Unit III

Client Side Technologies

Client-side and server-side technologies are essential components of web development, each with distinct roles in creating and managing web applications. Here's a breakdown of their differences:

1. Client-Side Technologies:

Definition: These are technologies that run on the user's device (usually within a web browser). The client's browser processes the code and displays the content to the user.

Common Technologies:

HTML (HyperTextMarkup Language): Structures the content on the web page.

CSS (Cascading Style Sheets): Styles the content (e.g., colors, fonts, layouts).

JavaScript: Adds interactivity and dynamic content to web pages (e.g., form validations, animations).

Frameworks/Libraries: React, Angular, Vue.js are popular for building complex UIs.

Execution: Code is executed on the user's device (client).

Advantages:

Immediate interaction with the web page (e.g., form validation without page reload).

Reduces the load on the server by offloading processing to the client's device.

Disadvantages:

Dependent on the client's device and browser, which can vary in performance.

Security concerns, as the code is visible and modifiable by the user.

2. Server-Side Technologies:

Definition: These are technologies that run on a web server, processing requests, accessing databases, and generating responses to be sent to the client.

Common Technologies:

Programming Languages: PHP, Python (with Django or Flask), Ruby (with Ruby on Rails), Java (with Spring), Node.js, etc.

Databases: MySQL, PostgreSQL, MongoDB, etc.

Web Servers: Apache, Nginx, IIS.

Execution: Code is executed on the server before the content is sent to the client.

Advantages:

More secure since the code is not exposed to the client.

Handles complex operations like database queries, authentication, and file handling.

Centralized control over the application's logic and data.

Disadvantages:

Increased server load, as all processing happens on the server.

May result in slower user experience if the server or network is slow.

JavaScript:

Overview of JavaScript:

JavaScript is a versatile, high-level programming language that is widely used in web development. Initially created to add interactivity to web pages, it has grown into a powerful tool capable of handling a wide range of tasks, both on the client and server sides. Here's an overview:

1. History and Evolution:

Creation: JavaScript was created by Brendan Eich in 1995 while he was working at Netscape Communications. It was initially developed in just 10 days and was originally named Mocha, later renamed to LiveScript, and finally JavaScript.

Standardization: JavaScript was standardized by ECMA International, and the standardized version is called ECMAScript. The first edition, ECMAScript 1,

was released in 1997. Over the years, there have been several updates, with ECMAScript 6 (ES6) in 2015 being one of the most significant, introducing features like classes, modules, and arrow functions.

2. Key Features:

Interpreted Language: JavaScript is an interpreted language, meaning that it is executed line by line by the web browser or runtime environment without the need for prior compilation.

Dynamic Typing: Variables in JavaScript are dynamically typed, meaning you don't have to declare a specific data type for variables.

Prototypal Inheritance: JavaScript uses a prototype-based inheritance model, which is different from classical inheritance in languages like Java or C++.

Event-Driven: JavaScript is heavily event-driven, responding to user actions like clicks, form submissions, and other browser events.

First-Class Functions: Functions in JavaScript are first-class citizens, meaning they can be stored in variables, passed as arguments, and returned from other functions.

Asynchronous Programming: JavaScript supports asynchronous programming, primarily through callbacks, promises, and the async/await syntax introduced in ES6.

3. Core Concepts:

Variables and Data Types: JavaScript supports various data types, including strings, numbers, booleans, arrays, objects, and more. Variables can be declared using var, let, or const.

Functions: Functions are blocks of code designed to perform a particular task. They can be defined using the function keyword or as arrow functions (=>).

Objects and Arrays: Objects are collections of key-value pairs, while arrays are ordered lists of values. Both are fundamental to organizing and manipulating data in JavaScript.

DOM Manipulation: JavaScript can manipulate the Document Object Model (DOM), allowing developers to dynamically change the content, structure, and style of web pages.

Events: JavaScript handles user and browser events such as clicks, mouse movements, and key presses, allowing for interactive web applications.

Control Structures: JavaScript includes standard control structures like if, else, switch, for, while, and do-while loops.

4. Use Cases:

Client-Side Web Development: JavaScript is primarily used for client-side scripting to create dynamic, interactive web pages. It works alongside HTML and CSS to control the behaviour of web pages.

Server-Side Development: With the advent of Node.js, JavaScript can also be used on the server side, allowing for full-stack development using a single language.

Mobile App Development: Frameworks like React Native allow developers to build cross-platform mobile applications using JavaScript.

Game Development: JavaScript is used in developing browser-based games, often utilizing HTML5 and WebGL for graphics rendering.

Desktop Applications: Electron.js allows developers to build cross-platform desktop applications using JavaScript, HTML, and CSS.

5. Popular Libraries and Frameworks:

React.js: A JavaScript library for building user interfaces, particularly single-page applications.

Angular.js: A framework for building dynamic web applications, originally developed by Google.

Vue.js: A progressive framework for building user interfaces, known for its simplicity and flexibility.

Node.js: A runtime environment that allows JavaScript to be used for server-side scripting.

6. Advantages:

Wide Browser Support: JavaScript is supported by all modern web browsers, making it accessible to a vast audience.

Rich Ecosystem: A large community and ecosystem of libraries, frameworks, and tools support JavaScript development.

Interactivity: JavaScript is essential for creating rich, interactive web experiences.

Versatility: JavaScript can be used for both front-end and back-end development, as well as in mobile, desktop, and game development.

7. Challenges:

Security: Being a client-side language, JavaScript is susceptible to security issues like Cross-Site Scripting (XSS).

Browser Differences: Although JavaScript is standardized, subtle differences in how browsers implement JavaScript can cause inconsistencies.

Performance: For very complex or heavy computations, JavaScript may not perform as well as other languages, though modern engines like V8 have significantly improved this.

