

## Module-02

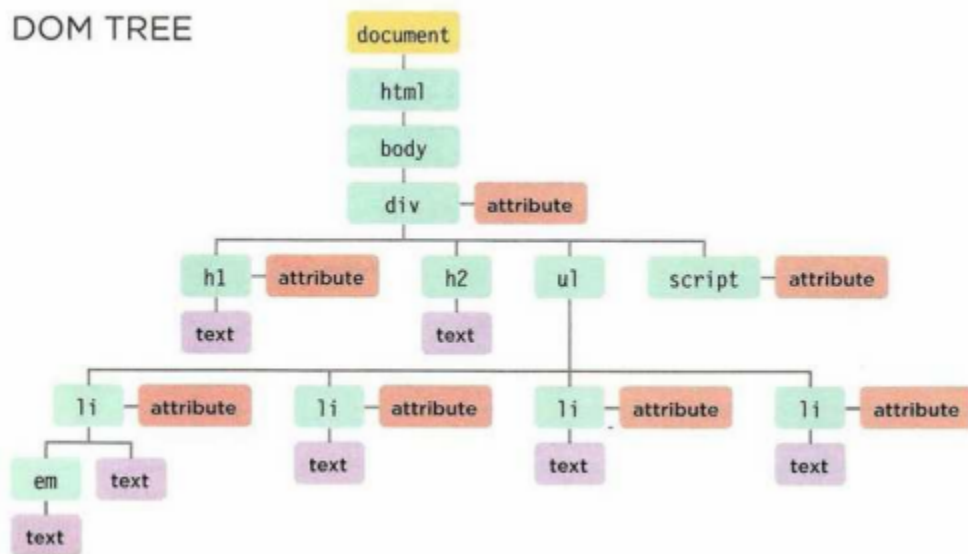
### Document Object Model

#### 1. Introduction to DOM

- DOM(Document Object Model) represents the structure of an HTML document as a tree of objects.

A **DOM Node** is any part of an HTML document (elements, text, attributes, or comments). JavaScript allows us to **create, modify, and remove** these nodes dynamically.

DOM TREE



#### **1** Types of DOM Nodes

Node Type	Description	Example
Element Node	Represents HTML elements	<code>&lt;div&gt;</code> , <code>&lt;p&gt;</code> , <code>&lt;button&gt;</code>
Text Node	Holds the text inside an element	"Hello World" inside <code>&lt;p&gt;Hello World&lt;/p&gt;</code>
Attribute Node	Represents attributes of an element	<code>id="myDiv"</code> , <code>class="box"</code>
Comment Node	Represents comments in the document	<code>&lt;!-- This is a comment --&gt;</code>

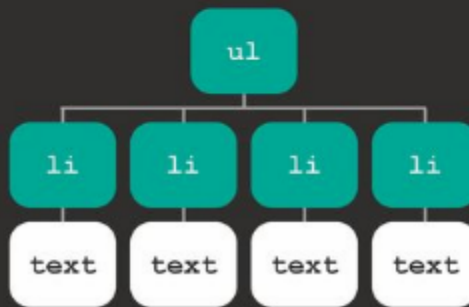
## ELEMENT NODES

```
<ul>  
  <li></li>  
  <li></li>  
  <li></li>  
  <li></li>  
</ul>
```



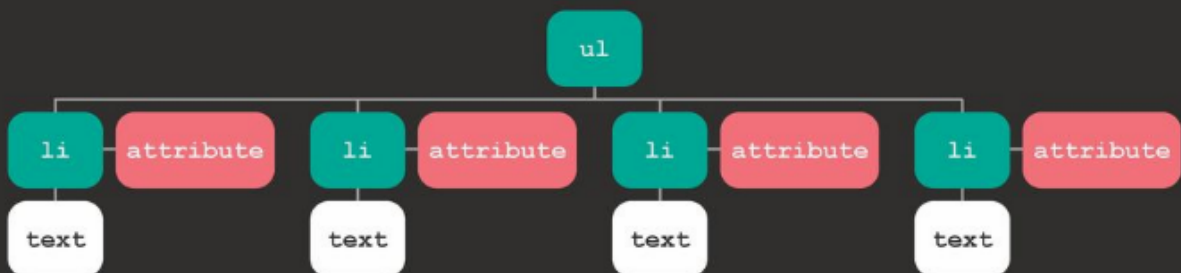
## TEXT NODES

```
<ul>
  <li>fresh figs</li>
  <li>pine nuts</li>
  <li>honey</li>
  <li>balsamic vinegar</li>
</ul>
```



## ATTRIBUTE NODES

```
<ul>
  <li id="one" class="hot">fresh figs</li>
  <li id="two" class="hot">pine nuts</li>
  <li id="three" class="hot">honey</li>
  <li id="four">balsamic vinegar</li>
</ul>
```



- 

**DOM Manipulation:** DOM Manipulation refers to dynamically changing HTML elements, styles, and attributes using JavaScript.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Hello World</p>
<button onclick="changeText()">Click Me</button>

<script>
function changeText() {
  let element = document.getElementById("demo");
  element.innerText = "Text Changed!";
  element.style.color = "red";
}
</script>

</body>
</html>
```

- Java Script can access and manipulate these elements dynamically.


## 2. Selecting Elements in DOM

### Selecting Elements

To modify elements, we first need to **select** them. JavaScript provides multiple methods for selection.

#### Different Ways to Select Elements

Here's an explanation of the table "Different Ways to Select Elements" in JavaScript with examples:

Method	Description	Example
<code>document.getElementById(id)</code>	Selects an element by its ID	<code>document.getElementById("demo")</code>
<code>document.getElementsByClassName(className)</code>	Selects all elements with a given class (returns an HTMLCollection)	<code>document.getElementsByClassName("myClass")</code> <code>[0]</code>
<code>document.getElementsByTagName(tagName)</code>	Selects all elements by tag name (returns an HTMLCollection)	<code>document.getElementsByTagName("p")</code>
<code>document.querySelector(selector)</code>	Selects the first matching element (by class, ID, or tag)	<code>document.querySelector(".myClass")</code>
<code>document.querySelectorAll(selector)</code>	Selects all matching elements 	<code>document.querySelectorAll("div")</code>

### 1) `getElementById`:

```
<p id="demo">Hello!</p>
<button onclick="changeText()">Click Me</button>
```

```
<script>
  function changeText() {
    document.getElementById("demo").innerText = "Hello, World!";
  }
</script>
```

### 2) `getElementsByClassName`:

```
<p class="text">Paragraph 1</p>
<p class="text">Paragraph 2</p>
```

```
<script>
  let elements = document.getElementsByClassName("text");
  for (let i = 0; i < elements.length; i++) {
    elements[i].style.color = "blue";
  }
</script>
```

```
}  
</script>
```

### 3) **getElementsByTagName(tagName)**

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
</ul>
```

```
<script>  
  let items = document.getElementsByTagName("li");  
  for (let i = 0; i < items.length; i++) {  
    items[i].innerText = "Updated Item " + (i + 1);  
  }  
</script>
```

### 4) **querySelector(selector):**

Select only one element

```
<p class="highlight">First highlighted text</p>  
<p class="highlight">Second highlighted text</p>
```

```
<script>  
  document.querySelector(".highlight").style.backgroundColor = "yellow";  
</script>
```

### 5) **querySelectorAll(selector)**

Select all elements


```
<div class="box">Box 1</div>  
<div class="box">Box 2</div>
```

```
<script>  
  let boxes = document.querySelectorAll(".box");  
  boxes.forEach(box => {  
    box.style.border = "2px solid red";  
  });  
</script>
```

## NodeList:

Collection of Element node is called node list.


**Functions that return nodelist are**



**getElementsByTagName('h1')**

Even though this query only returns one element, the method still returns a NodeList because of the potential for returning more than one element.


INDEX NUMBER & ELEMENT
0 <h1>

**getElementsByTagName('li')**

This method returns four elements, one for each of the <li> elements on the page. They appear in the same order as they do in the HTML page.


INDEX NUMBER & ELEMENT
0 <li id="one" class="hot">
1 <li id="two" class="hot">
2 <li id="three" class="hot">
3 <li id="four">

**getElementsByClassName('hot')**

This NodeList contains only three of the <li> elements because we are searching for elements by the value of their class attribute, not tag name.

INDEX NUMBER & ELEMENT
0 <li id="one" class="hot">
1 <li id="two" class="hot">
2 <li id="three" class="hot">

**querySelectorAll('li[id]')**

This method returns four elements, one for each of the <li> elements on the page that have an id attribute (regardless of the values of the id attributes).

INDEX NUMBER & ELEMENT
0 <li id="one" class="hot">
1 <li id="two" class="hot">
2 <li id="three" class="hot">
3 <li id="four">

## Accessing elements in node list

### Type1:

**let items = document.querySelectorAll(".item"); // Returns a NodeList**

**console.log(items[0]); // Logs the first element**

**console.log(items[1]); // Logs the second element**

### Type2:

**let items = document.querySelectorAll(".item");**

**console.log(items.item(1)); // Same as items[1]**

**Type3: let items = document.querySelectorAll(".item");**

```
items.forEach((item, index) => {  
    console.log(`Item ${index + 1}:`, item.textContent);  
});
```

```
Type4: let items = document.querySelectorAll(".item");  
let itemsArray = Array.from(items);
```

```
let itemTexts = itemsArray.map(item => item.textContent);  
console.log(itemTexts);
```

### **Looping through the nodelist**

#### **Example1:**

```
let items = document.querySelectorAll("li");  
  
for (let i = 0; i < items.length; i++) {  
    console.log(`Item ${i + 1}:`, items[i].textContent);  
}
```

#### **Example2:**

```
let divs = document.querySelectorAll("div");  
  
for (let div of divs) {  
    div.style.color = "blue";  
}
```

### **Creating DOM Nodes**

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

```
<button onclick="addElement()">Add Paragraph</button>
```

```
<script>
```

```
function addElement() {  
    let newPara = document.createElement("p"); // Create <p> element  
    newPara.innerText = "This is a new paragraph.";
```



```
    document.getElementById("container").appendChild(newPara); //
Append to div
}
</script>
```

### **Modifying DOM Nodes:**

```
<p id="demo">Original Text</p>
<button onclick="modifyText()">Change Text</button>
```

```
<script>
function modifyText() {
    let element = document.getElementById("demo");
    element.innerText = "Text Updated!";
    element.style.color = "blue";
}
</script>
```

### **Removing DOM Nodes**

```
<p id="removeMe">Click the button to remove me.</p>
<button onclick="removeElement()">Remove</button>
```

```
<script>
function removeElement() {
    let element = document.getElementById("removeMe");
    element.remove(); // Remove the paragraph
}
</script>
```

### **Replacing DOM Nodes**

```
<p id="oldPara">This will be replaced.</p>
<button onclick="replaceElement()">Replace</button>
```

```
<script>
function replaceElement() {
    let newPara = document.createElement("p");
    newPara.innerText = "This is the new paragraph!";
```

```
    let oldPara = document.getElementById("oldPara");  
    oldPara.parentNode.replaceChild(newPara, oldPara);  
  }  
</script>
```

## Cloning DOM Nodes

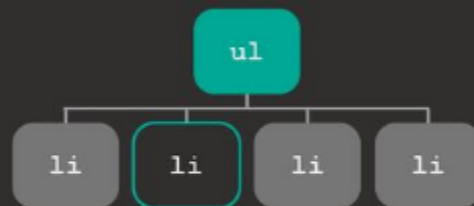
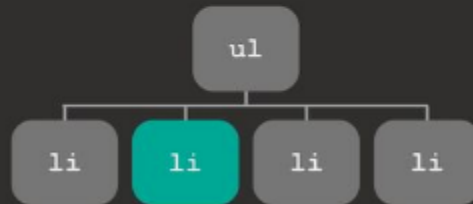
```
<p id="original">I am the original paragraph.</p>  
<button onclick="cloneElement()">Clone</button>  
  
<script>  
  function cloneElement() {  
    let original = document.getElementById("original");  
    let clone = original.cloneNode(true); // Clone the element  
    document.body.appendChild(clone); // Add clone to the body  
  }  
</script>
```

## Traversing the DOM in JavaScript

You can move from one node to another if it is a relation of it.

