UNIT-2	JAVASCRIPT
Introduction to Scripting - Core features - Data types and Variables -Operators, Expressions and Statements - Functions - Arrays - Objects - Document Object Model - Event Handling- JSON – Introduction to AJAX.	

INTRODUCTION TO SCRIPTING

The scripting language is basically a language where instructions are written for a run time environment. They do not require the compilation step and are rather interpreted. It brings new functions to applications and glue complex system together. A scripting language is a programming language designed for integrating and communicating with other programming languages.

Scripting languages are a specific kind of computer languages that you can use to give instructions to other software, such as a web browser, server, or standalone application. Many of today's most popular coding languages are scripting languages, such as JavaScript, PHP, Ruby, and Python.

i. Advantages of scripting languages:

- **Easy learning:** The user can learn to code in scripting languages quickly, not much knowledge of web technology is required.
- **Fast editing:** It is highly efficient with the limited number of data structures and variables to use.
- **Interactivity:** It helps in adding visualization interfaces and combinations in web pages. Modern web pages demand the use of scripting languages. To create enhanced web pages, fascinated visual description which includes background and foreground colors and so on.
- **Functionality:** There are different libraries which are part of different scripting languages. They help in creating new applications in web browsers and are different from normal programming languages.

ii. Application of Scripting Languages:

- Scripting languages are used in web applications. It is used in server side as well as client side. Server side scripting languages are: JavaScript, PHP, Perl etc. and client side scripting languages are: JavaScript, AJAX, jQuery etc.
- Scripting languages are used in system administration. For example: Shell, Perl, Python scripts etc.

- It is used in Games application and Multimedia.
- It is used to create plugins and extensions for existing applications.

iii. Why is JavaScript called a Scripting Language?

JavaScript is a scripting language because it's a lightweight programming language that's interpreted by a browser, rather than compiled. Scripting languages are designed for quick development and execution of code.

iv. What is JavaScript used for?

- **Web development:** JavaScript is used to create dynamic content on web pages, such as interactive maps, animated graphics, and dropdown menus.
- **Game development:** JavaScript can be used to create games.
- **Other software systems:** JavaScript engines are embedded in other software systems, such as Node.js, Apache Couch DB, and Adobe Acrobat.

CORE FEATURES

JavaScript is a highly popular and widely-used programming language for web development. It has a variety of features that make it powerful and flexible. Many open-source libraries are available. GitHub contains a large volume of Javascript code by developers across the world. Javascript works well in the front end and back end.

Javascript has a simple syntax. Without any settings, anyone can execute Javascript programs and make them user-friendly. One individual having basic knowledge of HTML, CSS and coding can work with Javascript.

Core Feature of JavaScript

1. **Scripting -** Javascript executes the client-side script in the browser.
2. **Interpreter -** The browser interprets JavaScript code.
3. **Event Handling -** Events are actions. Javascript provides event-handling options.
4. **Light Weight -** As Javascript is not a compiled language, source code never changes to byte code before running time. Low-end devices can also run Javascript because of its lightweight feature.
5. **Case Sensitive -** In Javascript, names, variables, keywords, and functions are case-sensitive.

6. **Control Statements -** Javascript has control statements like if-else-if, switch case, and loop. Users can write complex code using these control statements.
7. **Supports Functional Programming -** Javascript functions can be an argument to another function, can call by reference, and can assign to a variable.
8. **Dynamic Typing -** Javascript variables can have any value type. The same variable can have a string value, an integer value, or any other.
9. **Client-side Validations -** Javascript client-side validations allow users to submit valid data to the server during a form submission.
10. **Platform Independent -** Javascript will run in the same way in all systems with any operating system.
11. **Dom Manipulation -** JavaScript allows developers to manipulate the webpage elements. Also, you can control the browser.
12. **Event Handling -** JavaScript allows you to handle the events used to interact with the web page.
13. **Cross-platform Support -** Each operating system and browser support JavaScript. So, it is widely used for developing websites, mobile applications, games, desktop applications, etc.
14. **Object-oriented Programming -** JavaScript contains the classes, and we can implement all object-oriented programming concepts using its functionality.
15. **JSON -** JSON stands for JavaScript object notation. It is a widely used data format to exchange data between two networks. For example, server and client.
16. **Server-side Support -** The Node.js runtime environment of JavaScript is widely used to create the backend of the application, as javaScript can also be used to create servers. It allows you to create a scalable backend for the application.