Using JS in an HTML (Embedded, External):

JavaScript can be included in an HTML document in two main ways: Embedded (within the HTML file itself) and External (in a separate file that the HTML file references). Here's how to use both methods:

1. Embedded JavaScript:

Embedded JavaScript is written directly within the HTML file, typically inside <script> tags. This method is useful for small scripts or when you want to keep everything in a single file.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Embedded JavaScript Example</title>

```
</head>
<body>
<h1>Hello, World!</h1>
<button onclick="showMessage()">Click Me!</button>
<!-- Embedded JavaScript -->
<script>
functionshowMessage() {
alert("Hello, this is an embedded JavaScript example!");
    }
</script>
</body>
</html>
```

Explanation:

The <script> tag is placed within the <body> or <head> section of the HTML document.

The JavaScript function showMessage() is defined within the <script> tag.

The onclick event handler on the button element triggers the showMessage() function when the button is clicked.

2. External JavaScript:

External JavaScript involves placing the JavaScript code in a separate .js file and linking it to the HTML document. This method is preferred for larger scripts, better organization, and reusability.

External JavaScript involves placing the JavaScript code in a separate .js file and linking it to the HTML document. This method is preferred for larger scripts, better organization, and reusability.

Example:

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>External JavaScript Example</title>
</head>
<body>
<h1>Hello, World!</h1>
<button onclick="showMessage()">Click Me!</button>
<!-- Link to the external JavaScript file -->
<script src="script.js"></script>
</body>
</html>
```

JavaScript File (script.js):

```
functionshowMessage() {
alert("Hello, this is an external JavaScript example!");
}
```

Explanation:

The JavaScript code is placed in an external file named script.js.

The HTML file includes the external JavaScript file using the <script> tag with the src attribute, which points to the location of the script.js file.

When the HTML document is loaded, the JavaScript file is automatically loaded as well, and the showMessage() function is available to be called by the button's onclick event.

Advantages of External JavaScript:

Separation of Concerns: Keeps the HTML structure separate from the behavior (JavaScript), making the code more organized and easier to maintain.

Reusability: The same JavaScript file can be linked to multiple HTML files.

Caching: Browsers can cache external JavaScript files, which can improve performance if the same script is used across multiple pages.

Data types:

JavaScript has a variety of data types that are categorized into two main groups: primitive types and reference types (also known as objects). Understanding these data types is crucial for working with JavaScript effectively.

1. Primitive Data Types

Primitive data types are the most basic types of data, and they are immutable, meaning their values cannot be changed. When you manipulate a primitive data type, you work directly with its value.

a. Number

Represents both integer and floating-point numbers.

JavaScript does not differentiate between integers and floats; all are of type Number.

Examples:

```
let age = 25;         // Integer
let temperature = 98.6; // Floating-point number
```

b. String

Represents a sequence of characters enclosed in single (' '), double (" "), or backticks (` ` for template literals).

Strings are used for text.

Examples:

```
let name = "Alice";       // Double quotes
```

let greeting = 'Hello'; // Single quotes

let message = `Hi, ${name}!`; // Template literal

c. Boolean

Represents a logical entity and can have only two values: true or false.

Examples:

let is Online = true;

let has Completed = false;

d. Undefined

A variable that has been declared but has not yet been assigned a value.

Example:

let x;

console.log(x); // Outputs: undefined

e. Null

Represents the intentional absence of any object value. It is often used to signify that a variable should have no value.

Example:

let y = null;

f. Symbol

Introduced in ECMAScript 6 (ES6), Symbol is a unique and immutable primitive value often used as the key of an object property to avoid naming conflicts.

Example:

letsym = Symbol("description");

g. BigInt

Introduced in ECMAScript 2020 (ES11), BigInt is used for representing integers that are too large to be represented by the Number type.

Example:

letbigNumber = 1234567890123456789012345678901234567890n;

2. Reference Data Types (Objects)

Reference types, or objects, are more complex data types. Unlike primitive data types, they are mutable and are stored by reference.

a. Objects

An Object is a collection of properties, where each property is a key-value pair. The keys are strings (or Symbols), and the values can be of any data type.

Example:

let person = {

firstName: "John",

lastName: "Doe",

age: 30,

};

b. Arrays

An Array is a special type of object that stores values in a numerically indexed list.

Example:

let fruits = ["apple", "banana", "cherry"];

c. Functions

In JavaScript, functions are objects. A function is a block of code designed to perform a particular task, and it can be assigned to a variable, passed as an argument, or returned from another function.

Example:

function greet(name) {

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

d. Date

The Date object is used to work with dates and times.

Example:

e. Regular Expressions (RegExp)

The RegExp object is used for matching text with a pattern.

Example:

```
let pattern = /hello/i;
```

f. Null (again)

While null is often considered a primitive type, it is technically an object in JavaScript.

Example:

```
let x = null;  // The type of x is object
```

3. Special Cases

NaN (Not a Number): A special numeric value that represents an undefined or un representable value, typically resulting from invalid operations like dividing zero by zero.

Infinity and -Infinity: Special numeric values representing positive and negative infinity, typically resulting from operations like dividing a number by zero.

Type Checking

To check the type of a variable, you can use the type of operator:

```
console.log(typeof 42);        // "number"

console.log(typeof "Hello");   // "string"

console.log(typeof true);      // "boolean"

console.log(typeof undefined); // "undefined"

console.log(typeof null);      // "object" (this is a historical bug in JavaScript)

console.log(typeof {});        // "object"

console.log(typeof []);        // "object" (arrays are objects)

console.log(typeof function(){}); // "function"
```

Control Structures:

JavaScript control statement is used to control the execution of a program based on a specific condition. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.

1. Conditional Statements

A.if statement: Executes a block of code if a specified condition is true

```
if (condition) {
   // code to be executed if condition is true
}
```

B.if...else statement: Executes one block of code if a condition is true, and another block if it's false.

