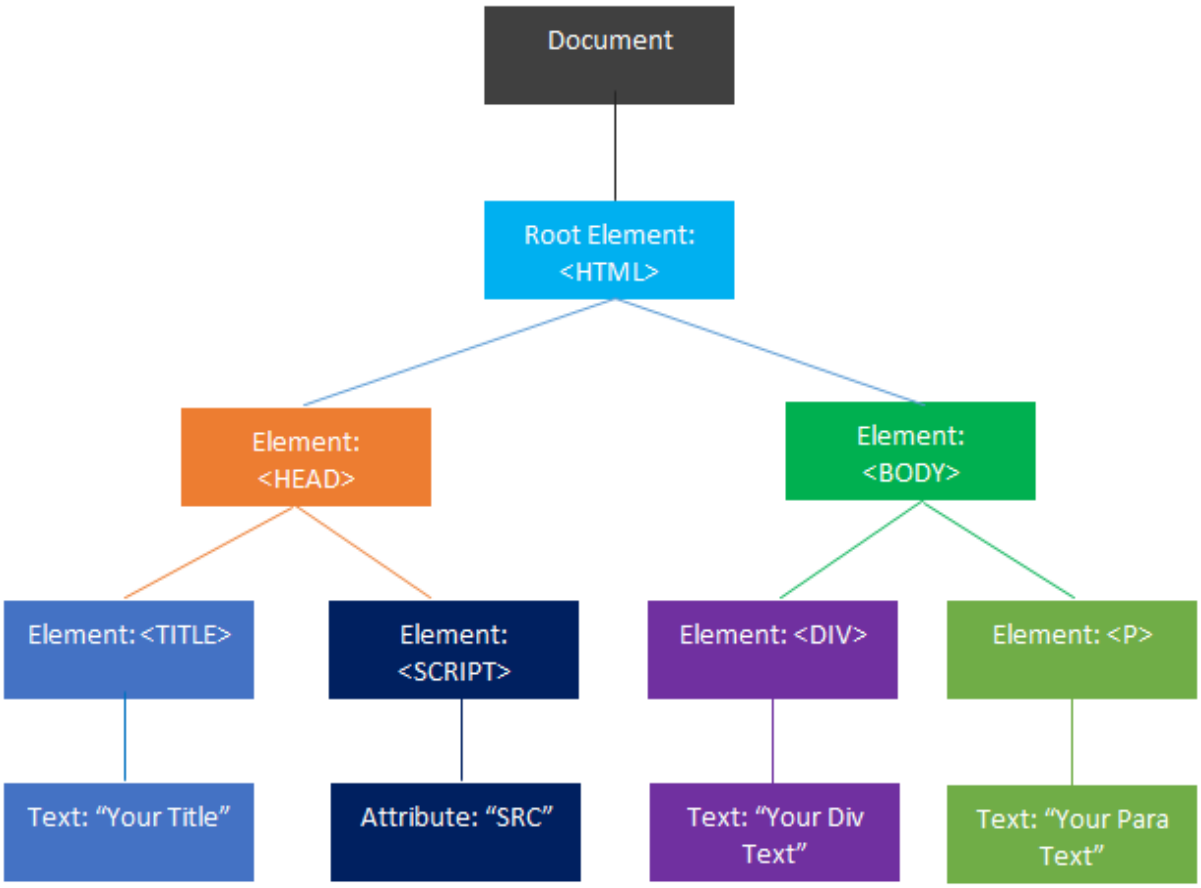
Document Object Model (DOM):

The DOM is a hierarchical representation of a web page's structure.

It provides a way to interact with HTML or XML documents using JavaScript.

When a web page is loaded, the browser creates a Document Object Model of the page.

The HTML DOM model is constructed as a tree of Objects:



A Simple DOM Tree

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page.

Direct Access From DOM(document Object):

```
console.log(document.title);

console.log(document.URL);
```

```
console.log(document.contentType); //text/html
console.log(document.characterSet); //utf-8
console.log(document.children);
console.log(document.addEventListener); //method
console.log(document.append);
console.log(document.appendChild);
console.log(document.body);
console.log(document.styleSheets);
console.log(document.images);
console.log(document.links);
console.log(document.COMMENT_NODE);
console.log(document.scripts);
console.log(document.close);
console.log(document.open); //method
console.log(document.cloneNode); //method
console.log(document.contains); //method
console.log(document.cookie); //important for server side sending small chunk of data.
console.log(document.createAttribute);
console.log(document.getElementById);
console.log(document.getElementsByTagName);
console.log(document.getElementsByClassName);
console.log(document.querySelector);
console.log(document.querySelectorAll);
console.log(document.write);
console.log(document.doctype);
console.log(document.createElement);
console.log(document.createComment);
console.log(document.createEvent);
```

Attribute methods in JS:

getAttribute(attributeName):

This method allows you to retrieve the value of a specific attribute of an element.

setAttribute(attributeName, value):

Use this method to set the value of a specific attribute for an element.

hasAttribute(attributeName):

This method checks if an element has a specific attribute and returns a boolean value.

removeAttribute(attributeName):

Use this method to remove a specific attribute from an element.