This is known as **traversing the DOM**.

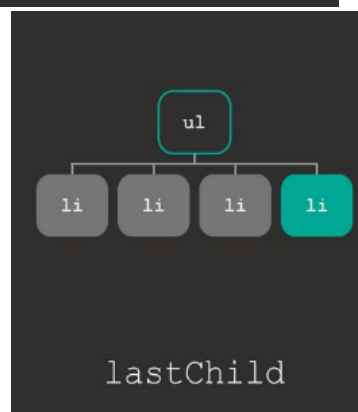
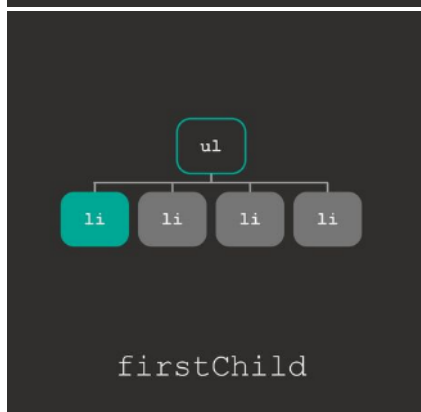
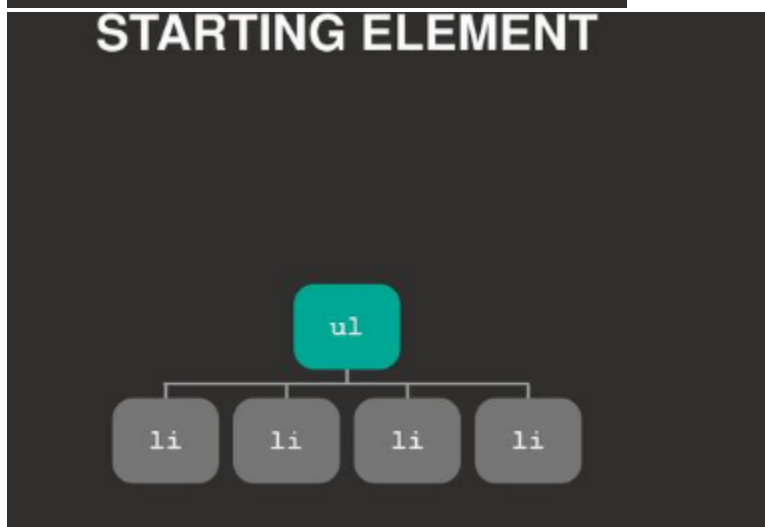
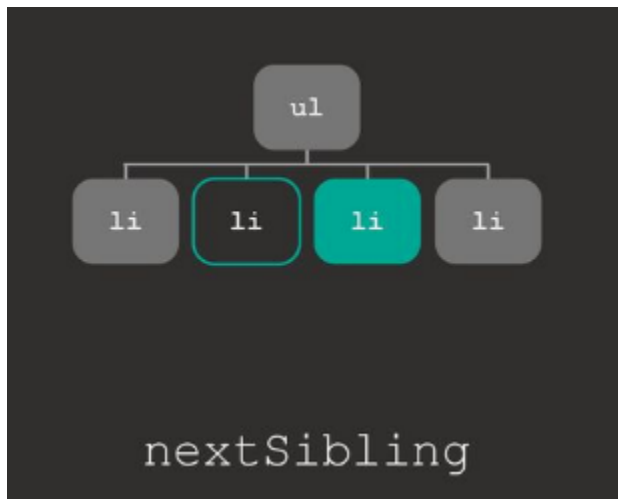
## STARTING ELEMENT



parentNode



previousSibling



## Parent, Child, and Sibling Relationships in DOM

The **DOM Tree** is structured as a hierarchy:

- **Parent Node** → The node that contains child nodes.
- **Child Node** → The nodes inside another node.
- **Sibling Nodes** → Nodes that share the same parent.

```
<div id="parent">
  <p id="child1">First Paragraph</p>
  <p id="child2">Second Paragraph</p>
</div>
```

## Accessing Parent Nodes

```
<p id="child">This is a paragraph.</p>
<button onclick="findParent()">Find Parent</button>
```

```
<script>
  function findParent() {
    let child = document.getElementById("child");
    console.log(child.parentNode); // Logs the parent element (usually
<body> or <div>)
  }
</script>
```

## Accessing Child Nodes

We can move down the DOM tree using:

- .children (Only element nodes)
- .firstElementChild
- .lastElementChild
- .childNodes (Includes text, comments, etc.)

```
<div id="parent">
  <p>First Child</p>
  <p>Second Child</p>
</div>
```

```
<script>
  let parent = document.getElementById("parent");
  console.log(parent.firstElementChild); // Logs the first <p>
  console.log(parent.lastElementChild); // Logs the last <p>
```

```
</script>
```

## Accessing Sibling Nodes

To move sideways between elements:

.nextElementSibling → Next element on the same level.

