JAVASCRIPT DOM MANIPULATION & EVENTS

ABSTRACT

This unit focuses on interacting with web pages using the Document Object Model (DOM). It covers selecting and manipulating elements, changing content, attributes, and styles dynamically. Learners also study event handling, including mouse, keyboard, form, and input events, along with the use of addEventListener and the event object to manage user interactions effectively.

Unit – 9 JS DOM MANIPULATION & EVENTS

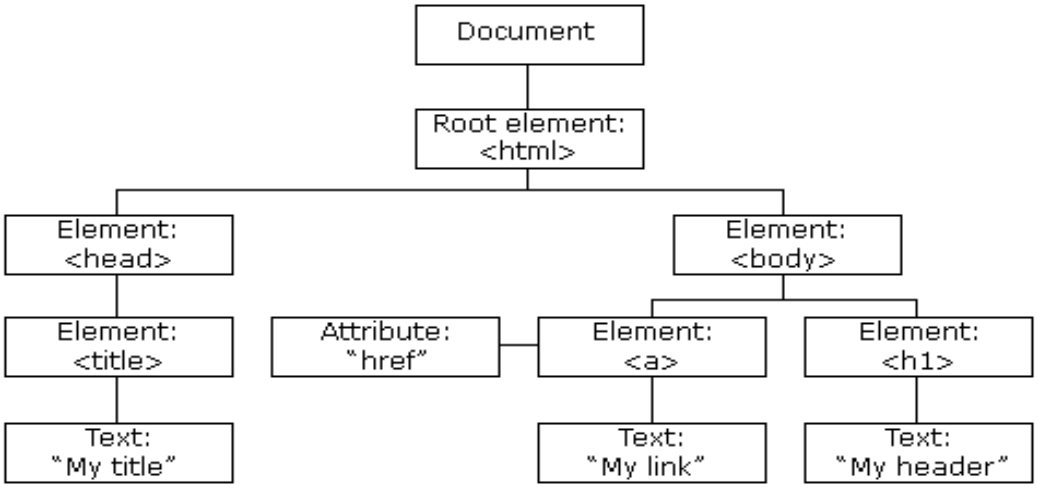
Document Object Model (DOM)

With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

When a web page is loaded, the browser creates a Document Object Model of the page. Every web page resides inside a browser window which can be considered as an object.

- The HTML DOM model is constructed as a tree of Objects.
- The HTML DOM Tree of Objects

The Document object has various properties that refer to other objects which allow access to and modification of document content.



What is the HTML DOM?

The HTML DOM is a standard object model and programming interface for HTML. It defines:

- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

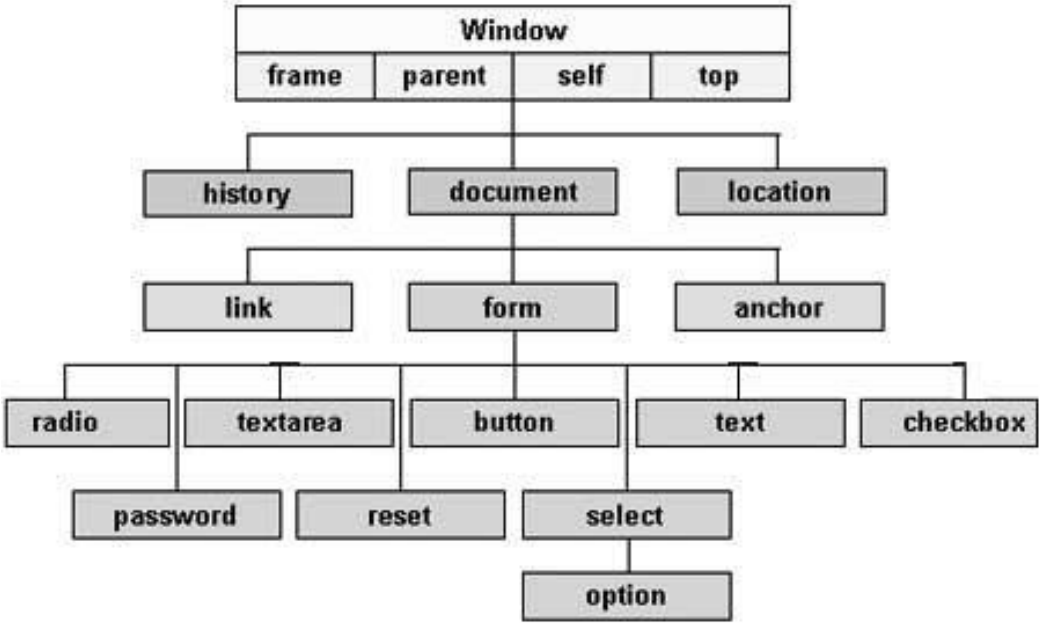
• **Window object** - Top of the hierarchy. It is the outmost element of the object hierarchy.

