and updated.
2. **let** – cannot be re-declared but can be updated.
3. **const** – **immutable** (cannot be reassigned, but objects/arrays can be mutated).

let name = "John";

const PI = 3.14;

var age = 25;

5.<u>printing & concatenation</u>

+ Operator → To joing a string and variable

Alert(“Hello”+name);

$ Operatot → to print the variable name

let user = { name: "Bob", age: 30 };

console.log(`User ${user.name} is ${user.age} years old.`);

<u>4. Data Types</u>

JavaScript provides primitive and reference data types.

Primitive Data Types (Immutable)

1. Number: let num = 10;
2. String: let str = "Hello";
3. Boolean: let isTrue = true;
4. Undefined: let x;
5. Null: let y = null;
6. BigInt: let bigNumber = 12345678901234567890n;
7. Symbol: let sym = Symbol('id');

Reference Data Types (Mutable)

- Objects, Arrays, Functions

In JavaScript, an **object** is declared using curly braces {} and consists of **key-value pairs**.

```
const person = { name: "John", age: 30, isStudent: false };

alert(“Hello”+person.name);
```

<u>Arrays</u>

JavaScript arrays store multiple values.

Array Declaration

```
let fruits = ["Apple", "Banana", "Cherry"];
```

Array Methods

- push(), pop(), shift(), unshift(), splice(), slice()
- map(), filter(), reduce(), forEach()

Example:

```
let numbers = [1, 2, 3, 4];

let doubled = numbers.map(num => num * 2);

console.log(doubled); // [2, 4, 6, 8]
```

<u>Functions</u>

A **function** in JavaScript is a reusable block of code designed to perform a specific task. Functions help **organize**, **reuse**, and **simplify** code.

```
function greetUser(name) {

  console.log("Hello, " + name + "!");

}


greetUser("Alice"); // Output: Hello, Alice!

greetUser("Bob");   // Output: Hello, Bob!


function add(a, b) {

  return a + b;

}
```

```
let sum = add(5, 10);

console.log(sum); // Output: 15
```

5.Operators, Expressions, and Statements

Operators

Arithmetic: +, -, *, /, %, ++, --

Comparison: ==, ===, !=, !==, >, <, >=, <=

Logical: &&, ||, !

Assignment: =, +=, -=, *=, /=

Bitwise: &, |, ^, ~, <<, >>

Ternary: condition ? expr1 : expr2

Expressions

An expression is a piece of code that produces a value.

Example:

```
let x = (10 + 5) * 2;
```

Statements

JavaScript statements execute a task.

Example:

```
let a = 5;  // Declaration statement

if (a > 3) { console.log("A is greater"); } // Conditional statement
```

6. Event Handling

Event handling in JavaScript allows us to detect user interactions such as clicks, key presses, mouse movements, and form submissions. JavaScript provides various methods to handle these events and execute specific actions.

addEventListener() is the best method for handling events.

Types of Events

Here are some common JavaScript events:

Event	**Description**

click	Triggered when an element is clicked
mouseover	Triggered when the mouse hovers over an element
mouseout	Triggered when the mouse leaves an element
keydown	Triggered when a key is pressed
keyup	Triggered when a key is released
change	Triggered when an input value changes
submit	Triggered when a form is submitted
load	Triggered when a page finishes loading

```
<button id="myButton">Click Me</button>
  <script>
    document.getElementById("myButton").addEventListener("click", function() {
      alert("Button Clicked!");
    });
  </script>
```

6. Document Object Model (DOM)

The Document Object Model (DOM) is a programming interface that represents an HTML as a tree structure where each node is an object. It allows JavaScript to interact, modify, and update web pages dynamically.

Selecting Elements

```
document.getElementById("id");
document.getElementsByClassName("class");
document.querySelector("selector");
document.querySelectorAll("selector");
```

Modifying Content using DOM

```
<button id="btn">Click Me</button>
<p id="message"></p>
<script>
```

```
   document.getElementById("message").innerText = "Hello! This is internal JavaScript.";
   document.getElementById("message").style.color = "blue";
   });
</script>
document.getElementById("myForm").addEventListener("submit", function(event)   {   }
<h1 id="title">Hello, JavaScript!</h1>
 <button id="changeText">Click Me</button>
 <script>document.getElementById("changeText").addEventListener("click", function() {
document.getElementById("title").textContent = "Text Changed!";
</script>
```

What is XML?

XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. Unlike HTML, which has predefined tags, XML allows users to define their own tags based on the structure of the data they want to represent. Used to pass data in we Applications.

Why Do We Use XML?

XML is widely used for several reasons:

1. **Data Storage and Transport** – XML is used to store and exchange data between different systems, making it a popular choice for web services and APIs.
2. **Platform Independence** – Since XML is plain text, it can be used across different programming languages and platforms without compatibility issues.
3. **Self-Descriptive** – XML documents contain metadata within tags, making them easy
4. **Web and Business Applications** – XML is widely used in web development (e.g., XHTML), business data interchange (e.g., SOAP, RSS), and database management.

books.xml (We can use PHP or other programming languages like Python, Node.js, or Java to capture data from a web page and create a new XML file dynamically**)**

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
   <book id="1">
```

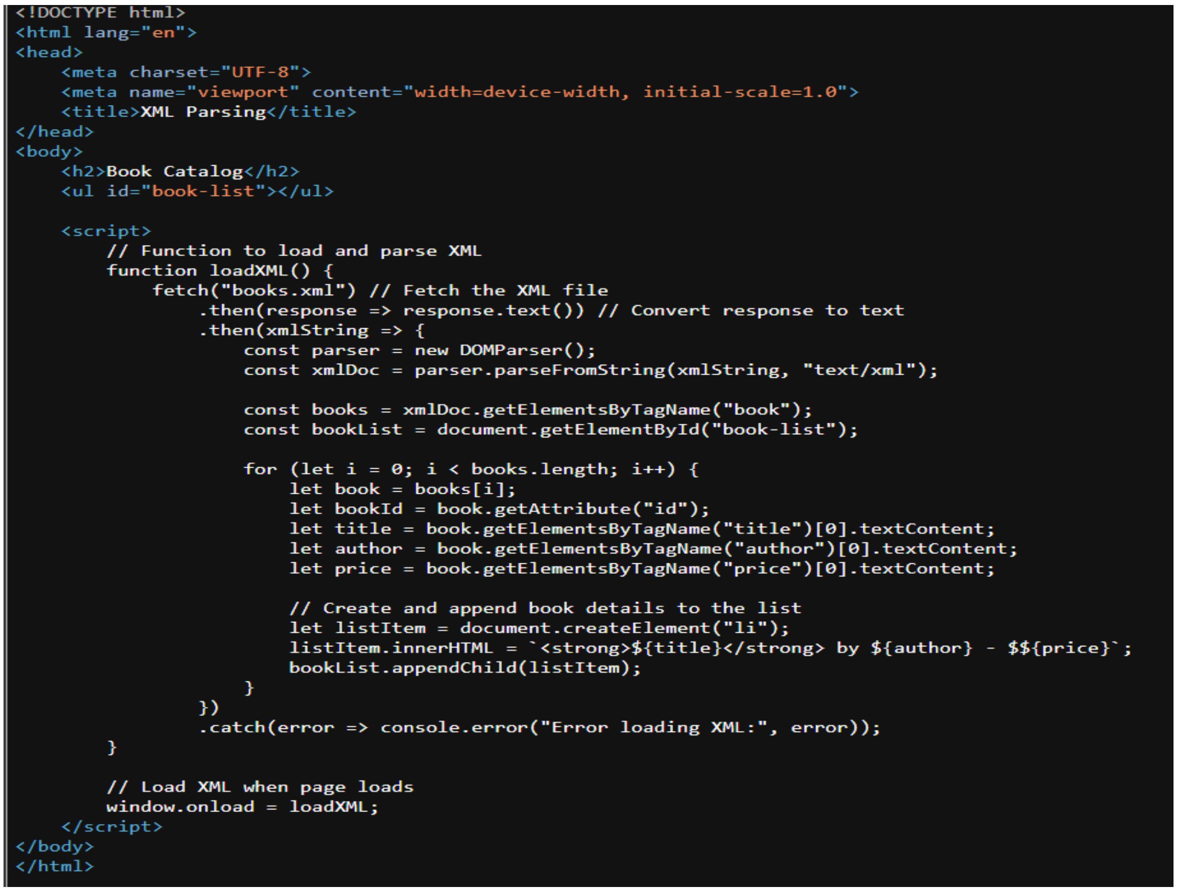
```xml
        <title>Introduction to XML</title>
        <author>John Doe</author>
        <publisher>Tech Books</publisher>
        <price>29.99</price>
        <publishedDate>2023-05-10</publishedDate>
    </book>
    <book id="2">
        <title>Mastering XML</title>
        <author>Jane Smith</author>
        <publisher>Web Press</publisher>
        <price>39.99</price>
        <publishedDate>2022-11-15</publishedDate>
    </book>
</catalog>
```

Technologies used to Fetch and Parse XML

1. **JavaScript/XML Parsing Technologies**
2. **PHP/XML Parsing Technologies**
3. **Python/XML Parsing Technologies**
4. **Java/XML Parsing Technologies**
5. **Node.js/XML Parsing Technologies**

Step 2: Fetch and Parse XML Using JavaScript

Use the fetch API to load the books.xml file and parse its data:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>XML Parsing</title>
</head>
<body>
    <h2>Book Catalog</h2>
    <ul id="book-list"></ul>

    <script>
        // Function to load and parse XML
        function loadXML() {
            fetch("books.xml") // Fetch the XML file
                .then(response => response.text()) // Convert response to text
                .then(xmlString => {
                    const parser = new DOMParser();
                    const xmlDoc = parser.parseFromString(xmlString, "text/xml");

                    const books = xmlDoc.getElementsByTagName("book");
                    const bookList = document.getElementById("book-list");

                    for (let i = 0; i < books.length; i++) {
                        let book = books[i];
                        let bookId = book.getAttribute("id");
                        let title = book.getElementsByTagName("title")[0].textContent;
                        let author = book.getElementsByTagName("author")[0].textContent;
                        let price = book.getElementsByTagName("price")[0].textContent;

                        // Create and append book details to the list
                        let listItem = document.createElement("li");
                        listItem.innerHTML = `<strong>${title}</strong> by ${author} - $${price}`;
                        bookList.appendChild(listItem);
                    }
                })
                .catch(error => console.error("Error loading XML:", error));
        }

        // Load XML when page loads
        window.onload = loadXML;
    </script>
</body>
</html>
```