DATA TYPES AND VARIABLES

Variables and data types are foundational concepts in programming, serving as the building blocks for storing and manipulating information within a program. In JavaScript, getting a good grasp of these concepts is important for writing code that works well and is easy to understand.

1. Variables

A variable is like a container that holds data that can be reused or updated later in the program. In JavaScript, variables are declared using the keywords var, let, or const.

i. var Keyword

The var keyword is used to declare a variable. It has a function-scoped or globally-scoped behaviour.

```
var n = 5;
console.log(n);
var n = 20; // reassigning is allowed
console.log(n);
```

Output:

5

20

ii. let Keyword

The let keyword is introduced in ES6, has block scope and cannot be re-declared in the same scope.

```
let  n= 10;
n = 20; // Value can be updated
// let n = 15; //can not redeclare
console.log(n)
```

Output:

20

iii. const Keyword

The const keyword declares variables that cannot be reassigned. It's block-scoped as well.

const n = 100;

// n = 200; This will throw an error

console.log(n)

Output:

100

2. Data Types

JavaScript supports various datatypes, which can be broadly categorized into primitive and non-primitive types.

I. Primitive Datatypes

Primitive datatypes represent single values and are immutable.

1. **Number:** Represents numeric values (integers and decimals).

```
let n = 42;
let pi = 3.14;
```

2. **String**: Represents text enclosed in single or double quotes.

```
let s = "Hello, World!";
```

3. **Boolean**: Represents a logical value (true or false).

```
let bool= true;
```

4. **Undefined:** A variable that has been declared but not assigned a value.

```
let notAssigned;
console.log(notAssigned);
```

Output:

undefined

5. **Null:** Represents an intentional absence of any value.

```
let empty = null;
```

6. **Symbol:** Represents unique and immutable values, often used as object keys.

```
let sym = Symbol('unique');
```

7. **BigInt**: Represents integers larger than number.MAX_SAFE_INTEGER.

```
let bigNumber = 1234567890123456789012345678901234567890n;
```

II. Non-Primitive Datatypes

Non-primitive types are objects and can store collections of data or more complex entities.

1. **Object:** Represents key-value pairs.

```
let obj = {
    name: "Amit",
    age: 25
};
```

2. **Array:** Represents an ordered list of values.

```
let a = ["red", "green", "blue"];
```

3. **Function:** Represents reusable blocks of code.

```
function fun() {
    console.log("GeeksforGeeks");
}
```

Global and Local Variable in Javascript

Global Variable

Global variables in JavaScript are those declared outside of any function or block scope. They are accessible from anywhere within the script, including inside functions and blocks.

Variables declared without the var, let, or const keywords (prior to ES6) inside a function automatically become global variables.

Key Characteristics of Global Variables:

- **Scope:** Accessible throughout the entire script, including inside functions and blocks.
- **Automatic Global Variables:** If a variable is declared inside a function without var, let, or const, it automatically becomes a global variable (a common source of bugs).

Example of Global Variable:

```
<!DOCTYPE html>
<html>
<body>
```

```
<h2>JavaScript Scope</h2>
<p>A GLOBAL variable can be accessed from any script or function.</p>
<p id="demo"></p>
<script>
    let carName = "Volvo”; // Global Variable
    myFunction();

    function myFunction() {
     document.getElementById("demo").innerHTML = "I can display " + carName;
    }
</script>
</body>
</html>
```

Local Variable

Local variables are defined within functions in JavaScript. They are confined to the scope of the function that defines them and cannot be accessed from outside.

Attempting to access local variables outside their defining function results in an error.

Key Characteristics of Local Variables:

- **Scope:** Limited to the function or block in which they are declared.
- **Function-Specific:** Each function can have its own local variables, even if they share the same name.