.previousElementSibling → Previous element on the same level.

```
<p id="para1">First Paragraph</p>
```

```
<p id="para2">Second Paragraph</p>
```

```
<script>
```

```
  let secondPara = document.getElementById("para2");
```

```
  console.log(secondPara.previousElementSibling); // Logs the first  
  paragraph
```

```
</script>
```

## Looping Through Child Nodes

```
<ul id="list">
```

```
  <li>Item 1</li>
```

```
  <li>Item 2</li>
```

```
  <li>Item 3</li>
```

```
</ul>
```

```
<script>
```

```
  let list = document.getElementById("list");
```

```
  let children = list.children; // Gets all <li> elements
```

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

```
    console.log(children[i].innerText);
```

```
  }
```

```
</script>
```

## Summary Table

Property	Description	Example
<code>.parentNode</code>	Gets parent node	<code>element.parentNode</code>
<code>.parentElement</code>	Gets parent element	<code>element.parentElement</code>
<code>.children</code>	Gets only child elements	<code>element.children</code>
<code>.childNodes</code>	Gets all child nodes (including text/comments)	<code>element.childNodes</code>
<code>.firstElementChild</code>	First child element	<code>element.firstElementChild</code>
<code>.lastElementChild</code>	Last child element	<code>element.lastElementChild</code>
<code>.nextElementSibling</code>	Next sibling element	<code>element.nextElementSibling</code>
<code>.previousElementSibling</code>	Previous sibling element	<code>element.previousElementSibling</code>

### Final Example: Traversing Parent, Child, and Sibling Nodes

```
<div id="container">
  <h2>Heading</h2>
  <p>Paragraph 1</p>
  <p id="middle">Paragraph 2</p>
  <p>Paragraph 3</p>
</div>
```

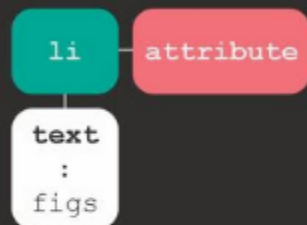
```
<script>
let middle = document.getElementById("middle");
console.log(middle.parentNode);      // Logs <div id="container">
console.log(middle.previousElementSibling); // Logs <p>Paragraph 1</p>
console.log(middle.nextElementSibling);  // Logs <p>Paragraph 3</p>
</script>
```

# Elements can contain:

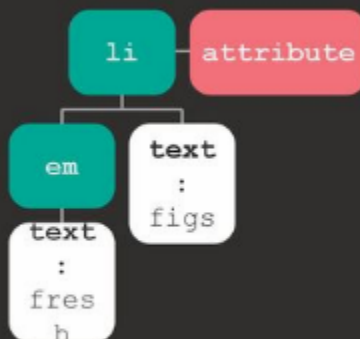
Text nodes

Element content

Attributes

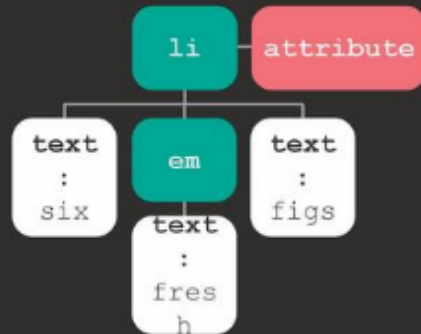


```
<li id="one">figs</li>
```



```
<li id="one"><em>fresh</em> figs</li>
```



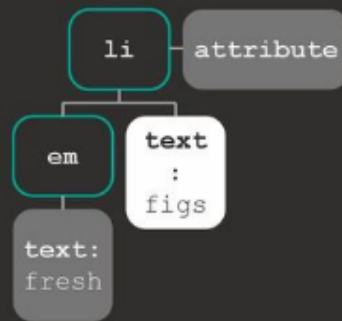


```
<li id="one">six <em>fresh</em> figs</li>
```

To access their content you can use:

- nodeValue on text nodes
- textContent for text content of elements
- innerHTML for text and markup

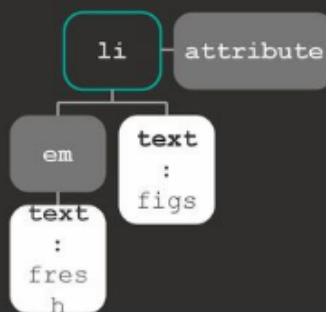
nodeValue works on text nodes



```
var el = document.getElementById('one');  
el.firstChild.nextSibling.nodeValue;
```

**returns:** figs

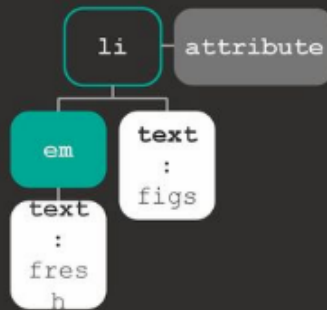
textContent just collects text content



```
document.getElementById('one').textContent;
```

**returns:** fresh figs

# innerHTML gets text and markup



```
document.getElementById('one').innerHTML;
```

**returns:** `<em>fresh</em> figs`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>nodeValue vs textContent vs innerHTML</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin-top: 50px;
    }
    .container {
      width: 50%;
      margin: auto;
      padding: 20px;
      border: 2px solid black;
    }
    button {
```

```

        margin: 10px;
        padding: 10px;
        font-size: 16px;
        cursor: pointer;
    }
</style>
</head>
<body>

    <h2>nodeValue vs textContent vs innerHTML</h2>

    <div class="container">
        <p id="demo">Hello, <b>World!</b></p>
    </div>

    <button onclick="getNodeValue()">Get nodeValue</button>
    <button onclick="gettextContent()">Get textContent</button>
    <button onclick="getinnerHTML()">Get innerHTML</button>

    <button onclick="setTextContent()">Set textContent</button>
    <button onclick="setinnerHTML()">Set innerHTML</button>

    <script>
        function getNodeValue() {
            let textNode = document.getElementById("demo").firstChild;
            alert("nodeValue: " + textNode.nodeValue); // May return null if
whitespace exists
        }

        function getTextContent() {
            let element = document.getElementById("demo");
            alert("textContent: " + element.textContent); // Gets only text,
ignores HTML tags
        }

        function getinnerHTML() {
            let element = document.getElementById("demo");

```

```

        alert("innerHTML: " + element.innerHTML); // Includes HTML tags
    }

    function settextContent() {
        let element = document.getElementById("demo");
        element.textContent = "This is new plain text!";
    }

    function setInnerHTML() {
        let element = document.getElementById("demo");
        element.innerHTML = "This is <strong>bold</strong> new
content!";
    }
</script>