ClassList property:

In JavaScript, the classList property of an HTML element provides an interface to manipulate the class attributes of that element. The classList property is particularly useful for adding, removing, toggling, and checking the presence of CSS classes on an element. Here are some common methods and properties of the classList object:

classList.add(className1, className2, ...):

Adds one or more CSS classes to the element.

classList.remove(className1, className2, ...):

Removes one or more CSS classes from the element.

classList.toggle(className):

Toggles a CSS class on the element. If the class is present, it removes it; if it's absent, it adds it.

classList.contains(className):

Checks if the element has a specific CSS class and returns true or false.

innerText:

innerText is a property that represents the text content within an HTML element, excluding any HTML tags or elements inside the element.

It returns or sets only the plain text content and does not interpret or modify any HTML.

It is a safer option when you want to work specifically with the text inside an element and avoid issues related to HTML injection or unintentional code execution.

innerHTML:

innerHTML is a property that represents the HTML content within an HTML element, including any HTML tags or elements inside the element.It can be used to both read and set the HTML content, making it a powerful tool for dynamically generating or modifying content.

the main difference between innerText and innerHTML is that innerText deals with the plain text content of an element and does not consider HTML tags, while innerHTML deals with the HTML content, including any tags and elements within the element.

DOM inbuilt methods:

Accessing Elements:

- document.getElementById(id): Retrieves an element with the specified id.
- document.getElementsByClassName(className): Retrieves elements with the specified class name, returning them as an HTMLCollection.
- document.getElementsByTagName(tagName): Retrieves elements with the specified tag name, returning them as an HTMLCollection.
- document.querySelector(selector): Retrieves the first element that matches the specified CSS selector.
- document.querySelectorAll(selector): Retrieves all elements that match the specified CSS selector, returning them as a NodeList.

Creating and Modifying Elements:

- document.createElement(tagName): Creates a new HTML element with the specified tag name.
- element.appendChild(newChild): Appends a new child node (element or text) to an existing element.
- element.removeChild(child): Removes a child node from an element.
- element.innerHTML: Gets or sets the HTML content within an element (can be used for creating or modifying elements).

Styling Elements:

element.style.property: Allows you to access or modify CSS properties of an element directly (e.g., element.style.color).

Dealing with NodeList vs. HTMLCollection:

When you use methods like querySelectorAll or getElementsByTagName, you get a NodeList or HTMLCollection, respectively. These are array-like objects but not actual arrays. To convert them to an array, you can use the Array.from method or the spread operator:

EX:

```
const elements = document.querySelectorAll(".myClassName");

const elementArray = Array.from(elements); // Convert NodeList to an array
```

DOM Events:

Events in the context of the Document Object Model (DOM) in JavaScript refer to specific interactions or occurrences that take place in a web page. These interactions can include user actions (e.g., clicking a button, moving the mouse).

Event handling:

Event Handling in the DOM involves writing JavaScript code to respond to these events. Event handling allows you to define how your web page or application should react when these events occur.

addEventListener():

The addEventListener method in JavaScript is used to attach event handlers (functions) to DOM elements, allowing you to respond to specific events that occur on those elements. It is a fundamental method for event handling in web development.

syntax: element.addEventListener(eventType, eventHandler, options);

1. **eventType**: A string specifying the type of event you want to listen for (e.g., "click," "keydown," "submit").
2. **eventHandler:** A function that gets executed when the specified event occurs on the element.
3. **options (optional):** An optional object that can be used to configure the event listener, such as specifying whether the event should use capturing (true) or bubbling (false) as the propagation mode.

Event Object:

The event handler function receives an event object as a parameter. This object provides information about the event, such as its type, target element, and additional properties specific to the event type.

You can access this event object and use its properties to gather information about the event.

Types of events in DOM:

Click Event:

The click event occurs when a mouse click (left button) is detected on an element.