Example of Local Variable:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Scope</h2>
<p><b>carName</b> is undefined outside myFunction():</p>
```

```
<p id="demo1"></p>
<p id="demo2"></p>
<script>
        myFunction();


        function myFunction() {
          let carName = "Volvo";  // Local Variable
          document.getElementById("demo1").innerHTML = typeof carName + " " + carName;
        }
        document.getElementById("demo2").innerHTML = typeof carName;
</script>
</body>
</html>
```

OPERATORS, EXPRESSIONS AND STATEMENTS

Operators

Javascript operators are used to perform different types of mathematical and logical computations.

Types of JavaScript Operators

There are different types of JavaScript operators:

- **Arithmetic Operators**
- **Assignment Operators**
- **Comparison Operators**
- **String Operators**
- **Logical Operators**
- **Bitwise Operators**
- **Ternary Operators**
- **Type Operators**

JavaScript Arithmetic Operators

- **Arithmetic Operators** are used to perform arithmetic on numbers:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

- **Example of Arithmetic Operator:**

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arithmetic</h1>
<h2>Arithmetic Operations</h2>
<p>A typical arithmetic operation takes two numbers (or expressions) and produces a new number.</p>
<p id="demo"></p>
<script>
        let a = 3;
        let x = (100 + 50) * a;
        document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Output:

JavaScript Arithmetic

Arithmetic Operations

A typical arithmetic operation takes two numbers (or expressions) and produces a new number.

450

JavaScript Assignment Operators

- Assignment operators assign values to JavaScript variables.
- The **Addition Assignment Operator (+=)** adds a value to a variable.

Operator	Example	Explanation
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

- **Example of Assignment Operator:**

  <!DOCTYPE html>

  <html>

  <body>

  <h1>JavaScript Arithmetic Assignment Operator</h1>

  <h2>The += Operator</h2>

```
<p id="demo"></p>
<script>
    var x = 10;
    x += 5;
    document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Output:

JavaScript Arithmetic Assignment Operator

The += Operator

15

JavaScript Comparison Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

JavaScript String Comparison:

All the comparison operators above can also be used on strings:

Example:

```
<script>
    let text1 = "A";
    let text2 = "B";
    let result = text1 < text2;
    document.getElementById("demo").innerHTML = "Is A less than B? " + result;
</script>
```

JavaScript Logical Operators

Operator	Description
&&	logical and
||	logical or
!	logical not

JavaScript Type Operators

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

JavaScript Bitwise Operators

- Bit operators work on 32 bits numbers.
- Any numeric operand in the operation is converted into a 32-bit number. The result is converted back to a JavaScript number.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

FUNCTIONS

Functions in JavaScript are reusable blocks of code designed to perform specific tasks. They allow you to organize, reuse, and modularize code. It can take inputs, perform actions, and return outputs.

JavaScript Function Syntax

- A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: **(*parameter1*, *parameter2*, ...)**
- The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name (parameter1, parameter2, parameter3)
{
    // code to be executed
}
```

- Function parameters are listed inside the parentheses () in the function definition.

- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Different ways of writing functions in JavaScript

JavaScript provides different ways to define functions, each with its own syntax and use case. Below are the ways of writing functions in JavaScript:

1. **Function Declaration**
2. **Function Expression**
3. **Arrow Functions**
4. **Key Differences: Function Declaration vs. Function Expression**

1) Function Declaration

A Function Declaration is the traditional way to define a function. It starts with the function keyword, followed by the function name and any parameters the function needs.

```
// function declaration
function add (a, b)
{
        console.log(a + b);
}

// calling a function
add(2, 3);
```

Output: 5

2) Function Expression

Function Expression is another way to define a function. Here, the function is defined inside a variable, and the function's value (its returned value) is stored in that variable.

```
// function expression
const add = function (a, b)
{
        console.log(a + b);
}
```

```
// calling a function
add(2, 3);
```

Output: 5

3) Arrow Functions

Arrow Functions were introduced in ES6 (a newer version of JavaScript). They provide a shorter syntax for writing functions. Instead of using the function keyword, you use an arrow (=>).