</body>
</html>

```

### Working with attributes

1. Use a DOM query to select an element:

```
var el = document.getElementById('one');
```

2. Method gets attribute from element:

```
el.getAttribute('class');
```

Check for attribute and update it:

```
var el = document.getElementById('one');

if (el.hasAttribute('class') {
    el.setAttribute('class', 'cool');
}
```

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

</head>
<body>
  <h2>JavaScript Attribute Manipulation</h2>

  <a id="myLink" href="https://chatgpt.com/c/67bfed3e-ff50-8003-
831c-36163103642b">Click Me</a>
  <br><br>

  
  <br><br>

  <button id="myButton" onclick=change()>Submit</button>
  <script>
// Selecting Elements
function change()
{
```

```

let link = document.getElementById("myLink");
let image = document.getElementById("myImage");
let button = document.getElementById("myButton");

// Create / Set Attribute
link.setAttribute("href", "https://www.google.com");
link.setAttribute("target", "_blank"); // Opens in new tab
console.log("Updated Link:", link.getAttribute("href")); // Outputs:
https://www.google.com

// Change Attribute
image.src = "images1.jpg"; // Directly modifying attribute
console.log("Updated Image Source:", image.src); // Outputs: new-
image.jpg

// Get Attribute
console.log("Image Alt Text:", image.getAttribute("alt")); // Outputs:
Old Image

// 4 Remove Attribute
button.removeAttribute("disabled");
console.log("Button Disabled?:", button.hasAttribute("disabled")); //
Outputs: false
}
</script>

</body>
</html>

```

### **Cross site scripting (XSS) Attacks**

Cross-Site Scripting (XSS) is a type of security vulnerability where attackers **inject malicious scripts** into web applications. These scripts run in the browser of unsuspecting users, allowing attackers to steal data, manipulate content, or perform actions on behalf of users.

# Sources of untrusted data:

User creates a profile

Multiple contributors

Data from third-party sites

Files such as images / videos  
are uploaded

## Input Validation (Server-Side Security)

"Validate all input that is sent to the server"

- Before accepting user input, the **server must check and sanitize** it.
- Prevents malicious scripts from entering the **database** or being processed.
- This applies to **forms, URLs, cookies, and API requests**.

### ✓ Best Practices for Input Validation

- ✓ **Sanitize user input** – Remove harmful characters like `<script>`.
- ✓ **Use allowlists** – Only allow expected input formats.
- ✓ **Reject unexpected input** – E.g., validate email format before storing it

```
function sanitizeInput(input) {  
    return input.replace(/[<>'"/]/g, ""); // Removes harmful characters  
}
```

## Escaping Data on Output (Client-Side Security)

Escape data coming from the server"

- Any **data retrieved from the server** (database) must be **escaped** before displaying it in the **browser**.
- Prevents **injected scripts** from executing in the user's browser.

### ✓ Best Practices for Output Escaping



- ✓ **Use encoding techniques** – Convert `<script>` into safe HTML entities (`&lt;script&gt;`).
- ✓ **Use frameworks that escape content** – React, Angular, etc.
- ✓ **Avoid `innerHTML`** – Use `textContent` instead.

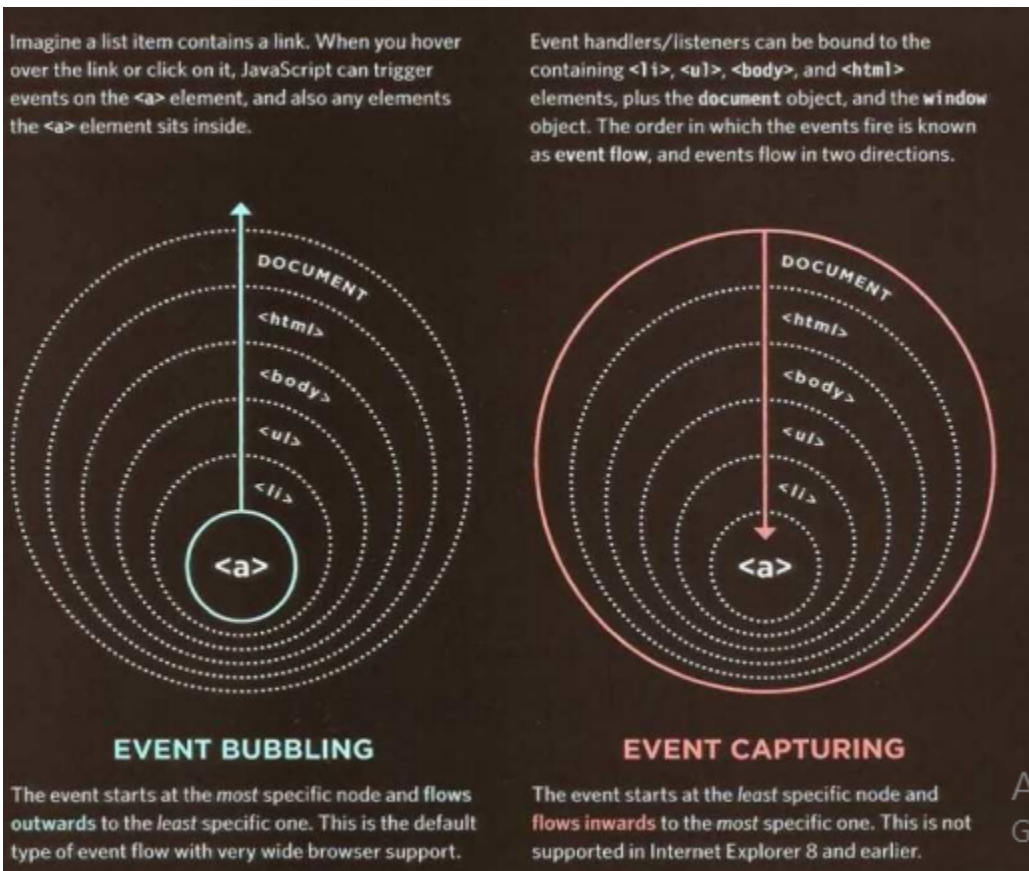
```
function encodeHTML(str) {  
  
    let div = document.createElement('div');  
  
    div.innerText = str;  
  
    return div.innerHTML; // Converts <script> to &lt;script&gt;  
  
}
```

## Events in java script

An **event** in JavaScript is an action or occurrence detected by the browser, such as a user clicking a button, pressing a key, or resizing the window. JavaScript allows developers to handle these events to create interactive web applications.