```
if (condition) {
   // code to be executed if condition is true
} else {
   // code to be executed if condition is false
}
```

C.else if ladder: Checks multiple conditions in sequence

```
if (condition1) {
   // code to be executed if condition1 is true
} else if (condition2) {
   // code to be executed if condition2 is true
} else {
   // code to be executed if all conditions are false
}
```

D.switch statement: Selects one of many code blocks to be executed

```
switch(expression) {
case value1:
```

```
        // code block
break;
case value2:
        // code block
break;
default:
        // default code block
}
```

2. Looping Structures

A.for loop: Repeats a block of code a specified number of times

```
for (initialization; condition; increment) {
   // code to be executed
}
```

B.while loop: Repeats a block of code as long as a specified condition is true

```
while (condition) {
   // code to be executed
}
```

C.do...while loop: Executes a block of code once, and then repeats the loop as long as a specified condition is true

```
do {
   // code to be executed
} while (condition);
```

D.for...in loop: Iterates over the properties of an object

```
for (let key in object) {
   // code to be executed
}
```

E.for...of loop: Iterates over the values of an iterable object (like an array or string).

```
for (let value of iterable) {
   // code to be executed
}
```

3. Control Flow Interruptions

A.break: Exits a loop or switch statement prematurely

```
for (let i = 0; i < 10; i++) {
if (i === 5) {
break; // breaks out of the loop
   }
}
```

B.continue: Skips the rest of the current loop iteration and moves to the next iteration.

```
for (let i = 0; i < 10; i++) {
if (i === 5) {
continue; // skips the rest of the loop iteration when i is 5
```

```
    }
}
```
C.return: Exits a function and optionally returns a value
```
function sum(a, b) {
return a + b; // exits the function and returns the sum
}
```
4. Exception Handling

A.try...catch: Handles exceptions and errors in the code
```
try {
    // code that may throw an error
} catch (error) {
    // code to handle the error
} finally {
    // code that will run regardless of the try/catch result
}
```

Arrays:

Arrays in JavaScript are used to store multiple values in a single variable. They can hold a mix of data types, including numbers, strings, objects, and even other arrays.

Creating Arrays

You can create an array using either the array literal syntax or the Array constructor:

// Array literal syntax

```
let fruits = ['apple', 'banana', 'cherry'];
```

// Array constructor

```
let numbers = new Array(1, 2, 3);
```

Common Array Methods

.push(): Adds an element to the end of an array.

.pop(): Removes the last element from an array.

.shift(): Removes the first element from an array.

.unshift(): Adds an element to the beginning of an array.

.slice(): Returns a shallow copy of a portion of an array.

.map(): Creates a new array with the results of calling a provided function on every element.

Functions

Functions are blocks of code designed to perform a particular task. They can be defined using function declarations or function expressions.

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

Function Expression

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

Arrow Functions

A more concise syntax introduced in ES6:

```
const greet = (name) => `Hello, ${name}!`;
```

Parameters and Arguments

Functions can accept parameters and you can pass arguments when calling the function.

```
function add(a, b) {
return a + b;
}
console.log(add(5, 10)); // Output: 15
```

Scope:

Scope determines the accessibility of variables in your code. There are three main types of scope in JavaScript:

1.Global Scope: Variables declared outside any function or block are in the global scope and can be accessed from anywhere in the code.

```
letglobalVar = 'I am global';

functioncheckScope() {

console.log(globalVar); // Accessible

}

letglobalVar = 'I am global';


functioncheckScope() {

console.log(globalVar); // Accessible

}

functionmyFunction() {

letfunctionVar = 'I am local';

console.log(functionVar); // Accessible

}

console.log(functionVar); // Error: functionVar is not defined
```

2. Block Scope: Introduced with ES6, let and const create block-scoped variables. This means they are only accessible within the block (e.g., within {}) in which they are defined.

```
if (true) {

letblockVar = 'I am block-scoped';

console.log(blockVar); // Accessible

}

console.log(blockVar); // Error: blockVar is not defined.
```

Objects in JS:

In JavaScript, objects are collections of key-value pairs, where the keys (also known as properties) are strings or symbols, and the values can be any valid JavaScript data type, including other objects, functions, or primitive data types like numbers and strings.

Basic Syntax:

```
letobj = {
   key1: value1,
   key2: value2,
   // ...
};
```

Example:

```
let person = {
firstName: "John",
lastName: "Doe",
age: 30,
isEmployed: true,
greet: function() {
console.log("Hello, " + this.firstName);
   }
};
```

Accessing Object Properties:

Dot Notation:

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

Bracket Notation:

```
console.log(person['lastName']); // Output: Doe
```

Adding or Updating Properties:

person.age = 31; // Update

person.gender = "male"; // Add new property

Deleting Properties:

deleteperson.isEmployed;

Object Methods:

Objects can have functions as values, called methods. In the example above, greet is a method.

Looping through Object Properties:

Use for...in to loop over object properties.

for (let key in person) {

console.log(key + ": " + person[key]);

}

Other Important Features of Objects:

Objects are mutable: When you modify an object, the changes are reflected in all references to that object.

Nested objects: You can have objects within objects.

Example of a Nested Object:

let company = {

name: "Tech Corp",

employees: {

manager: "Alice",

developer: "Bob"

    }

};

console.log(company.employees.developer); // Output: Bob

Objects in JavaScript are fundamental for organizing and managing data in a structured way.

DOM: DOM levels:`

The DOM (Document Object Model) is a programming interface for web documents. It represents the page structure as a tree of objects that can be manipulated by scripting languages like JavaScript. Over time, the DOM specification has evolved through different levels or versions, which introduced various features and improvements.

DOM Levels Overview:

1. DOM Level 0 (Legacy DOM)

Not standardized, but refers to the early ways of accessing and manipulating HTML documents via JavaScript.

Supported basic operations like document.getElementById, document.forms, and document.images.

It is primarily limited to working with HTML forms, links, images, and document navigation.

2. DOM Level 1 (1998)

First W3C standard for the DOM.

Introduced the core DOM Core and DOM HTML modules.

The core module defined a standard way to navigate and manipulate document structures (HTML or XML), including elements, attributes, and text content.

Allowed dynamic access and update of the content, structure, and style of documents.

Key Features:

getElementById

getElementsByTagName

Attribute and element manipulation (setAttribute, getAttribute)

Tree navigation (e.g., parentNode, firstChild, lastChild)

3. DOM Level 2 (2000)

Extended the Level 1 specification and introduced additional features and modules.

It introduced events, CSS manipulation, and XML support.

Key Features:

DOM Events: A new way to register and handle events like click, mouseover, keydown, etc., using the addEventListener() method (vs the older onclick attribute).

DOM CSS: A module to access and manipulate CSS styles (getComputedStyle).

DOM Traversal and Range: Enhanced navigation with features like createRange() for working with parts of the document.

Namespace support: Specifically for XML documents that require namespace handling.

4. DOM Level 3 (2004)

Built on the previous versions and added more sophisticated features, especially for working with XML documents.

Key Features:

XPath Support: Enabled the use of XPath (a language to query XML documents) within the DOM.

Load and Save: Allowed loading and saving of XML documents.

Validation: Added support for checking whether a document adheres to its schema or DTD.

Text and Character Data Manipulation: Improvements for handling text nodes and character data within the document.

5. DOM Level 4 (Obsolete)

DOM Level 4 was initially planned to be an official update but was abandoned, and most of its features were integrated directly into the HTML5 specification.

Focused primarily on dealing with event handling, mutation observers, and cross-document messaging.

Modern DOM (Post-Level 3)

After DOM Level 3, the DOM API began to evolve more directly as part of HTML5 and other web standards. Today, modern DOM APIs are more modular and continuously updated, so there are no specific "levels" anymore. Instead, newer **features** are added incrementally and managed through various specifications like:

Selectors API: For example, documentquerySelector and document. QuerySelectorAll for more flexible and CSS-like element selection.

HTML5 DOM Extensions: Introduced new elements and APIs to handle form data (localStorage, sessionStorage), media, history, and much more.

Mutation Observers: Replaced the deprecated Mutation Events for observing changes to the DOM in a more efficient way.

Shadow DOM: Part of the Web Components specification, allowing for encapsulation of DOM elements and styles.

DOM Objects and their properties and methods:

The DOM (Document Object Model) objects represent the elements and structure of a web page. These objects allow developers to interact with and manipulate HTML or XML documents using properties (to get or set values) and methods (to perform actions).

Here are some of the most common DOM objects and their properties and methods:

1. document Object:

The document object represents the entire HTML or XML document and serves as the entry point to access other elements in the DOM.

Properties:

document.title: Gets or sets the title of the document.
document.body: Represents the <body> element.
document.URL: The full URL of the current page.

document.domain: The domain name of the server hosting the document.
document.cookie: Allows access to the document's cookies.

Methods:

getElementById(id): Returns an element by its id.
getElementsByClassName(class): Returns a collection of elements with the specified class name.
getElementsByTagName(tag): Returns a collection of elements with the specified tag name.
querySelector(cssSelector): Returns the first element matching a CSS selector.
querySelectorAll(cssSelector): Returns all elements matching a CSS selector.
createElement(tagName): Creates a new element of the given tag.
createTextNode(text): Creates a new text node.

2. Element Object

Represents any HTML or XML element. The Element object is one of the most common objects for accessing and manipulating individual elements.

Properties:

innerHTML: Gets or sets the HTML content inside an element.
outerHTML: Gets or sets the HTML content including the element itself.
textContent: Gets or sets the text content of the element (without HTML tags).
id: Gets or sets the id attribute of the element.
className: Gets or sets the class attribute.
style: Accesses the inline CSS styles of the element.
attributes: Returns a live collection of all attributes.

Methods:

getAttribute(attrName): Gets the value of a specified attribute.
setAttribute(attrName, value): Sets the value of an attribute.
removeAttribute(attrName): Removes an attribute.
appendChild(childNode): Adds a child node to the element.
removeChild(childNode): Removes a child node from the element.
replaceChild(newChild, oldChild): Replaces a child node with another node.
hasChildNodes(): Returns true if the element has any child nodes.
cloneNode(deep): Clones the element (optionally deep clones its child elements as well).

3. Event Object

The Event object represents any action or occurrence (such as clicks, key presses, etc.) that can be handled in the DOM.

Properties:

type: The type of event (e.g., "click", "keydown").
target: The element that triggered the event.
currentTarget: The element currently handling the event.
timeStamp: The time when the event was created.
bubbles: Whether the event bubbles up through the DOM.

Methods:
preventDefault(): Prevents the default action associated with the event (e.g., preventing form submission).
stopPropagation(): Stops the event from bubbling up to parent elements.

4. Window Object
The window object represents the browser window or frame that contains the document.

Properties:
window.innerWidth: The width of the window's content area.
window.innerHeight: The height of the window's content area.
window.location: Represents the current URL of the window, allowing you to access or manipulate parts of the URL.
window.localStorage: Provides access to a storage object for persisting key-value pairs in the browser (data persists across sessions).
window.sessionStorage: Similar to localStorage, but data persists only for the duration of the page session.

Methods:
alert(message): Displays an alert box with a message.
confirm(message): Displays a confirmation dialog with OK and Cancel.
prompt(message): Displays a prompt dialog asking for user input.
setTimeout(function, delay): Executes a function after a delay.
setInterval(function, delay): Repeatedly executes a function at specified intervals.
open(url, target): Opens a new browser window or tab with the specified URL.

5. Node Object
Represents a generic node in the DOM tree. Every element, attribute, and text is a type of Node.

Properties:
nodeType: Returns the type of the node (e.g., element, text, comment).
nodeName: The name of the node (e.g., DIV, TEXT).
parentNode: The parent node of the current node.
childNodes: A collection of child nodes (elements, text, comments).

Methods:
appendChild(node): Adds a child node.
removeChild(node): Removes a child node.
cloneNode(deep): Clones the node (optionally deep-cloning child nodes).

6. CSSStyleDeclaration Object
Represents the CSS style declarations of an element, allowing you to manipulate the element's inline styles.

Properties:

cssText: Represents the inline styles as a text string.
width, height, backgroundColor, etc.: Properties that represent individual CSS styles.

Methods:

getPropertyValue(propertyName): Gets the value of a specific CSS property.
setProperty(propertyName, value): Sets the value of a CSS property.
removeProperty(propertyName): Removes a specific CSS property.

7. Form and Input Objects

Used to represent forms and form elements like text inputs, checkboxes, and buttons.

Form Object Properties:

action: The URL to which the form will submit.
method: The HTTP method (GET, POST) used when submitting the form.
elements: A collection of all form controls.
Form Object Methods:
submit(): Submits the form.
reset(): Resets the form to its default values.
Input Object Properties:
value: The current value of the input field.
checked: Returns whether a checkbox or radio button is checked.

8. History Object

The history object allows you to interact with the browser's session history.

Methods:

history.back(): Moves the user to the previous page in the session history.
history.forward(): Moves the user to the next page in the session history.
history.go(n): Moves the user by n pages in the session history.
These DOM objects, properties, and methods form the foundation of JavaScript's interaction with HTML and XML documents, allowing developers to create dynamic and interactive web applications.

Manipulating DOM:

Manipulating the DOM (Document Object Model) refers to the process of using JavaScript (or similar technologies) to interact with and modify the structure, style, or content of a webpage. Here's a brief overview of key DOM manipulation techniques:

1. Selecting Elements

To modify any part of the DOM, you first need to select the element you want to manipulate.
getElementById: Selects an element by its ID.
var element = document.getElementById("myId");
getElementsByClassName: Selects elements by their class name (returns a collection).
var elements = document.getElementsByClassName("myClass");

getElementsByTagName: Selects elements by their tag name (returns a collection).
var elements = document.getElementsByTagName("p");
querySelector: Selects the first element that matches a CSS selector.
var element = document.querySelector(".myClass");
querySelectorAll: Selects all elements that match a CSS selector.
var elements = document.querySelectorAll(".myClass");

2. Modifying Content

You can change the content inside an HTML element.
innerHTML: Sets or gets the HTML inside an element.
element.innerHTML = " New content ";
textContent: Sets or gets the text inside an element.
element.textContent = "New text content";

3. Modifying Attributes

You can get, set, or remove attributes from an element.
setAttribute: Sets a specified attribute.
element.setAttribute("class", "newClass");
getAttribute: Retrieves the value of an attribute.
varattr = element.getAttribute("href");
removeAttribute: Removes a specified attribute.
element.removeAttribute("class");

4. Modifying Styles

You can change the CSS styles of an element dynamically.
style: Modifies the inline style of an element.
element.style.color = "red";
element.style.fontSize = "20px";

5. Creating and Appending Elements

You can create new elements and append them to the DOM.
createElement: Creates a new element.
varnewElement = document.createElement("div");
appendChild: Appends an element as a child to another element.
document.body.appendChild(newElement);
insertBefore: Inserts a new element before another one.
varreferenceElement = document.getElementById("myId");
document.body.insertBefore(newElement, referenceElement);

6. Event Handling

You can add or remove event listeners to handle user interactions.
addEventListener: Attaches an event handler.
element.addEventListener("click", function() {
alert("Element clicked");
});
removeEventListener: Removes an event handler.
element.removeEventListener("click", handleClickFunction);

Example: Changing the text of a paragraph on button click

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">This is a paragraph.</p>
<button id="myButton">Change Text</button>
<script>
var button = document.getElementById("myButton");
button.addEventListener("click", function() {
document.getElementById("demo").textContent = "Text has been changed!";
  });
</script>
</body>
</html>
```