How It Works:

1. **Fetch XML File**: fetch("books.xml") loads the file asynchronously.
2. **Convert to String**: .then(response => response.text()) ensures we get XML as a string.
3. **Parse XML**: DOMParser().parseFromString(xmlString, "text/xml") converts it to a DOM structure.
4. **Extract Data**: getElementsByTagName("book") retrieves all <book> elements.
5. **Display Data**: Books are added as list items (<li>) inside <ul id="book-list">.

Expected Output in Browser:

```
- Introduction to XML by John Doe - $29.99
- Mastering XML by Jane Smith - $39.99
```

Disadvantages of XML

1. Increased File Size and Overhead

XML documents contain a lot of markup (tags), which makes them larger than other data formats like JSON or binary formats.

2. Slower Processing Speed

XML requires parsing (reading and interpreting tags), which is computationally expensive.

Compared to JSON or binary formats, XML processing is slower.

3. Complex Syntax and Readability Issues

XML syntax is verbose and difficult to read when handling large datasets.

4. High Memory and Bandwidth Consumption

Because of extra tags, XML consumes more memory and requires higher bandwidth for data transmission.

Large XML files can cause performance issues in web applications.

5. No Native Data Types

XML stores everything as text, meaning developers must manually convert numbers, dates, or boolean values.

Example:

```
<price>100</price>  <!-- Stored as text, needs conversion to integer -->
```

JSON, on the other hand, natively supports data types like numbers, booleans, and arrays.

6. Poor Support for Data Manipulation

XML lacks built-in data querying capabilities like SQL for relational databases.

Unlike JSON, which integrates well with JavaScript, XML requires XPath or XQuery for data retrieval.

7. Limited Browser Compatibility

Modern browsers and web applications prefer JSON over XML.

AJAX calls and API responses are easier to handle using JSON because JavaScript supports JSON natively.

8. Security Issues

XML is vulnerable to attacks like:

XML External Entity (XXE) Attacks – attackers manipulate XML input to access sensitive files.

XML Bombs – attackers use recursive entity expansion to crash systems.

JSON

Introduction to JSON:

- JSON stands for JavaScript Object Notation.
- It is a lightweight data-interchange format that is easy to read and write for humans and machines.
- JSON is primarily used to transmit data between a server and a web application.
- It is text-based and language-independent, making it widely adopted across different programming languages.

