JavaScript – Part 2: DOM

Presented By

Anuradha Kumari Singh
(Research Scholar)
Department of Computer Science

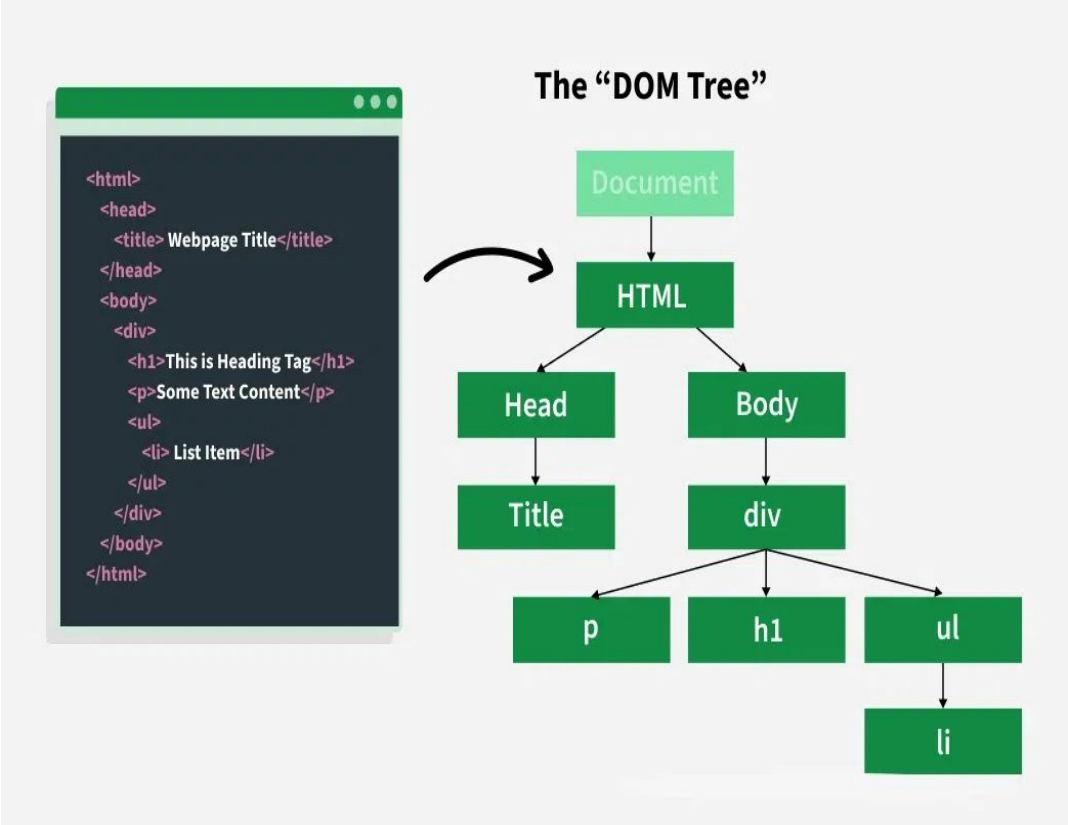
Topics

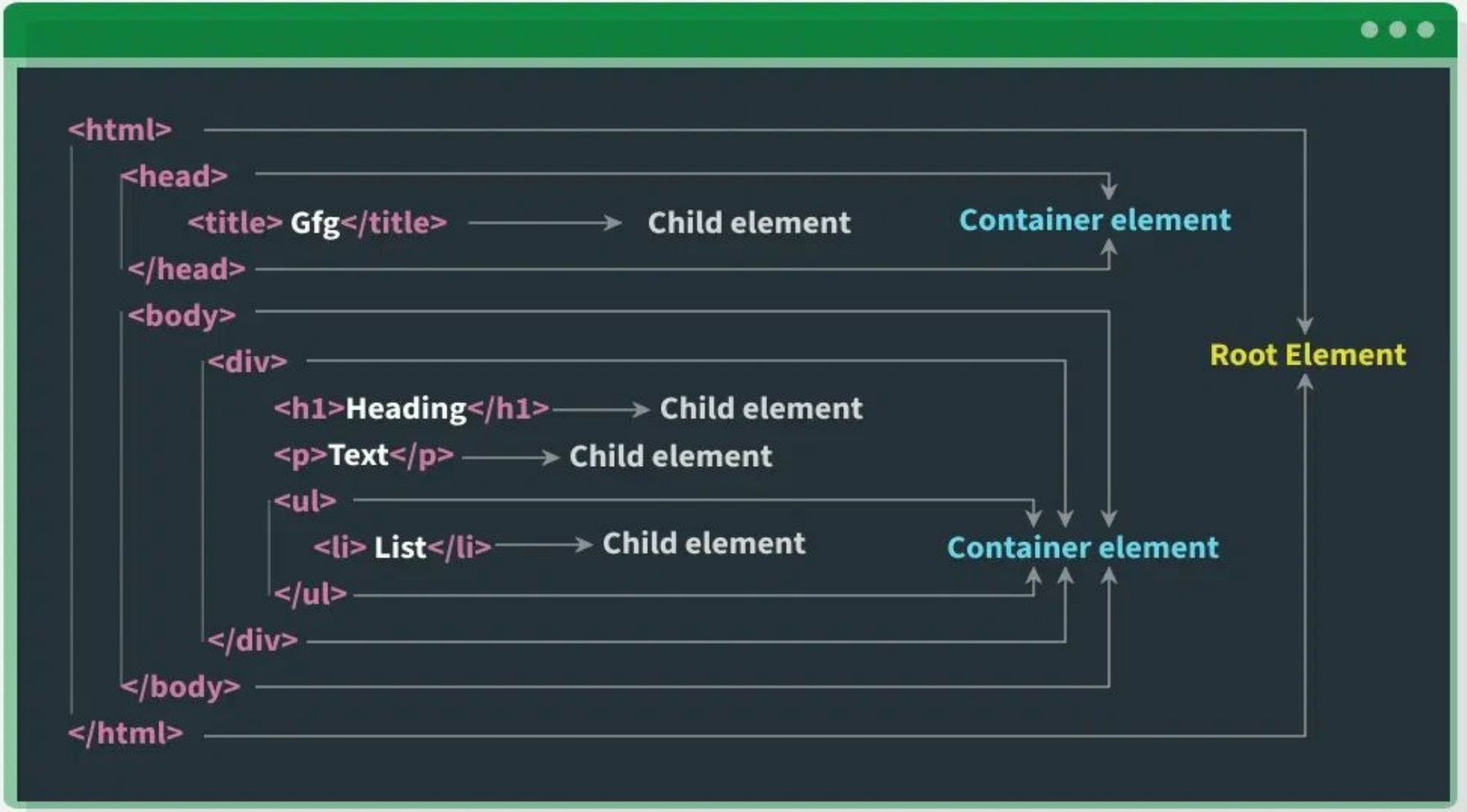
- **HTML DOM (Document Object Model)**
- **How to select DOM Elements in JavaScript ?**
- **Events**
- **JavaScript addEventListener() with Examples**
- **Popup box**
- **Cookies**
- **Form Validation**

HTML DOM (Document Object Model)

HTML DOM (Document Object Model)

- The HTML DOM (Document Object Model) is a structured representation of a web page that allows developers to access, modify, and control its content and structure using JavaScript.
- It powers most dynamic website interactions, enabling features like real-time updates, form validation, and interactive user interfaces.




<html>
<head>
<title> Gfg</title>
Child element
Container element
</head>
<body>
Root Element

<h1>Heading</h1>
Child element
Text
Child element

 List
Child element
Container element

</body>
</html>

- The HTML Document Object Model (DOM) is a tree structure, where each HTML tag becomes a node in the hierarchy.
- At the top, the <html> tag is the root element, containing both <head> and <body> as child elements.
- These in turn have their own children, such as <title>, , and nested elements like <h1>, <p>, <ul>, and <li>.
- Elements that contain other elements are labeled as container elements, while elements that do not are simply child elements.
- This hierarchy allows developers to navigate and manipulate web page content using JavaScript by traversing from parent to child and vice versa.

What Does the HTML DOM Look Like?