JQuery:

Introduction to jQuery:

JQuery is a fast, small, and feature-rich JavaScript library. It simplifies HTML document traversal and manipulation, event handling, and animation, making it easier to develop interactive websites.

Key Features of jQuery

DOM Manipulation:jQuery provides an easy-to-use syntax for selecting elements and modifying their content and styles.

Event Handling: It simplifies handling events like clicks, mouse movements, and keyboard inputs with cross-browser compatibility.

AJAX Support:jQuery offers a simple way to make asynchronous HTTP requests, allowing for dynamic updates to web pages without a full reload.

Animation and Effects: Built-in methods make it easy to create animations and effects like fading, sliding, and showing/hiding elements.

Cross-Browser Compatibility:jQuery normalizes differences between browsers, allowing developers to write code that works consistently across all major browsers.

Plugins:jQuery's architecture allows developers to create plugins that extend its capabilities, leading to a rich ecosystem of reusable components.

Getting Started with jQuery

Include jQuery: You can include jQuery in your project either by downloading it or linking to a CDN. Here's how to link it via a CDN:

<scriptsrc="https://code.jquery.com/jquery-3.6.0.min.js"></script>

Basic Syntax:jQuery uses the $ symbol to access its functions. The basic syntax is:

$(selector).action();

For example, to hide a paragraph:

$("p").hide();

Document Ready: To ensure your code runs after the DOM is fully loaded, use:

```
$(document).ready(function() {
    // Your code here
});
```

Or the shorthand version:

```
$(function() {
    // Your code here
});
```

Example

Here's a simple example that changes the text of a button when clicked:

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

```
    .js"></script>
</head>
<body>
<button id="myButton">Click me!</button>
<script>
$(document).ready(function() {
   $("#myButton").click(function() {
      $(this).text("You clicked me!");
   });
});
</script>
</body>
</html>
```

Loading Jquery:

To use jQuery in a project, you first need to load the jQuery library. There are two primary ways to load jQuery: via a CDN (Content Delivery Network) or by downloading it locally.

1. Loading jQuery via CDN

This is the most common and convenient method. You simply link to a hosted version of jQuery using a <script> tag in your HTML file.