```
// Multiple line of code
const great = (a, b) => {
   if (a > b)
      return "a is greater";
   else
      return "b is greater";
}
console.log(great(3, 5));
```

Output: b is greater

4) Key Differences: Function Declaration vs. Function Expression

Function Declaration: The function is defined and can be called anywhere in the code (even before it's defined, due to something called "hoisting").

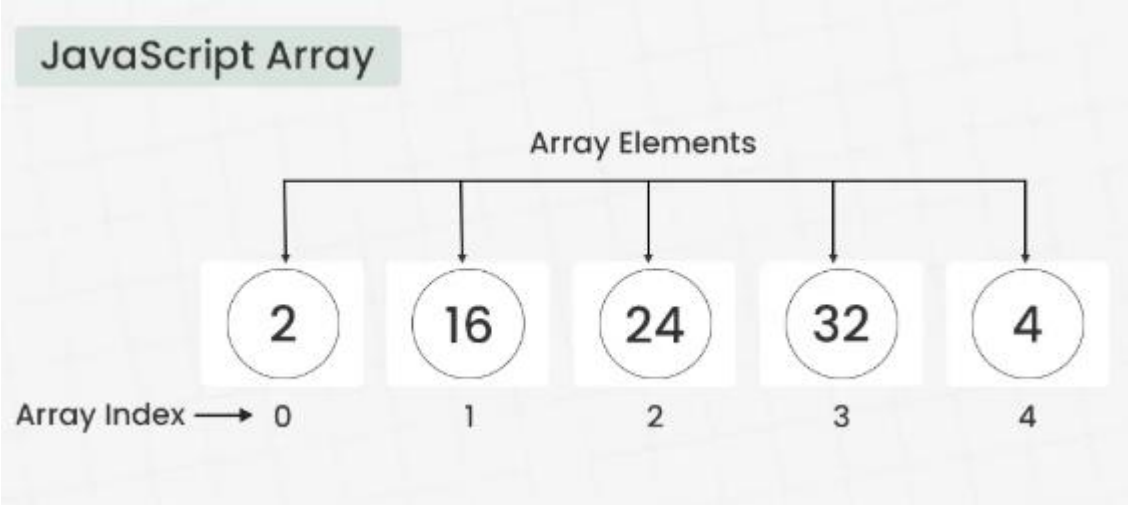
Function Expression: The function is defined as part of an expression (usually assigned to a variable), and it can only be called after the line where it's defined.

ARRAYS

An array in JS 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.

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
<p id="demo"></p>
<script>
    const cars = ["Saab", "Volvo", "BMW"];
    document.getElementById("demo").innerHTML = cars;
</script>
</body>
</html>
```



- **Create Array using Literal**

  Creating an array using array literal involves using square brackets [] to define and initialize the array.

```
// Creating an Empty Array
let a = [];

console.log(a);

// Creating an Array and Initializing with Values
let b = [10, 20, 30];

console.log(b);
```

Output

[]

[10, 20, 30]

- **Create using new Keyword (Constructor)**

The “Array Constructor” refers to a method of creating arrays by invoking the Array constructor function.

// Creating and Initializing an array with values

let a = new Array(10, 20, 30);

console.log(a);

Output

[10, 20, 30]

Basic Operations on JavaScript Arrays

1. **Accessing Elements of an Array**

Any element in the array can be accessed using the index number. The index in the arrays starts with 0.

// Creating an Array and Initializing with Values

let a = ["HTML", "CSS", "JS"];

// Accessing Array Elements

console.log(a[0]);

console.log(a[1]);

Output:

HTML

CSS

2. **Accessing the First Element of an Array**

The array indexing starts from 0, so we can access first element of array using the index number.

// Creating an Array and Initializing with Values

let a = ["HTML", "CSS", "JS"];

// Accessing First Array Elements

let fst = a[0];

console.log("First Item: ", fst);

Output

First Item: HTML

3. **Accessing the Last Element of an Array**

We can access the last array element using [array.length – 1] index number.

// Creating an Array and Initializing with Values

let a = ["HTML", "CSS", "JS"];

// Accessing Last Array Elements

let lst = a[a.length - 1];

console.log("First Item: ", lst);

Output

First Item: JS

4. **Modify the Array Elements**

Elements in an array can be modified by assigning a new value to their corresponding index.

// Creating an Array and Initializing with Values

let a = ["HTML", "CSS", "JS"];

console.log(a);

a[1]= "Bootstrap";

console.log(a);

Output

['HTML', 'CSS', 'JS']

['HTML', 'Bootstrap', 'JS']

5. **Adding Elements to the Array**

Elements can be added to the array using methods like push() and unshift().