• **Document object** - Each HTML document that gets loaded into a window becomes a DOM M
document object. The document contains the contents of the page.

• **Form object** - Everything enclosed in the <form>...</form> tags sets the form object.

• **Form control elements** - The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects -



DOM Manipulation follows this sequence:
document → select element → modify content / attributes / style

The HTML DOM is an API (Programming Interface) for JavaScript:

- JavaScript can add/change/remove HTML elements
- JavaScript can add/change/remove HTML attributes
- JavaScript can add/change/remove CSS styles
- JavaScript can react to HTML events
- JavaScript can add/change/remove HTML events

The Document object has various properties that refer to other objects which allow access to and modification of document content.

Dom Manipulation

Topic / Method	Description	Basic Requirement	Syntax
document object	Represents the entire HTML document	HTML page loaded in browser	document.property/method
Element Selection Methods			
getElementById()	Finds ONE element using unique id	Element must have id attribute	document.getElementById("id")
getElementsByClassName()	Finds ALL elements with same class	Elements must share same class	document.getElementsByClassName("class")
querySelector()	Selects FIRST matching element (CSS selector)	Valid CSS selector	document.querySelector("selector")
querySelectorAll()	Selects ALL matching elements	Valid CSS selector	document.querySelectorAll("selector")
Content Manipulation			
innerHTML	Gets or sets HTML content inside element	Valid HTML element	element.innerHTML="text"
innerText	Gets or sets ONLY text (no HTML)	Valid HTML element	element.innerText="text"
Attribute Manipulation			
setAttribute()	Sets/changes attribute value	Element reference	element.setAttribute(name,value)
getAttribute()	Gets attribute value	Existing attribute	element.getAttribute(name)
removeAttribute()	Removes attribute from element	Existing attribute	element.removeAttribute(name)
Style Manipulation			
Style Manipulation	Changes CSS styles dynamically	Element reference	element.style.property=value

Document object

What is it?
document is the **main entry point** to the HTML page.
It represents the **entire web page loaded in the browser**.

Why we use it
- Without document, JavaScript **cannot access HTML**
- All DOM methods come from document

When to use
- Whenever you want to **read, write, or modify HTML**

Example

```
<script>
document.write("Welcome to DOM");
</script>
```

Explanation
- document.write() writes content **directly to the page**
- Using document.write() after page load can overwrite the entire document.

DOM methods: Find HTML Element

1. getElementById()

The most common way to access an HTML element is to use the id of the element. In the example above the getElementById method used id="demo" to find the element.

What is it?
Selects **ONE element** using its **unique ID**

Why we use it
- Fastest and simplest DOM method
- IDs are unique → only one element returned

When to use
- When you know the element's **id**
- When only **one element** is needed

Example

```
<p id="p1">Hello</p>
<script>
document.getElementById("p1").innerHTML = "Hi";
</script>
```

Explanation
- Finds <p id="p1">
- Changes its content to "Hi"
- Returns null if ID does not exist

2. getElementsByClassName()

What is it?
Selects **ALL elements** with the same class name

Why we use it

- Classes are used for **grouping elements**
- Returns **HTMLCollection** (array-like)

When to use

- When multiple elements share **same class**
- When you want to apply **same change to many elements**

Example

```
<p class="c1">Text 1</p>
<p class="c1">Text 2</p>
<script>
var x = document.getElementsByClassName("c1");
x[0].innerHTML = "Changed";
</script>
```

Explanation

- x contains **both <p> elements**
- x[0] accesses the first one

Using FOR loop (IMPORTANT)

```
<p class="c1">Text 1</p>
<p class="c1">Text 2</p>
<script>
var x = document.getElementsByClassName("c1");
for (var i = 0; i < x.length; i++) {
   x[i].innerHTML = "Changed using for loop";
}
</script>
```

Why loop is needed
Because **multiple elements** are returned

Counting HTML Elements by Class Name

```
<p class="c1">Text 1</p>
<p class="c1">Text 2</p>
<script>
var x = document.getElementsByClassName("c1").length;
console.log(x)  //  This will log 2 as two <p> used
</script>
```