JSON Syntax:

- JSON consists of key-value pairs.
- Data is enclosed in curly braces {} for objects and in square brackets [] for arrays.
- Key names must be in double quotes.
- Values can be strings, numbers, boolean (true/false), null, objects, or arrays.

Example of JSON Object:

```
{
  "name": "John Doe",
  "age": 30,
  "email": "john.doe@example.com",
  "isStudent": false,
  "address": {
    "street": "123 Main St",
    "city": "New York",
    "zip": "10001"
  },
  "hobbies": ["reading", "traveling", "gaming"]
}
```

JSON Data Types:

- **String**: Enclosed in double quotes (e.g., "hello").
- **Number**: Integer or floating-point (e.g., 10, 10.5).
- **Boolean**: true or false.
- **Null**: Represents an empty value (null).
- **Object**: Collection of key-value pairs enclosed in {}.
- **Array**: Ordered list of values enclosed in [].

Parsing JSON:

- In JavaScript, JSON.parse() is used to convert a JSON string into a JavaScript object.
- Example:

```
let jsonString = '{"name":"John", "age":30}';
let obj = JSON.parse(jsonString);
console.log(obj.name); // Output: John
```

Stringifying JSON:

- JSON.stringify() converts a JavaScript object into a JSON string.
- Example:

```
let obj = { name: "John", age: 30 };
let jsonString = JSON.stringify(obj);
console.log(jsonString);
```

Applications of JSON:

- Data exchange in web applications (e.g., REST APIs, AJAX requests).
- Configuration files (e.g., package.json in Node.js).
- Storing data in NoSQL databases like MongoDB.

- Serializing and deserializing data structures in different programming languages.

Comparison Between JSON and XML:

Feature	**JSON**	**XML**
Format	Key-value pairs	Tags and attributes
Readability	More readable	Verbose
Data Types	Supports various types	Everything is a string
Parsing	Faster and native in JavaScript	Slower and requires parsing libraries

INTRODUCTION TO AJAX

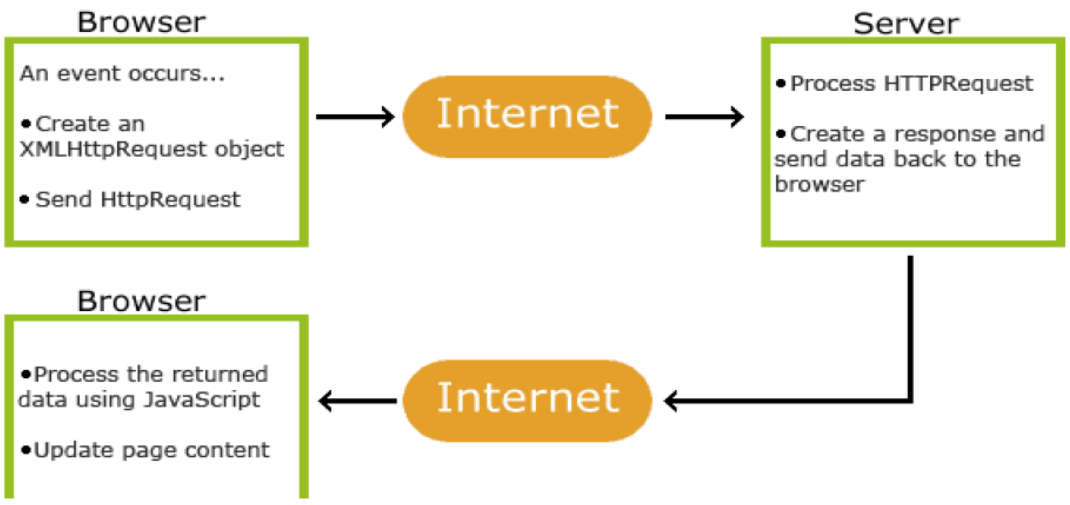
What is AJAX?

AJAX = Asynchronous JavaScript And XML. It is not a programming language.

AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text. AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

How AJAX Works

1. An event occurs in a web page (the page is loaded, a button is clicked)

2. An XMLHttpRequest object is created by JavaScript

3. The XMLHttpRequest object sends a request to a web server

4. The server processes the request

5. The server sends a response back to the web page

6. The response is read by JavaScript

7. Proper action (like page update) is performed by JavaScript