- The **push()** method add the element to the end of the array.
- The **unshift()** method add the element to the starting of the array.

```
// Creating an Array and Initializing with Values
let a = ["HTML", "CSS", "JS"];
// Add Element to the end of Array
a.push("Node.js");
// Add Element to the beginning
a.unshift("Web Development");
console.log(a);
```

Output

['Web Development', 'HTML', 'CSS', 'JS', 'Node.js']

6. **Removing Elements from an Array**

To remove the elements from an array we have different methods **like pop(), shift(), or splice().**

- The pop() method removes an element from the last index of the array.
- The shift() method removes the element from the first index of the array.
- The splice() method removes or replaces the element from the array.

7. **Array Length**

We can get the length of the array using the array length property.

```
// Creating an Array and Initializing with Values
let a = ["HTML", "CSS", "JS"];
let len = a.length;
console.log("Array Length: " + len);
```

Output: Array Length: 3

8. Iterating Through Array Elements

We can iterate array and access array elements using for loop and forEach loop.

Example: It is an example of for loop.

```
// Creating an Array and Initializing with Values

let a = ["HTML", "CSS", "JS"];

// Iterating through for loop

for (let i = 0; i < a.length; i++) {

   console.log(a[i])

}
```

Output

HTML

CSS

JS

OBJECTS

An object in JavaScript is a data structure used to store related data collections. It stores data as key-value pairs, where each key is a unique identifier for the associated value. Objects are dynamic, which means the properties can be added, modified, or deleted at runtime.

There are two primary ways to create an object in JavaScript: Object Literal and Object Constructor.

1. Creation sing Object Literal

The object literal syntax allows you to define and initialize an object with curly braces {}, setting properties as key-value pairs.

```
let obj = {

        name: "Sourav",
```

```
    age: 23,
    job: "Developer"
};
console.log(obj);
```

Output

{name: 'Sourav', age: 23, job: 'Developer'}

2. Creation Using new Object() Constructor

```
let obj = new Object();
obj.name= "Sourav",
obj.age= 23,
obj.job= "Developer"
console.log(obj);
```

Output

{name: 'Sourav', age: 23, job: 'Developer'}

Basic Operations on JavaScript Objects

1. Accessing Object Properties

You can access an object's properties using either dot notation or bracket notation

```
let obj = { name: "Sourav", age: 23 };
// Using Dot Notation
console.log(obj.name);
// Using Bracket Notation
console.log(obj["age"]);
```

Output

Sourav

23

2. Modifying Object Properties

Properties in an object can be modified by reassigning their values.

```
let obj = { name: "Sourav", age: 22 };

console.log(obj);

obj.age = 23;

console.log(obj);
```

Output

{name: 'Sourav', age: 22}

{ ame: 'Sourav', age: 23}

3. Adding Properties to an Object

You can dynamically add new properties to an object using dot or bracket notation.

```
let obj = { model: "Tesla" };

obj.color = "Red";

console.log(obj);
```

Output

{ model: 'Tesla', color: 'Red' }

4. Removing Properties from an Object

The delete operator removes properties from an object.

```
let obj = { model: "Tesla", color: "Red" };

delete obj.color;
```

```
console.log(obj);
```

Output

```
{ model: 'Tesla' }
```

5. **Checking if a Property Exists**

You can check if an object has a property using the in operator or hasOwnProperty() method.

```
let obj = { model: "Tesla" };

console.log("color" in obj);

console.log(obj.hasOwnProperty("model"));
```