3. querySelector()

What is it?
Selects the **FIRST matching element** using a **CSS selector**

Why it is powerful

- Uses **CSS rules**
- Can select:
  - tag (p)
  - class (.c1)
  - id (#p1)
  - combinations (div p, .box p)

When to use

- When selection is **complex**
- When you want **modern, readable code**

Example

```
<p>Paragraph</p>
<script>
document.querySelector("p").style.color = "red";
</script>
```

Explanation

- Finds the **first <p>**
- Changes text color to red

More querySelector() Examples

```
document.querySelector("#p1");     // by ID
document.querySelector(".c1");     // by class
document.querySelector("div p");   // p inside div
```

Even if multiple elements match, it returns **only the first one**

Example:

```
<div>
   <p>Inside</p>
</div>
<p>outside</p>
   <script>
  document.querySelector("div p").innerHTML = "Hello"
   </script>
```

Output:

Hello

outside

4. querySelectorAll()

What is it?
Selects **ALL matching elements** using CSS selectors

Why we use it
- More flexible than getElementsByClassName()
- Returns **NodeList**

When to use
- When selecting **multiple elements**
- When using **CSS selectors**

Example

```
<p>Para 1</p>
<p>Para 2</p>
<script>
var x = document.querySelectorAll("p");
x[1].innerHTML = "Second paragraph";
</script>
```

Explanation
- x contains all <p> elements
- x[1] refers to second paragraph

Using FOR loop

```
<script>
var x = document.querySelectorAll("p");
for (var i = 0; i < x.length; i++) {
   x[i].style.color = "blue";
}
</script>
```

Using for…of (Modern)

```
<script>
for (let p of document.querySelectorAll("p")) {
   p.style.fontSize = "20px";
}
</script>
```

Content Manipulation

1. innerHTML

The easiest way to get the content of an element is by using the inner HTML property. The inner HTML property is useful for getting or replacing the content of HTML elements. The inner HTML property can be used to get or change any HTML element, including <html> and <body>.

What is it?
Changes **HTML content** inside an element

Why we use it

- Allows **HTML tags**
- Used for dynamic content

Example

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "<b>Hello</b>";
</script>
```

Result : Text appears **bold**

2. innerText

What is it?
Changes **only text**, ignores HTML tags

Why we use it

- Prevents HTML rendering
- Safer for user input

Example

```
<p id="demo"></p>
<script>
document.getElementById("demo").innerText = "<b>Hello</b>";
</script>
```

Output
Hello

Attribute Manipulation

1. setAttribute()

What is it?
Adds or changes an attribute

When to use
- Changing src, href, class, etc.

Example

```
<img id="img1">
<script>
document.getElementById("img1").setAttribute("src", "image.jpg");
</script>
```

2. getAttribute()

What is it?
Reads an attribute value

Example

```
<p id="p1" title="Demo Text">Hello</p>

<script>
alert(document.getElementById("p1").getAttribute("title"));
</script>
```

3. removeAttribute()

What is it?
Removes an attribute completely

Example

```
<p id="p1" title="Demo">Hello</p>
<script>
document.getElementById("p1").removeAttribute("title");
</script>
```

Attribute Property: Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:

document.getElementById(*id*).***attribute*** = *new value*

Example

document.getElementById('p1').**title = "demo";**

Style Manipulation

What is it?
Changes CSS using JavaScript

Why we use it

- Dynamic styling
- Respond to events

Syntax:

element.**style.property** = 'value'

Example

```
<p id="p1">Hello</p>
<script>
document.getElementById("p1").style.backgroundColor = "yellow";
</script>
```

Note:
CSS properties written in kebab-case (such as background-color and font-size) must be converted to camelCase when used with JavaScript's style property, so background-color becomes backgroundColor and font-size becomes fontSize.

Example -1 : Write Js to change text, color and background color on click using "ClassName" method.

```
<head>
<script type="text/javascript">
  function cls()
  {
        cl = document.getElementsByClassName("same");
        for(i=0; i<cl.length;i++)
        {
          cl[i].innerHTML = "Changed text";
          cl[i].style.color = "blue";
          cl[i].style.backgroundColor = "limegreen";
        }
   }
</script></head>
<body>
        <h1 class="same">H1 tag</h1>
        <p class="same">P tag</p>
        <pre class="same">Pre tag</pre>
        <input type="button" onclick="cls();" value="CLICK"/>
</body>
```

Output:

(after clicking "CLICK" button)

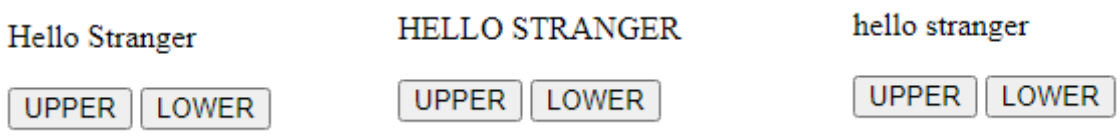
Example -2: Write JS code to Convert Uppercase and Lowercase from written text on click.