It's often used for handling user interactions such as button clicks, links, or toggling elements.

Mouseover and Mouseout Events:

mouseover occurs when the mouse pointer enters an element.

mouseout occurs when the mouse pointer leaves an element.

These events are useful for creating hover effects or tooltips.

Keyup, Keydown, and Keypress Events:

keydown occurs when a key is pressed down.

keyup occurs when a key is released.

keypress (less commonly used) occurs when a key is pressed and released.

These events are used to capture keyboard input and can be used for tasks like form validation or creating keyboard shortcuts.

Input Event:

The input event is triggered when the value of an input field or textarea changes, typically as the user types or pastes content.

It's commonly used to provide real-time feedback or perform actions as the user interacts with an input element.

Change Event:

The change event occurs when the value of an input field, select dropdown, or textarea changes and then loses focus (e.g., clicking outside the input).

It's often used for form validation or handling changes in select menus.

Submit Event:

The submit event occurs when a form is submitted, either by clicking a submit button or pressing Enter in a text input.

It's used to handle form submissions, validate user input, and send data to a server.

These events are fundamental to building interactive web applications.

Event Propagation:

Event Propagation refers to how events travel through the Document Object Model (DOM) tree. The DOM tree is the structure which contains parent/child/sibling elements in relation to each other.

Phases in Event Propagation:

Capturing Phase:

The event starts at the root of the DOM tree and moves down towards the target element.

During this phase, event handlers attached with the capture option set to true are executed on ancestor elements before reaching the target.

Target Phase:

This phase occurs when the event reaches the target element.

The event handler attached directly to the target element is executed in response to the event.

The event object contains information about the event and can be used by the handler to perform actions based on the event.

Bubbling Phase:

After the target phase, the event continues to propagate back up the DOM tree, moving away from the target element.

During this phase, event handlers attached without specifying the capture option or with false (the default) are executed on ancestor elements.

preventDefault() Method:

The preventDefault() method is used to prevent the default action associated with an event from occurring. Each type of event has a default action associated with it. For example, clicking on a link (an anchor element) typically navigates to the linked page, which is the default action for a click event on a link.

By calling event.preventDefault() within an event handler, you can prevent this default behavior from happening. This is commonly used to implement custom behavior when, for example, you want to handle navigation within a single-page application without a full page reload.

stopPropagation() Method:

The stopPropagation() method is used to stop the propagation of an event through the DOM tree. Events typically propagate from the target element (where the event occurred) up to the root of the DOM (the window object) in a process known as event bubbling.

By calling event.stopPropagation() within an event handler, you can prevent the event from continuing to propagate up or down the DOM tree. This can be useful when you want to handle an event only at a specific level of the DOM hierarchy without triggering other event handlers further up or down the tree.

localStorage and sessionStorage:

localStorage and sessionStorage are two web storage APIs in JavaScript that allow you to store key-value pairs locally on a user's web browser. They are useful for persisting data between page loads or maintaining data within a session, but they have different lifespans and use cases:

localStorage:

Lifetime: Data stored in localStorage persists even after the browser is closed and is available across browser sessions. It is stored until explicitly removed by the user or JavaScript code.

Scope: Data stored in localStorage is accessible to all windows or tabs from the same origin (i.e., the same protocol, domain, and port).

Use Cases:

- Storing user preferences or settings that should persist across sessions.
- Caching data to reduce server requests and improve app performance.
- Storing data for offline use in web applications.

sessionStorage:

Lifetime: Data stored in sessionStorage is available only for the duration of the page session. It is cleared when the page is closed or the session ends (e.g., when the user closes the browser or tab).

Scope: Data stored in sessionStorage is limited to the current window or tab. It is not accessible from other windows or tabs even if they are from the same origin.

Use Cases:

- Storing temporary data that should be available only during the current session.
- Implementing shopping cart functionality or managing state during a single browsing session.

Common Methods for localStorage and sessionStorage:

setItem(key, value):

Adds a key-value pair to the storage. If the key already exists, it updates the associated value.

getItem(key):

Retrieves the value associated with the specified key.

removeItem(key):

Removes the key-value pair associated with the specified key from storage.

clear():

Removes all key-value pairs from storage.

Regular expressions:

Regular expressions, often referred to as regex or regexp, are powerful patterns that allow you to search for and manipulate text in JavaScript and many other programming languages. They are used to match and manipulate strings based on specific patterns or rules. In JavaScript, you can work with regular expressions using the RegExp object or by using regex literals enclosed in forward slashes (/).