Output

```
false

true
```

DOCUMENT OBJECT MODEL

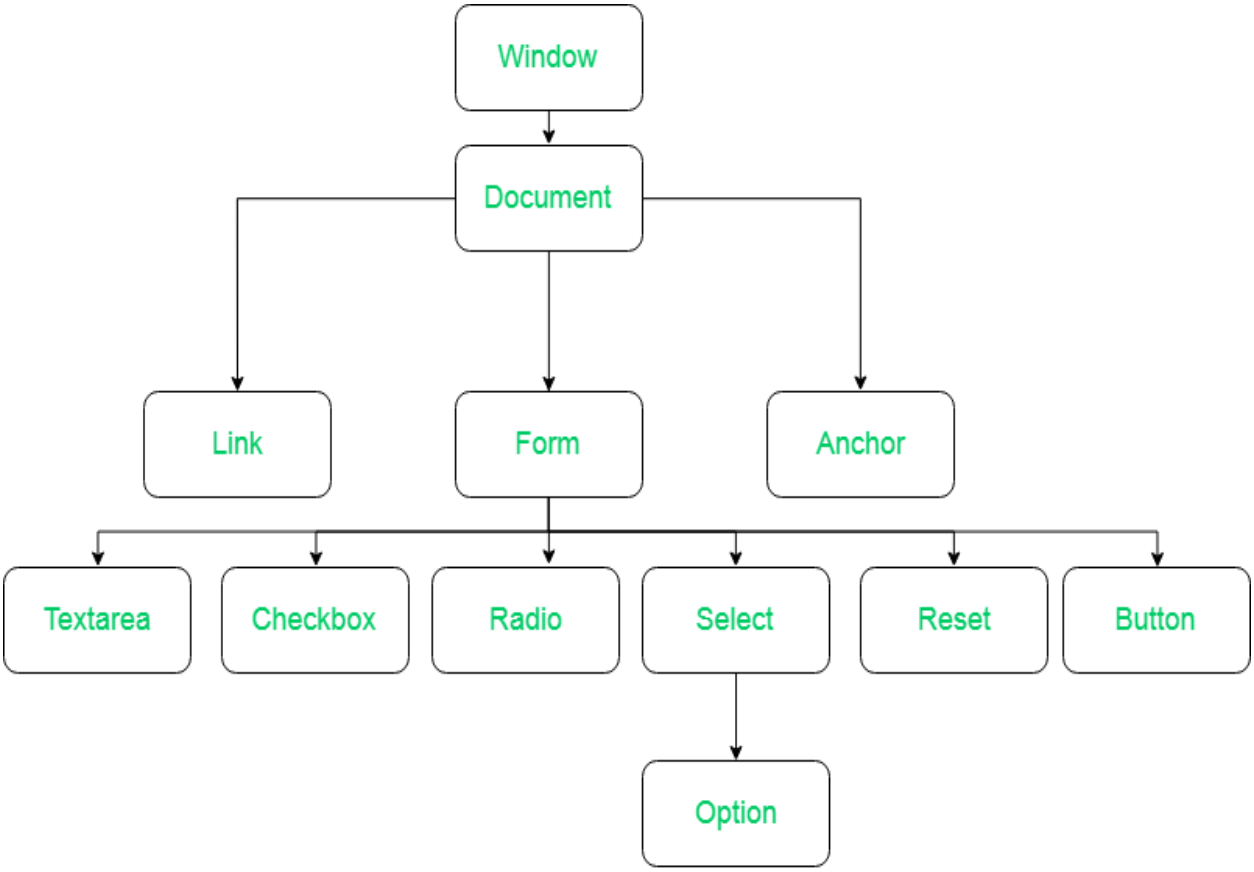
The HTML DOM (Document Object Model) is a programming interface that represents the structure of a web page in a way that programming languages like JavaScript can understand and manipulate.

Think of it as a tree of objects where each part of your HTML document (elements, attributes, text) is represented as a node, allowing you to dynamically change or interact with the content and structure of the page.

What is the DOM?

- The DOM is a W3C (World Wide Web Consortium) standard.
- The DOM defines a standard for accessing documents:
- "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
- The W3C DOM standard is separated into 3 different parts:
    1. Core DOM - standard model for all document types

2. XML DOM - standard model for XML documents
3. HTML DOM - standard model for HTML documents



Properties of the DOM

- **Node-Based:** Everything in the DOM is represented as a node (e.g., element nodes, text nodes, attribute nodes).
- **Hierarchical**: The DOM has a parent-child relationship, forming a tree structure.
- **Live:** Changes made to the DOM using JavaScript are immediately reflected on the web page.
- **Platform-Independent:** It works across different platforms, browsers, and programming languages.

EVENT HANDLING

JavaScript Events are actions or occurrences that happen in the browser. They can be triggered by various user interactions or by the browser itself.

```
<html>
<script>
        function myFun() {
           document.getElementById(
              "hai").innerHTML = "Hello World!";
        }
</script>
<body>
   <button onclick="myFun()">Click me</button>
   <p id="hai"></p>
</body>
</html>
```

- The onclick attribute in the <button> calls the myFun() function when clicked.
- The myFun() function updates the <p> element with id=”hai” by setting its innerHTML to “Hello World!”.
- Initially, the <p> is empty, and its content changes dynamically on button click.