```
<body>
 <p>Hello Stranger</p>
 <input type="button" onclick="upr()" value="UPPER"/>
 <input type="button" onclick="lwr()" value="LOWER"/>
 <script>
        function upr() {
        let cl = document.querySelectorAll("p");
                for (let i = 0; i < cl.length; i++) {
                cl[i].style.textTransform = "uppercase";
```

```
  }
}

function lwr() {
  let cl = document.querySelectorAll("p");
  for (let i = 0; i < cl.length; i++) {
    cl[i].style.textTransform = "lowercase";
  }
}
</script>
</body>
```

Output:



For the understanding.

Below example finds the element with id="main", and then finds all <p> elements inside "main":

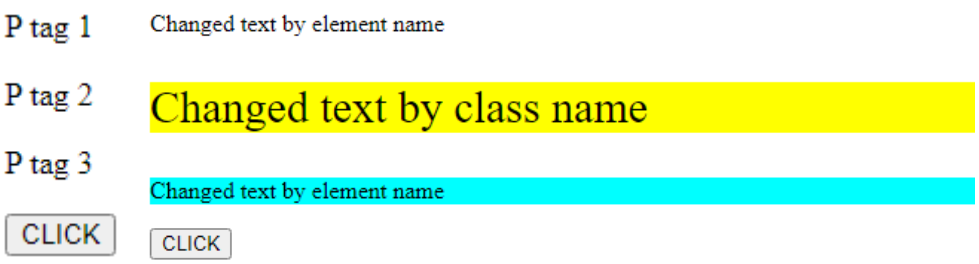
```
<div id="main">
   <p>Inside</p>
</div>
<p>outside</p>
   <script>
   const x = document.getElementById("main");
   const y = x.getElementsByTagName("p");
   y[0].style.backgroundColor='yellow';
   </script>
```

Output:

Inside

outside

Example -3: Write a JS to perform as shown in image.




```
<html><head>
<script type="text/javascript">
function demo()
   {
   p_element = document.getElementsByTagName("p");
   p_id = document.getElementById("p1");
   p_class= document.getElementsByClassName("pc")
   for(i=0; i<p_element.length;i++)
   {
      p_element[i].innerHTML = "Changed text by element name";
   }
   for(i=0; i<p_class.length;i++)
   {
      p_class[i].innerHTML = "Changed text by class name";
      p_class[i].style.backgroundColor = "yellow";
      p_class[i].style.fontSize = "30px";
   }
   p_id.style.backgroundColor = "cyan";
   }
</script>
</head>
<body>
<p>P tag 1</p>
<p class="pc">P tag 2</p>
<p id="p1">P tag 3</p>
<input type="button" onclick="demo();" value="CLICK"/>
</body></html>
```

Event Handling with JavaScript

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When JavaScript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

An **event** can be:

- Mouse click
- Key press
- Mouse movement
- Form submission
- Focus or blur of input field

JavaScript executes specific code when these events occur.

Event Handling Methods

1. Function Call Without Parameter

Example

```
<p onclick="fun()" id="test">Hello</p>

<script>
function fun() {
   document.getElementById("test").innerHTML = "Mouse clicked";
}
</script>
```

Explanation

- No argument is passed to the function.
- The function accesses the element using getElementById().
- Used when the element ID is fixed and known.

2. Function Call With this as Parameter

Example

```
<script>
function fun2(x) {
   x.innerHTML = "Mouse Click";
}
</script>
<p onclick="fun2(this)">test</p>
```

Explanation

- this refers to the element that triggered the event.
- The element is passed to the function as a parameter.
- This method is reusable and cleaner.

3. Inline JavaScript (No Function Call)

Example

```
<p onclick="this.style.color='red'; this.innerHTML='Mouse click';">Test</p>
```

Explanation

- JavaScript is written directly inside the HTML event.
- this refers to the current element.
- Used only for very small logic or demonstrations.

4. addEventListener()

addEventListener() is a JavaScript method used to **attach an event to an HTML element**.
It tells the browser:

```
“When this event happens, run this function.”
```

Why do we use addEventListener()?
We use addEventListener() because:

- It keeps **HTML and JavaScript separate**
- It is **cleaner and more readable**
- Multiple events can be added to the same element
- It is the **modern and recommended** way to handle events

Syntax

```
element.addEventListener("event", handler);
```

- event → event type (e.g. "click", "mouseover")
- handler → function reference used in addEventListener

Example: (function assignment to an event)

```
<button id="a"> Click Me</button>
<script>
const box = document.querySelector("#a");
const show = () => alert('Button Clicked');

box.addEventListener("click", show);
</script>
```

Example (Anonymous callback function)

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

How it works

- getElementById("btn") selects the button
- addEventListener("click", ...) attaches the click event
- When button is clicked → alert appears

Note: Remove **on** keyword when using addEventListener() from traditional HTML event attributes.

(*Ref) removeEventListener() is used to remove an event listener that was previously added to an element

Basic Syntax: element.removeEventListener(event, handler);

event → event type (e.g. "click", "mouseover")

handler → function reference used in addEventListener

Example

```
<button id="btn">Click me</button>
<script>
function sayHello() {
  console.log("Hello!");
}
const btn = document.getElementById("btn");
btn.addEventListener("click", sayHello);
// Remove the event listener
btn.removeEventListener("click", sayHello);
</script>
```

Mouse events

HTML Event Attribute	Event Name for addEventListener()	Description	When to Use	Simple Example
onclick	click	Fires when mouse click is completed	Button clicks	<button onclick="alert('Clicked')"> Click</button>
onmousedown	mousedown	Fires when mouse button is pressed	Detect press	Press
onmouseup	mouseup	Fires when mouse button is released	Detect release	Release
onmouseover	mouseover	Fires when mouse enters element	Hover effect	<h1 onmouseover ="this.style.color='red'">Hover</h1>
onmouseout	mouseout	Fires when mouse leaves element	Remove hover	<h1 onmouseout ="this.style.color='black'">Out</h1>

- **onclick**

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

Example

```
<body>
<h1 onclick="this.innerHTML = 'Text Changed!'">Click on this text!</h1>
<h2 onclick="alert(this.innerHTML)">Hello</h2>
</body>
```

- **onmousedown, onmouseup**

The **onmousedown, onmouseup** events are all parts of a mouse-click. First when a mouse-button is clicked, the onmousedown event is triggered, then, when the mouse-button is released, the onmouseup event is triggered,

finally, when the mouse-click is completed, the onclick event is triggered (discussed above).

Example

```
<div onmousedown="mdown(this)" onmouseup="mup(this)"
style="background-color:lightblue;width:70px;height:30px;padding:20px;">Click Me</div>
<script>
function mdown(e) {
  e.style.backgroundColor = "yellow";
  e.innerHTML = "click";
}
function mup(e) {
  e.style.backgroundColor="pink";
 e.innerHTML="Release";
}
</script>
```

- **onmouseover and onmouseout**

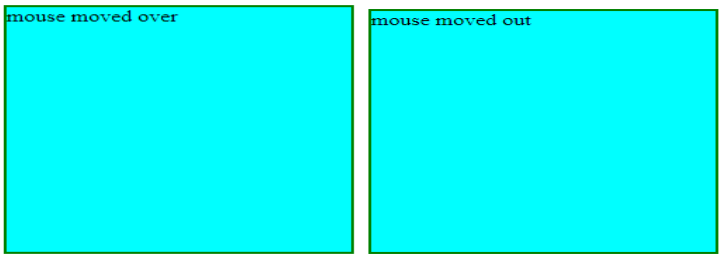
The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element:

Example: (onmouseout, onmouseover)

```
<div onmouseover="fun(this)" onmouseout="fun2(this)" style="width:50%; height:40%;
background-color:cyan; border:2px solid green;">

<script >
function fun(id){
   id.innerHTML = "mouse moved over";
}
function fun2(id){
   id.innerHTML = "mouse moved out";
}
</script>
</div>
```

Output:



Example : Write JS to handle following mouse events

1) If mouse is over heading should turn yellow, If mouse goes out then it should turn black.

2) If find time button is clicked then show date and time information.

3) If button named “red” is clicked then background color should turn red, and button named “green” is clicked then background color should turn green.