Example (Using jQuery from CDN):

Add this line in the <head> or just before the closing </body> tag of your HTML document:

```
<!-- Latest version of jQuery from Google CDN -->
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
```

Alternatively, you can use the official jQuery CDN:

<!-- Latest version of jQuery from jQuery CDN -->

<scriptsrc="https://code.jquery.com/jquery-3.6.0.min.js"></script>

2. Local jQuery (Downloading and Hosting Locally)

If you prefer to host the jQuery library on your own server, you can download it and include it in your project.

Steps:

1.DownloadjQuery from the official website:

https://jquery.com/download/

2.Save the file (e.g., jquery.min.js) into your project folder, typically inside a js directory.

Include the local file in your HTML:

<scriptsrc="js/jquery.min.js"></script>

3. Loading jQuery with Fallback

For better reliability, you can load jQuery via a CDN and provide a fallback to the local copy in case the CDN is unavailable.

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

<script>

if (typeofjQuery == 'undefined') {

document.write('<script src="js/jquery.min.js"><\/script>');

}

</script>

4. Verifying jQuery is Loaded

To verify that jQuery has been loaded correctly, you can add a simple test script to check if jQuery is available.

```
<script>
  $(document).ready(function() {
console.log("jQuery is successfully loaded!");
  });
</script>
```

5. Loading jQuery at the Bottom of the Page

For better performance, it's often recommended to load jQuery right before the closing </body> tag, as it ensures that the HTML is fully loaded before JavaScript starts running.

```
<!DOCTYPE html>
<html>
<head>
<title>My Page</title>
</head>
<body>
<!-- Your content here -->
<!-- Load jQuery at the end of the page -->
<scriptsrc="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
  $(document).ready(function() {
console.log("Page and jQuery are fully loaded");
  });
</script>
</body>
</html>
```

changing styles:

In jQuery, you can easily change the styles of HTML elements dynamically. There are multiple ways to modify CSS properties, including using the .css() method, adding/removing classes, or modifying inline styles directly.

1. Using .css() Method

The .css() method in jQuery allows you to get or set CSS properties for selected elements.

Syntax:

```
$(selector).css("property", "value");
```

Example (Changing the Color):

```
<p id="myText">This is a paragraph.</p>
<script>
  $(document).ready(function() {
    $("#myText").css("color", "blue"); // Changes the text color to blue
  });
</script>
```

Example (Setting Multiple Styles):

You can also change multiple CSS properties at once by passing an object to the .css() method.

```
$(selector).css({
  "property1": "value1",
  "property2": "value2"
});
<p id="myText">This is a paragraph.</p>

<script>
  $(document).ready(function() {
    $("#myText").css({
      "color": "red",
      "font-size": "20px",
      "background-color": "yellow"
    });
  });
</script>
```

2. Adding or Removing Classes

Instead of directly changing styles, it's often better to use classes for styling. jQuery provides methods like .addClass(), .removeClass(), and .toggleClass() to manipulate classes.

Adding a Class:

```
$(selector).addClass("className");
```

Example:

```
<p class="text">This is a paragraph.</p>
```

```
<style>
  .highlight {
color: green;
font-weight: bold;
  }
</style>

<script>
  $(document).ready(function() {
    $(".text").addClass("highlight"); // Adds the 'highlight' class, changing the
color and font weight
  });
</script>
```

Removing a Class:

```
$(selector).removeClass("className");
```

Example:

```
<p class="text highlight">This is a paragraph.</p>

<script>
  $(document).ready(function() {
    $(".text").removeClass("highlight"); // Removes the 'highlight' class,
reverting the style
  });
</script>
```

Toggling a Class:

The .toggleClass() method will add the class if it's not present and remove it if it is.

```
$(selector).toggleClass("className");
```

Example:

```
<p class="text">This is a paragraph.</p>

<style>
  .highlight {
color: green;
font-weight: bold;
  }
</style>

<script>
  $(document).ready(function() {
    $(".text").click(function() {
      $(this).toggleClass("highlight"); // Toggles the 'highlight' class on click
```

```
  });
 });
</script>
```

3. Modifying Inline Styles

If you need to modify an element's inline style attribute, you can use the .attr() method in jQuery.

Example:

```
<p id="myText">This is a paragraph.</p>

<script>
 $(document).ready(function() {
  $("#myText").attr("style", "color: blue; font-size: 18px;");
 });
</script>
```

This will directly set the style attribute with the specified properties.

4. Getting CSS Property Values

You can also use the .css() method to retrieve the value of a CSS property.

Example:

```
<p id="myText">This is a paragraph.</p>

<script>
 $(document).ready(function() {
var color = $("#myText").css("color");
console.log("The color of the text is: " + color); // Logs the current color of the text
 });
</script>
```

5. Resetting Styles

To remove any inline style from an element, you can use the .css() method and set the property to an empty string.

Example:

```
<p id="myText" style="color: red;">This is a paragraph.</p>

<script>
 $(document).ready(function() {
  $("#myText").css("color", ""); // Removes the inline color style
 });
</script>
```

6. Animating Style Changes

jQuery provides methods like .animate() to gradually change CSS properties over time.

Example (Animating a Background Color):

```
$(selector).animate({property: value}, duration);
<p id="myText">This is a paragraph.</p>

<script>
  $(document).ready(function() {
    $("#myText").click(function() {
      $(this).animate({
        "font-size": "30px",
        "margin-left": "50px"
      }, 1000); // Animates the font size and margin over 1 second
    });
  });
</script>
```

Creating elements:

Creating elements dynamically with jQuery is straightforward and allows you to add new content to your webpage without needing to reload it. You can use methods like .append(), .prepend(), .before(), .after(), and .html() to create and insert new elements.