### **Event Flow:**

Event flow determines **how events propagate** through the **DOM (Document Object Model)** when an event occurs.



## Three Phases of Event Flow

1. **Capturing Phase (Trickling Down)**
  - The event starts at the **window** and moves **down** the DOM tree to the target element.
2. **Target Phase (Event Handling)**
  - The event reaches the **target element**, where the event listener is executed (if present).
3. **Bubbling Phase (Bubbling Up)**
  - After reaching the target, the event **bubbles up** back through the DOM tree to the **window**.

**Example:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Event Flow Example</title>
  <style>
    div { padding: 20px; border: 2px solid black; margin: 10px; }
```

```
</style>
</head>
<body>

  <div id="parent">
    Parent Div
    <div id="child">
      Child Div
    </div>
  </div>

  <script>
    const parent = document.getElementById("parent");
    const child = document.getElementById("child");

    // Capturing Phase (Trickling Down)
    parent.addEventListener("click", function() {
      console.log("Parent Capturing");
    }, true); // 'true' enables capturing

    child.addEventListener("click", function() {
      console.log("Child Capturing");
    }, true);

    // Bubbling Phase (Bubbling Up)
    child.addEventListener("click", function() {
      console.log("Child Bubbling");
    }, false); // 'false' enables bubbling

    parent.addEventListener("click", function() {
      console.log("Parent Bubbling");
    }, false);

  </script>

</body>
</html>
```

### 3. Event Delegation

- Used for handling events dynamically on elements that may not exist at the start.

Example:

Without Event Delegation (Inefficient)

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

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

```
  <title>Without Event Delegation</title>
```

```
</head>
```

```
<body>
```

```
  <button class="item">Item 1</button>
```

```
  <button class="item">Item 2</button>
```

```
  <button class="item">Item 3</button>
```

```
<script>
```

```
  // Adding separate event listeners to each button (inefficient)
```

```
  document.querySelectorAll(".item").forEach(button => {
```

```
    button.addEventListener("click", function() {
```

```
      console.log("Clicked:", this.textContent);
```

```
    });
```

```
  });
```

```
</script>
```

```
</body>
```

```
</html>
```

## Using Event Delegation (Efficient)

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

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

```
  <title>Event Delegation Example</title>
```

```
</head>
```

```
<body>
```

```
  <div id="parent">
```

```
    <button class="item">Item 1</button>
```

```
    <button class="item">Item 2</button>
```

```
    <button class="item">Item 3</button>
```

```
  </div>
```

```
  <button onclick="addNewItem()">Add New Item</button>
```

```
<script>
```

```
  const parent = document.getElementById("parent");
```

```
// Using event delegation - adding event listener to parent
parent.addEventListener("click", function(event) {
    if (event.target.classList.contains("item")) {
        console.log("Clicked:", event.target.textContent);
    }
});

// Function to dynamically add a new button
function addNewItem() {
    const newButton = document.createElement("button");
    newButton.textContent = "New Item";
    newButton.classList.add("item");
    parent.appendChild(newButton);
}

</script>

</body>
</html>
```

Types of Events:

Here is a selection of the events that occur in the browser while you are browsing the web. Any of these events can be used to trigger a function in your JavaScript code.

**UI EVENTS**      Occur when a user interacts with the browser's user interface (UI) rather than the web page

EVENT	DESCRIPTION
load	Web page has finished loading
unload	Web page is unloading (usually because a new page was requested)
error	Browser encounters a JavaScript error or an asset doesn't exist
resize	Browser window has been resized
scroll	User has scrolled up or down the page

**KEYBOARD EVENTS**      Occur when a user interacts with the keyboard (see also input event)

EVENT	DESCRIPTION
keydown	User first presses a key (repeats while key is depressed)
keyup	User releases a key
keypress	Character is being inserted (repeats while key is depressed)

**MOUSE EVENTS**      Occur when a user interacts with a mouse, trackpad, or touchscreen

EVENT	DESCRIPTION
click	User presses and releases a button over the same element
dblclick	User presses and releases a button twice over the same element
mousedown	User presses a mouse button while over an element
mouseup	User releases a mouse button while over an element
mousemove	User moves the mouse (not on a touchscreen)
mouseover	User moves the mouse over an element (not on a touchscreen)
mouseout	User moves the mouse off an element (not on a touchscreen)

## TERMINOLOGY

### EVENTS FIRE OR ARE RAISED

When an event has occurred, it is often described as having **fired** or been **raised**. In the diagram on the right, if the user is tapping on a link, a click event would fire in the browser.



### EVENTS TRIGGER SCRIPTS

Events are said to **trigger** a function or script. When the click event fires on the element in this diagram, it could trigger a script that enlarges the selected item.

### FOCUS EVENTS

Occur when an element (e.g., a link or form field) gains or loses focus

EVENT	DESCRIPTION
focus / focusin	Element gains focus
blur / focusout	Element loses focus

### FORM EVENTS

Occur when a user interacts with a form element

EVENT	DESCRIPTION
input	Value in any <input> or <textarea> element has changed (IE9+) or any element with the contenteditable attribute
change	Value in select box, checkbox, or radio button changes (IE9+)
submit	User submits a form (using a button or a key)
reset	User clicks on a form's reset button (rarely used these days)
cut	User cuts content from a form field
copy	User copies content from a form field
paste	User pastes content into a form field
select	User selects some text in a form field

### MUTATION EVENTS\*

Occur when the DOM structure has been changed by a script  
\* To be replaced by mutation observers (see p284)

EVENT	DESCRIPTION
DOMSubtreeModified	Change has been made to document
DOMNodeInserted	Node has been inserted as a direct child of another node
DOMNodeRemoved	Node has been removed from another node
DOMNodeInsertedIntoDocument	Node has been inserted as a descendant of another node
DOMNodeRemovedFromDocument	Node has been removed as a descendant of another node

## HOW EVENTS TRIGGER JAVASCRIPT CODE

When the user interacts with the HTML on a web page, there are three steps involved in getting it to trigger some JavaScript code. Together these steps are known as **event handling**.

1

Select the **element** node(s) you want the script to respond to.

2

Indicate which **event** on the selected node(s) will trigger the response.

3

State the **code** you want to run when the event occurs.



**Example1:**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>JavaScript Event Examples</title>
  <style>
    #hoverDiv, #dragItem, #dropZone, #noRightClick {
      margin-top: 10px;
      padding: 20px;
      text-align: center;
      font-weight: bold;
    }
    #hoverDiv {
      width: 150px;
      height: 50px;
      background-color: lightblue;
    }
    #dragItem {
      width: 100px;
      height: 100px;
      background-color: red;
      color: white;
      cursor: grab;
    }
    #dropZone {
      width: 200px;
      height: 100px;
      background-color: lightgray;
      border: 2px dashed black;
    }
    #noRightClick {
      background-color: orange;
    }
  </style>
```