```
<body id="bd">
    <h1 onmouseover="fun(this)" onmouseout="fun2(this)">Hello</h1>
    <input type="submit" value="Find Time" onclick="fun3(this)"/>
    <p id="demo"></p>
    <input type="submit" value="red" onclick="fun4()"/>
    <input type="submit" value="green" onclick="fun5()"/>
<script type="text/javascript">
function fun(id){
     id.style.color = "yellow";
}
function fun2(id){
     id.style.color = "black";
}
function fun3(id){
     d = new Date();
     document.getElementById("demo").innerHTML = d;
}
function fun4(){
    id=document.getElementById("bd");
    id.bgColor = "red";
}
function fun5(){
     id=document.getElementById("bd");
     id.bgColor = "green";
}
</script>
</body>
```

Keyboard events

HTML Event Attribute	Event Name for addEventListener()	Description
onkeyup	keyup	when the user releases a key on the keyboard
onkeydown	keydown	when the user presses a key on the keyboard. (for ctrl, shift, function etc. keys)
onkeypress	Keypress (deprecated)	when the user presses a key on the keyboard (for letters, numbers, symbols, when some output is there on screen)

Example:

Write Html/js to perform the task as asked below.
Background color should turn red while key is down and it should turn blue while key is up.

```
<head>
<script type="text/javascript">
function fun1(id)
{
id.bgColor="blue";
}
function fun2(id)
{
id.bgColor="red";
}
</script>
</head>
<body onkeyup="fun1(this)"  onkeydown="fun2(this)">
</body>
```

Form events

HTML Event Attribute	Event Name for addEventListener()	Description
onfocus	focus	When the user focuses on an element (clicks in input field)
oninput	input	Immediately when value changes
onchange	change	When the user modifies or changes the value of a form element.Also works on <select>
onblur	blur	When the focus is away from a form element (clicks out of input field after clicking in input field)
onsubmit	submit	When the user submits the form

- **onfocus**

The onfocus event occurs when an element gets focus.

The onfocus event is often used on input fields.

```
<element onfocus="myScript">
```

- **oninput**

The oninput event occurs **immediately when the value of an input element changes** (while typing).

When to Use

- Live validation
- Auto formatting
- Real-time response

Syntax

```
<element oninput="myScript">
```

Example

```
<input type="text" oninput="this.value=this.value.toUpperCase()">
```

- **onchange**

The onchange event occurs when **the value of an HTML element is changed**.

***<element* onchange="*myScript*">**

Tip: This event is similar to the oninput event. The difference is that the oninput event occurs immediately after the value of an element has changed, while onchange occurs when the element loses focus, after the content has been changed. The other difference is that the onchange event also works on <select> elements.

- **onblur**

The onblur event occurs when an **HTML element loses focus**.

```
<element onblur="myScript">
```

The onblur event is often used on input fields.
The onblur event is often used with form validation (when the user leaves a form field).

Example

```
<input type="text" onblur="this.value=this.value.toLowerCase()">
```

Remember:
The onBlur event is fired when you have moved away from an object without necessarily having changed its value.
The onChange event is only called when you have changed the value of the field and it loses focus.

- **onsubmit**

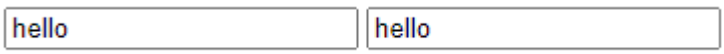
The onsubmit event occurs when a form is submitted.

```
<form action="#" onsubmit="myFunction()">
  Enter name: <input type="text" name="fname" id="fname">
  <input type="submit" value="Submit">
</form>
<script>
function myFunction() {
   a =  document.getElementById("fname").value;
  alert("The form was submitted! Welcome " +  a);
}
</script>
```

Example: (onchange, onfocus)

```
   <input type="text" value="hello" onchange="upper(this)"/>
    <input type="text" value="hello" onfocus="this.style.backgroundColor='yellow'"/>
<script type="text/javascript">
    function upper(id){
        id.value=id.value.toUpperCase();}
</script>
```

Output:



(after change and focus)

Example:

Write an HTML and JavaScript program to perform the following tasks:

1. Add **three text input fields**.
2. In the **first text field**, convert the entered text into **uppercase** when the **value of the field is changed** using the onchange event.
3. In the **second text field**:
   - Change the **background color to yellow** when the field **gets focus** using the onfocus event.
   - Change the **text color to blue while typing** using the oninput event.
4. In the **third text field**, convert the entered text into **lowercase** when the field **loses focus** using the onblur event.