1. Creating an Element

You can create a new element using the jQuery function by passing the HTML string to it

```
$("<tagname>content</tagname>");
```

Example (Creating a New Paragraph):

```
$(document).ready(function() {
varnewParagraph = $("<p>This is a new paragraph.</p>");
  $("body").append(newParagraph); // Adds the new paragraph to the end of the body
});
```

2. Appending an Element

The .append() method adds content to the end of the selected elements.

```
$(selector).append(content);
```

Example:

```
<div id="container">
<h1>My Title</h1>
</div>

<script>
  $(document).ready(function() {
    $("#container").append("<p>This is an appended paragraph.</p>");
```

```
  });
</script>
```

3. Prepending an Element

The .prepend() method adds content to the beginning of the selected elements.

```
$(selector).prepend(content);
```

Example:

```
<div id="container">
<h1>My Title</h1>
</div>

<script>
  $(document).ready(function() {
    $("#container").prepend("<p>This is a prepended paragraph.</p>");
  });
</script>
```

4. Inserting Elements Before or After

You can use the .before() and .after() methods to insert content relative to the selected elements.

Example (Using .before()):

```
<div id="container">
<h1>My Title</h1>
</div>

<script>
  $(document).ready(function() {
    $("#container").before("<p>This paragraph is before the container.</p>");
  });
</script>
```

Example (Using .after()):

```
<div id="container">
<h1>My Title</h1>
</div>

<script>
  $(document).ready(function() {
    $("#container").after("<p>This paragraph is after the container.</p>");
  });
</script>
```

5. Replacing Existing Content

You can use the .html() method to replace the content of an element.

Example:

```
<div id="container">
<h1>My Title</h1>
</div>

<script>
  $(document).ready(function() {
    $("#container").html("<p>This content has been replaced.</p>");
  });
</script>
```

6. Creating Elements with Attributes and Classes

You can also create elements with attributes and classes directly.

Example:

```
<script>
  $(document).ready(function() {
varnewDiv = $("<div></div>")
      .addClass("myClass")       // Add a class
      .attr("id", "myDiv")      // Set an ID
      .text("This is a new div."); // Set text content

    $("body").append(newDiv); // Append the new div to the body
  });
</script>
```

7. Creating Multiple Elements

You can create and insert multiple elements by using a loop.

Example:

```
<div id="container"></div>

<script>
  $(document).ready(function() {
for (vari = 1; i <= 5; i++) {
      $("#container").append("<p>Paragraph " + i + "</p>");
    }
  });
</script>
```

8. Using .clone() to Duplicate Elements

You can clone existing elements with the .clone() method and then append them.

Example:

```
<div id="container">
```

```
<p id="original">This is the original paragraph.</p>
</div>

<script>
  $(document).ready(function() {
varclonedParagraph = $("#original").clone(); // Clones the original paragraph
    $("#container").append(clonedParagraph); // Appends the cloned paragraph
  });
</script>
```

Appending elements:

Appending elements in jQuery is a straightforward way to add content to your webpage dynamically. The .append() method is commonly used for this purpose. Here's a detailed guide on how to use it effectively, along with examples.

1. Using .append() Method

The .append() method adds the specified content as the last child of the selected elements.

Syntax:

```
$(selector).append(content);
```

Example:

```
<div id="container">
<h1>My Title</h1>
</div>

<script>
  $(document).ready(function() {
    $("#container").append("<p>This is an appended paragraph.</p>");
  });
</script>
```

In this example, the new paragraph is added inside the #container div, below the heading.

2. Appending Multiple Elements

You can append multiple elements at once by including them in the same .append() call.

Example:

```
<div id="container"></div>

<script>
  $(document).ready(function() {
```

```
    $("#container").append("<p>Paragraph 1</p><p>Paragraph 2</p>");
  });
</script>
```
Both paragraphs will be added to the #container div.

3. Appending jQuery Objects

You can also append jQuery objects or previously created elements.

Example:

```
<div id="container"></div>

<script>
  $(document).ready(function() {
varnewParagraph = $("<p>This is a new paragraph.</p>");
    $("#container").append(newParagraph); // Appending the jQuery object
  });
</script>
```

4. Appending Elements with Attributes and Classes

You can create elements with attributes and classes before appending them.

Example:

```
<div id="container"></div>

<script>
  $(document).ready(function() {
varnewDiv = $("<div></div>")
      .addClass("myClass")
      .attr("id", "myDiv")
      .text("This is a new div.");

    $("#container").append(newDiv); // Appending the new div with attributes
and class
  });
</script>
```

5. Appending After an Event

You can also append elements based on user interactions, like clicking a button.

Example:

Every time the button is clicked, a new paragraph will be added to the #container div.

6. Using .append() with HTML Structure

You can append more complex HTML structures, including lists or other nested elements.

Example:

```
<div id="container"></div>

<script>
  $(document).ready(function() {
    $("#container").append(`
<ul>
<li>Item 1</li>
<li>Item 2</li>
</ul>
    `);
  });
</script>
```

7. Appending Text or HTML

You can also append raw text or HTML content. When appending text, you can use the .text() method to avoid HTML injection.

Example (Appending Text):

```
<div id="container"></div>

<script>
  $(document).ready(function() {
    $("#container").append("This is a simple text appended to the div.");
  });
</script>
```

8. Chaining .append() with Other jQuery Methods

You can chain the .append() method with other jQuery methods for more complex manipulations.

Example:

```
<div id="container">
<h1>My Title</h1>
</div>
<script>
  $(document).ready(function() {
    $("#container")
      .append("<p>This is an appended paragraph.</p>")
      .css("border", "1px solid black"); // Append and then add a border
  });
</script>
```

Removing elements:

Removing elements in jQuery is easy and can be accomplished using several methods. The most common methods include .remove(), .empty(), and .detach(). Here's a detailed guide on how to use these methods effectively.

1. Using .remove() Method

The .remove() method is used to delete selected elements from the DOM along with their data and events.

Syntax:

```
$(selector).remove();
```

Example:

```
<div id="container">
<p>This paragraph will be removed.</p>
</div>

<script>
  $(document).ready(function() {
    $("#container p").remove(); // Removes the paragraph inside #container
  });
</script>
```

2. Using .empty() Method

The .empty() method removes all child elements from the selected elements but keeps the parent element intact. This means the element itself will remain, but its contents will be cleared.