</head>

<body>

<h1>JavaScript Event Examples</h1>

<!-- Mouse Events -->

<h2>Mouse Events</h2>

<button id="clickBtn">Click Me</button>

<button id="dblClickBtn">Double Click Me</button>

<div id="hoverDiv">Hover Over Me</div>

<!-- Keyboard Events -->

<h2>Keyboard Events</h2>

<input type="text" id="keyInput" placeholder="Type something...">

<!-- Form Events -->

<h2>Form Events</h2>

<form id="myForm">

<input type="text" id="nameInput" placeholder="Enter your name" required>

<button type="submit">Submit</button>

</form>

<!-- Window Events -->

<h2>Window Events</h2>

<button id="resizeBtn">Resize the Window</button>

<!-- Clipboard Events -->

<h2>Clipboard Events</h2>

<input type="text" id="copyInput" value="Try copying this text">

<!-- Drag & Drop Events -->

<h2>Drag & Drop Events</h2>

<div id="dragItem" draggable="true">Drag Me</div>

<div id="dropZone">Drop Here</div>

<!-- Media Events -->

```

<h2>Media Events</h2>
<video id="videoPlayer" width="300" controls>
  <source src="https://www.w3schools.com/html/mov_bbb.mp4"
type="video/mp4">
</video>

<!-- Miscellaneous Events -->
<h2>Miscellaneous Events</h2>
<div id="noRightClick">Right-click Disabled Here</div>

<script>
  // Mouse Events
  document.getElementById("clickBtn").addEventListener("click",
function () {
    alert("Button Clicked!");
  });

document.getElementById("dblClickBtn").addEventListener("dblclick",
function () {
  alert("Button Double Clicked!");
});

let hoverDiv = document.getElementById("hoverDiv");
hoverDiv.addEventListener("mouseover", function () {
  hoverDiv.style.backgroundColor = "yellow";
});
hoverDiv.addEventListener("mouseout", function () {
  hoverDiv.style.backgroundColor = "lightblue";
});

// Keyboard Events
document.getElementById("keyInput").addEventListener("keydown",
function (event) {
  console.log("Key Pressed: " + event.key);
});

// Form Events

```

```
document.getElementById("myForm").addEventListener("submit",
function (event) {
    event.preventDefault(); // Prevents page refresh
    alert("Form Submitted: " +
document.getElementById("nameInput").value);
});
```

```
// Window Events
window.addEventListener("resize", function () {
    console.log("Window Resized: " + window.innerWidth + " x " +
window.innerHeight);
});
```

```
// Clipboard Events
document.getElementById("copyInput").addEventListener("copy",
function () {
    alert("Text Copied!");
});
```

```
// Drag & Drop Events
let dragItem = document.getElementById("dragItem");
let dropZone = document.getElementById("dropZone");
```

```
dragItem.addEventListener("dragstart", function (event) {
    event.dataTransfer.setData("text", event.target.id);
});
```

```
dropZone.addEventListener("dragover", function (event) {
    event.preventDefault();
});
```

```
dropZone.addEventListener("drop", function (event) {
    event.preventDefault();
    let data = event.dataTransfer.getData("text");
    dropZone.appendChild(document.getElementById(data));
});
```

```

// Media Events
let video = document.getElementById("videoPlayer");

video.addEventListener("play", function () {
    console.log("Video Playing...");
});

video.addEventListener("pause", function () {
    console.log("Video Paused.");
});

// Miscellaneous Events
document.getElementById("noRightClick").addEventListener("contextmenu", function (event) {
    event.preventDefault();
    alert("Right-click Disabled!");
});

</script>

</body>
</html>

```

## Example2:

```

<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Events</title>
</head>
<body>
    <h1>JavaScript Events</h1>

    <!-- Mouse Events -->
    <button id="mouse-btn">Click Me!</button>
    <div id="mouse-div"></div>

```

```

<!-- Keyboard Events -->
<input id="keyboard-input" type="text" placeholder="Type something...">
</div id="keyboard-div"></div>

<!-- Form Events -->
<form id="form">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name"><br><br>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email"><br><br>
  <input type="submit" value="Submit">
</form>
</div id="form-div"></div>

<!-- Scroll Events -->
<div id="scroll-div" style="height: 500px; overflow-y: scroll;">
  Scroll me!
</div>
<div id="scroll-status"></div>

<script src="script.js"></script>
</body>
</html>

```

```

// script.js
// Mouse Events
const mouseBtn = document.getElementById('mouse-btn');
const mouseDiv = document.getElementById('mouse-div');

mouseBtn.addEventListener('click', () => {
  mouseDiv.innerText = 'Button clicked!';
});

mouseBtn.addEventListener('dblclick', () => {