```
<script type="text/javascript">
function uppercase()
{
   var a = document.getElementById("fname");
   a.value = a.value.toUpperCase();
}

function lower(a)
{
   a.value = a.value.toLowerCase();
}
function inputText(x)
{
   x.style.color = "blue";
}
</script>
<!-- onchange -->
<input type="text" value="hello" id="fname" onchange="uppercase()">
<!-- onfocus + oninput -->
<input type="text" value="hello" onfocus="this.style.backgroundColor='yellow'"
oninput="inputText(this)">
<!-- onblur -->
<input type="text" value="HELLO" onblur="lower(this)">
```

Methods to access values of Form elements

Using document.forms[“”][“”].value

obj = document.forms["form_name"]["element_name"].value;

- Accesses the form using its **name**
- Then accesses the input using its **name**
- Clear and safe when working with multiple forms
- Multiple forms exist on the page

Using form and element names directly(. Method)

obj = document.form_name.element_name.value;

- Shorter syntax
- Works only if both **form name** and **element name** exist
- When form and elements have **unique name attributes**
- When form and elements have **unique name attributes**
- Considered **old-style** and less recommended today

Using getElementById (Best method)

obj = document.getElementById("id").value;

- Accesses the element using its **id**
- **Most modern and recommended approach**
- Independent of form names

<form name= “form_name” action= “#”>
<input type= “text” name= “element_name”…..>
</form>

Can access form values by following methods

obj = document.forms[“form_name”][“element_name”].value;

or

obj = document.form_name.element_name.value;

or

obj = document.getElementByID(id).value;

Note: document.getElementById().value is the most reliable and recommended approach in modern JavaScript.

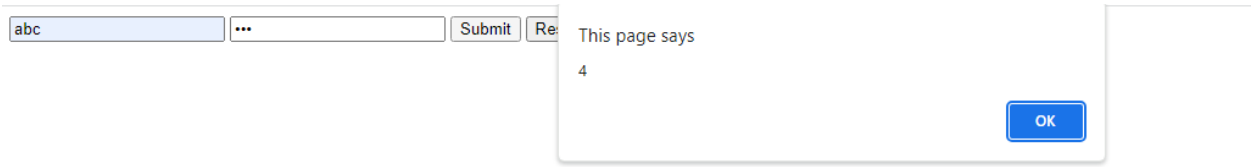
Example : Display User's Entry on Alert using Event (onsubmit)

```
<body>
<form name="f1" onsubmit="return fun()">
   <input type="text" name="t1" id="i1" />
   <input type="submit"/>
</form>
<script>
function fun(){
    // METHOD1
    var obj = document.f1.t1.value;
    // METHOD2
   // var obj = document.forms["f1"]["t1"].value;
   // METHOD3
   // var obj = document.getElementById("i1").value;
   alert(obj);
}
</script>
</body>
```

Example: write a code to find length of form elements.

```
<body>
<form name="f1" onsubmit="fun()">
   <input type="text" name="t1"/>
   <input type="password"/>
   <input type="submit"/>
   <input type="reset"/>
</form>
<script>
function fun()
{
   obj = document.forms["f1"];
   alert(obj.length);
}
</script>
</body>
```

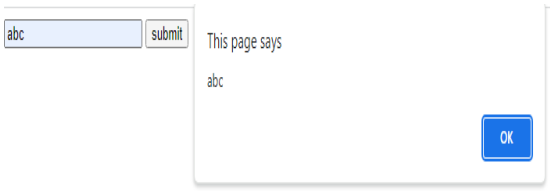
Output: (on clicking submit button.)

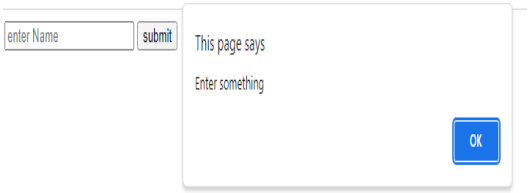


Example: Write a code that shows alert which pop up with text **"**Enter Something" if input field is empty on clicking submit button.