Event Types

JavaScript supports a variety of event types. Common categories include:

- **Mouse Events:** click, dblclick, mousemove, mouseover, mouseout
- **Keyboard Events:** keydown, keypress, keyup
- **Form Events:** submit, change, focus, blur
- **Window Events:** load, resize, scroll

JSON

JSON (JavaScript Object Notation) is a lightweight text-based format for storing and exchanging data. It is easy to read, write, and widely used for communication between a server and a client.

Key points:

- JSON stores data in key-value pairs.
- It is language-independent but derived from JavaScript syntax.
- JSON data is written in a human-readable format.
- It supports objects { } and arrays [ ] for data representation.

JSON Example

```
{
   "name": "Shubham Verma",
   "age": 22,
   "city": "Haryana"
}
```

Parsing JSON data in JavaScript and accessing its values

```
const jsonStr = '{
                        "name": "Mohit",
                        "age": 30,
                        "city": "New Delhi"
                    }';
// Convert JSON string into JavaScript object
```

```
const obj = JSON.parse(jsonStr);

// Accessing JSON data

console.log(obj.name);

console.log(obj.age);

console.log(obj.city);
```

Output:

```
Mohit

30

New Delhi
```

Key Points About JSON

- **Data Storage in Key-Value Pairs:** JSON organizes data as key-value pairs.
- **Language-Independent:** Although derived from JavaScript, JSON is supported by most programming languages.
- **Human-Readable:** JSON data is written in a clean and visually understandable format.
- **Supports Objects and Arrays:** JSON can represent data using objects {} and arrays [].

INTRODUCTION TO AJAX

What is AJAX?

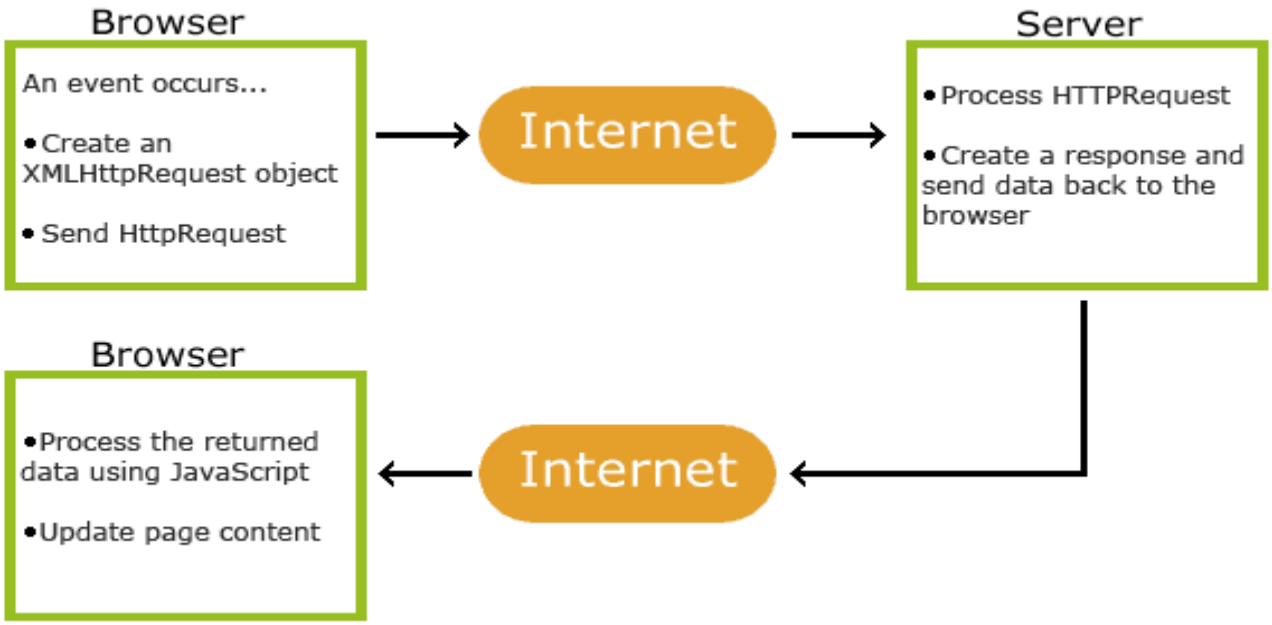
AJAX = **A**synchronous **J**avaScript **A**nd **X**ML. It is not a programming language.

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.

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