```

```
    mouseDiv.innerText = 'Button double-clicked!';  
  });
```

```
mouseBtn.addEventListener('mouseover', () => {  
  mouseDiv.innerText = 'Mouse over button!';  
});
```

```
mouseBtn.addEventListener('mouseout', () => {  
  mouseDiv.innerText = 'Mouse out of button!';  
});
```

```
// Keyboard Events
```

```
const keyboardInput = document.getElementById('keyboard-input');  
const keyboardDiv = document.getElementById('keyboard-div');
```

```
keyboardInput.addEventListener('keydown', (e) => {  
  keyboardDiv.innerText = `Key pressed: ${e.key}`;  
});
```

```
keyboardInput.addEventListener('keyup', (e) => {  
  keyboardDiv.innerText = `Key released: ${e.key}`;  
});
```

```
keyboardInput.addEventListener('keypress', (e) => {  
  keyboardDiv.innerText = `Key pressed: ${e.key}`;  
});
```

```
// Form Events
```

```
const form = document.getElementById('form');  
const formDiv = document.getElementById('form-div');
```

```
form.addEventListener('submit', (e) => {  
  e.preventDefault();  
  const name = document.getElementById('name').value;  
  const email = document.getElementById('email').value;  
  formDiv.innerText = `Form submitted! Name: ${name}, Email: ${email}`;  
});
```

```
// Scroll Events
const scrollDiv = document.getElementById('scroll-div');
const scrollStatus = document.getElementById('scroll-status');

scrollDiv.addEventListener('scroll', () => {
  const scrollTop = scrollDiv.scrollTop;
  scrollStatus.innerText = `Scrolled to: ${scrollTop}px`;
});
```

## Mutation Events

Event Name	Triggered When...
<code>DOMNodeInserted</code>	A new node is added.
<code>DOMNodeRemoved</code>	A node is removed.
<code>DOMSubtreeModified</code>	Any modification occurs in the subtree.
<code>DOMNodeInsertedIntoDocument</code>	A node is added to the document.
<code>DOMNodeRemovedFromDocument</code>	A node is removed from the document.
<code>DOMAttrModified</code>	An attribute is changed.
<code>DOMCharacterDataModified</code>	The text inside a node is modified.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Mutation Events (Deprecated)</title>
</head>
<body>

  <div id="container">Hello, World!</div>
  <button onclick="changeContent()">Change Content</button>

  <script>
    const container = document.getElementById("container");
```



```

    // Deprecated mutation events
    container.addEventListener("DOMSubtreeModified",
function(event) {
    console.log("DOM modified!", event);
});

    function changeContent() {
        container.textContent = "Updated Text!";
    }
</script>

</body>
</html>

```

## **Mutation Observer**

```

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

    <div id="content">Hello, World!</div>
    <button onclick="changeText()">Change Text</button>

    <script>
        // Select the target element
        const targetNode = document.getElementById("content");

        // Create MutationObserver
        const observer = new MutationObserver(function (mutationsList) {
            mutationsList.forEach(mutation => {
                if (mutation.type === "childList") {

```

```
        console.log("Child nodes changed:", mutation);
    }
    if (mutation.type === "attributes") {
        console.log("Attribute changed:", mutation);
    }
});
});
```

```
// Observer Configuration
```

```
const config = { childList: true, attributes: true };
```

```
// Start Observing
```

```
observer.observe(targetNode, config);
```

```
// Function to modify DOM
```

```
function changeText() {
```

```
    targetNode.textContent = "Text Updated!";
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

## Module 1:

1. Create an array of 5 cities and perform the following operations: Log the total number of cities. Add a new city at the end. Remove the first city. Find and log the index of a specific city. **(Lab Program)**
2. Explain the different data types and its literals in JavaScript. **(Refer Notes)**
3. Illustrate how an object can be created in JavaScript using direct method and also using constructor method. List and Explain different inbuilt objects of Javascript **(Refer Notes and Textbook)**
4. Illustrate the following with example. **(Refer Text Book)**
  - i) Function creation and calling
  - ii) Function Expression and Anonymous Function
  - iii) Immediately Invoked Function Expression
5. Illustrate with a programming example how array can be created and displayed. **(Refer Notes)**
6. Explain different looping statements available in Javascript with example for each
7. Read a string from the user, Find its length. Extract the word "JavaScript" using substring() or slice(). Replace one word with another word and log the new string. Write a function isPalindrome(str) that checks if a given string is a palindrome (reads the same backward). **(Lab Program)**
8. Create an object student with properties: name (string), grade (number), subjects (array), displayInfo() (method to log the student's details) Write a script to dynamically add a passed property to the student object, with a value of true or false based on their grade. Create a loop to log all keys and values of the student object. **(Lab Program)**

## Module 2:

1. Illustrate the different methods used for selecting single and multiple elements in DOM with example for each. **(Refer Notes and Textbook)**
2. Illustrate with an example, how an element and text node can be created and added to existing tree. **(Refer Notes and Textbook)**
3. Illustrate with an example how a node can be removed from the DOM tree **(Refer Notes and Textbook)**
4. Explain the following with example for each **(Refer Notes and Textbook)**  
InneterHTML, InnerText, Style.property, attribute.value, nodeValue
5. Explain the following with example for each. **(Refer Notes and Textbook)**  
getAttribute(), setAttribute(), hasAttribute(), removeAttribute()
6. Illustrate Different Javascript events under the category of UI/UX events, Mouse Events, Form Events, Key Board Events, Focus Events. **(Refer Notes and Textbook)**
7. Explain three different ways of binding event to element in JavaScript with example for each. **(Refer Notes and Textbook)**

## Module 3:

1. What does MERN stands for? Explain the components of MERN stack in detail **(Refer Notes)**

2. Illustrate how to use React and ReactDOM in a single HTML to render Hello Message without server. **(Refer Notes )**