Syntax:

```
$(selector).empty();
```

Example:

```
<div id="container">
<p>This will stay, but the content will be removed.</p>
<p>This paragraph will be removed.</p>
</div>

<script>
  $(document).ready(function() {
    $("#container").empty(); // Clears all child elements inside #container
  });
</script>
```

3. Using .detach() Method

The .detach() method removes the selected elements but keeps all data and events associated with them. This can be useful if you plan to re-insert the elements later.

Syntax:
$(selector).detach();
Example:

```
<div id="container">
<p>This paragraph will be detached.</p>
</div>

<script>
  $(document).ready(function() {
vardetachedElement = $("#container p").detach(); // Detaches the paragraph
console.log(detachedElement); // Can be reinserted later if needed
  });
</script>
```

4. Removing Elements on Events
You can also remove elements based on user interactions, like clicking a button.

Example:

```
<div id="container">
<p>Click the button to remove this paragraph.</p>
</div>
<button id="removeButton">Remove Paragraph</button>

<script>
  $(document).ready(function() {
    $("#removeButton").click(function() {
      $("#container p").remove(); // Removes the paragraph when the button is
clicked
    });
  });
</script>
```

5. Removing Elements by Class or Attribute
You can use jQuery selectors to remove elements that match specific criteria, such as class names or attributes.

Example (Removing by Class):

```
<div id="container">
<p class="remove-me">This will be removed.</p>
<p>This will stay.</p>
</div>

<script>
  $(document).ready(function() {
```

```
    $(".remove-me").remove(); // Removes all elements with the class 'remove-
me'
  });
</script>
```

6. Chaining Removal Methods

You can chain multiple jQuery methods, including removal methods, to perform several actions at once.

Example:

```
<div id="container">
<p>This will be removed and then emptied.</p>
<p>This will stay.</p>
</div>
<script>
  $(document).ready(function() {
    $("#container p").remove().end().empty(); // Remove paragraphs and then
empty the container
  });
</script>
```

Handling events:

Handling events in jQuery is a powerful way to add interactivity to your web pages. jQuery provides a simplified way to manage events like clicks, mouse movements, keyboard inputs, and more. Here's an overview of how to handle events effectively.

1. Basic Event Handling

You can bind event handlers using the .on() method, which allows you to specify the event type and the function to execute when the event occurs.

Handling events in jQuery is a powerful way to add interactivity to your web pages. jQuery provides a simplified way to manage events like clicks, mouse movements, keyboard inputs, and more. Here's an overview of how to handle events effectively.

```
$(selector).on("event", function);
```

Example (Click Event):

```
<button id="myButton">Click Me!</button>
<script>
  $(document).ready(function() {
    $("#myButton").on("click", function() {
alert("Button was clicked!");
    });
  });
</script>
```

2. Multiple Events

You can bind multiple events to the same element by passing a space-separated list of events.
Example:

```
<button id="myButton">Click or Hover</button>

<script>
  $(document).ready(function() {
    $("#myButton").on("click mouseenter", function() {
alert("Button was clicked or hovered over!");
    });
  });
</script>
```

3. Event Delegation

Event delegation allows you to attach a single event handler to a parent element that will respond to events on its child elements. This is useful for dynamically added elements.

Syntax:

```
$(parentSelector).on("event", childSelector, function);
```

Example:

```
<div id="container">
<button class="dynamicButton">Dynamic Button</button>
</div>
<script>
  $(document).ready(function() {
    $("#container").on("click", ".dynamicButton", function() {
alert("Dynamic button was clicked!");
    });

    // Adding a new button dynamically
    $("#container").append('<button class="dynamicButton">New Dynamic Button</button>');
  });
</script>
```

4. Using .off() to Remove Event Handlers

If you need to remove an event handler, you can use the .off() method.

Example:

```
<button id="myButton">Click Me!</button>
<script>
  $(document).ready(function() {
functionhandleClick() {
alert("Button was clicked!");
      $("#myButton").off("click"); // Remove the click event handler
    }
```

```
    $("#myButton").on("click", handleClick);
  });
</script>
```

5. Event Object

The event object is automatically passed to your event handler and contains information about the event.

Example (Using Event Object):

```
<button id="myButton">Click Me!</button>
<script>
  $(document).ready(function() {
    $("#myButton").on("click", function(event) {
console.log("Mouse position: (" + event.pageX + ", " + event.pageY + ")");
    });
  });
</script>
```

6. Preventing Default Behavior

You can prevent the default action of an event (like submitting a form) by using event.preventDefault().

Example (Preventing Form Submission):

```
<form id="myForm">
<input type="submit" value="Submit">
</form>
<script>
  $(document).ready(function() {
    $("#myForm").on("submit", function(event) {
event.preventDefault(); // Prevent the form from submitting
alert("Form submission prevented!");
    });
  });
</script>
```

7. Using .trigger() to Manually Trigger Events

You can manually trigger an event using the .trigger() method.

Example:

```
<button id="myButton">Click Me!</button>
<button id="triggerButton">Trigger Click</button>
<script>
  $(document).ready(function() {
    $("#myButton").on("click", function() {
alert("Button was clicked!");
    });

    $("#triggerButton").on("click", function() {
```

```
    $("#myButton").trigger("click"); // Manually trigger the click event on
#myButton
  });
});
</script>
```

8. Chaining Event Handlers

You can chain multiple event handlers together.

Example:

```
<button id="myButton">Click Me!</button>
<script>
  $(document).ready(function() {
    $("#myButton")
      .on("click", function() {
alert("Button clicked!");
      })
      .on("mouseenter", function() {
        $(this).css("background-color", "yellow");
      })
      .on("mouseleave", function() {
        $(this).css("background-color", "");
      });
  });
</script>
```