```
<html>
<body>
<form action="#" onsubmit="fun(this)">
   <input type="text" maxlength="10" placeholder="enter Name" id="t1"/>
   <input type="submit" value="submit"/>
</form>
<script type="text/javascript">
function fun(demo)
{
     val=demo.t1.value;
  // OR   val=document.getElementById("t1").value;
     if(val=="")
     {
         alert("Enter something");
         return false;
    }
     alert(val);
}
</script>
</body></html>
```

Output:





Event Object & Handling Essentials

Property	Description	Used For
event.target	Element that triggered event	Identify element
event.type	Type of event	Debugging
event.key	Key pressed	Keyboard input
event.keyCode	Keycode of key	Keyboard input
event.button	Value of Button click	Mouse click
event.preventDefault()	Stops default behavior	Form control

1. event.target : **event.target refers to the** HTML element that triggered the event**.**

Used for

- Identifying which element was clicked
- Handling multiple elements using one function

Example

```
<select onchange="show(event)">
  <option value="Red">Red</option>
  <option value="Blue">Blue</option>
</select>
<script>
function show(e) {
   alert(e.target.value);
//alert(e.target.tagName)
}
</script>
```

2. event.type : **event.type** tells which event occurred**.**

Used for

- Debugging
- Handling multiple events using one function

Example

```
<button onclick="check(event)">Click</button>
<script>
function check(e) {
   alert(e.type);
}</script>
```

Explanation

- Clicking the button triggers a click event
- Output shows click

3. event.key : **event.key** returns the key pressed by the user.

Used for

- Keyboard input
- Validation
- Games and shortcuts
- **Return Value: A String**

Example

```
<body onkeypress="keyCheck(event)">
<script>
function keyCheck(e) {
   alert("Key Pressed: " + e.key);
}
</script>
</body>
```

Explanation

- User presses a key
- Returns The key that was pressed:
  - A single character ("A", "a", "4", "+", "$")
  - Multiple characters ("F1", "Enter", "HOME", "CAPS LOCK")

4. event.keyCode : Get the value of the pressed keyboard key

The keyCode property is deprecated.
Syntax : event.keyCode

Example: Write script to display Unicode on key press.

```
<script >
function fun(e)
{ alert(e.keyCode); }
</script>
<body onkeypress="fun(event)" ></body>
```

Example: Write JS to handle following key events

1) Give keycode for the key pressed

2) Script should give message “vowel is pressed” on pressing vowel key

```
<body onkeypress="fun2(event)"></body></html>

<script type="text/javascript">
function fun2(e){
```

```
c= (e.key);
 if(c=='a' || c=='e' || c=='i' || c=='o' || c=='u' || c=='A' || c=='E' || c=='I' ||
c=='O' || c=='U') {
 alert("vowel is entered "+e.keyCode+" "+ e.key); }
 else { alert(e.keyCode+" "+ e.key); }
}
</script>
```

4. event.button: Used to get the **mouse button** that was pressed.

- The button property returns which mouse button is pressed when a mouse event occurs.
- The button property is mostly used with the onmousedown event.
- The button property is read-only.

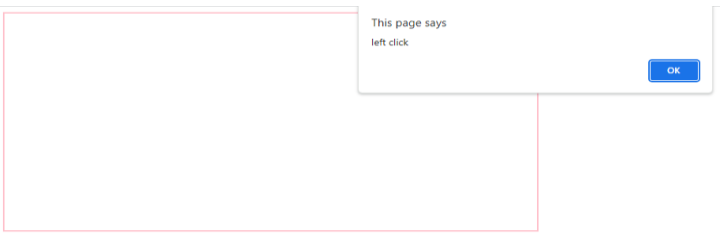
Syntax: event.button
Return Value: A Number.

Which mouse button that was pressed:
0 : Left button
1 : Wheel or middle button (if present)
2 : Right button
For a left-hand configured mouse, the values are reversed.

Example: Write a Js to check which mouse button is clicked (right, left or middle)

```
<body>
  <div style="width:50%;height:50%; border:2px solid pink;"
onmousedown="fun(event)"></div>
</body>
<script type="text/javascript">
function fun(e){
    val = e.button;
if (val === 0) { alert("left click"); }
else if (val === 1) { alert("middle click"); }
else{ alert("right click"); }
} </script>
```

Output:



6. event.preventDefault() : Stops the default browser action.

e.preventDefault() prevents the default action of an event from occurring.

Used for

- Preventing form submission
- Stopping page reload
- Custom form validation
-

Example :

```
<form onsubmit="stop(event)" action='a.png'>
<input type="submit" value="Submit">
</form>
<script>
function stop(e) {
e.preventDefault(); // Will prevent form to redirect  so image will not load.
alert("Form submission stopped");
}</script>
```

Example: Prevent form to not accept letters

```
<body>
<input type="text" id="num" placeholder="Enter numbers">
<script>
const input = document.querySelector("#num");

input.addEventListener("keydown", e => {
  if (isNaN(e.key)) {
    e.preventDefault();
  }
});
</script>
</body>
```