- The document is the root.
- HTML tags like <html>, <head>, and <body> are branches.
- Attributes, text, and other elements are the leaves.

Why is DOM Required?

The DOM is essential because:

- **Dynamic Content Updates:** Without reloading the page, the DOM allows content updates (e.g., form validation, AJAX responses).
- **User Interaction:** It makes your webpage interactive (e.g., responding to button clicks, form submissions).
- **Flexibility:** Developers can add, modify, or remove elements and styles in real-time.
- **Cross-Platform Compatibility:** It provides a standard way for scripts to interact with web documents, ensuring browser compatibility.

How the DOM Works?

The DOM connects your webpage to JavaScript, allowing you to:

- Access elements (like finding an <h1> tag).
- Modify content (like changing the text of a <p> tag).
- React to events (like a button click).
- Create or remove elements dynamically.

How to select DOM Elements in JavaScript?

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and with other programming languages).

In the DOM, all HTML elements are defined as objects.

The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Methods to select DOM elements in JavaScript:

1. Using getElementById
2. Using getElementByClassName
3. Using getElementByTagName
4. Using querySelector
5. Using querySelectorAll

1. Using getElementById

This method selects a single element by its unique ID attribute.

Syntax:

```
document.getElementById('id')
```

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>getElementById Example</title>
</head>

<body>
    <h1 id="gfg">GeeksForGeeks</h1>

    <script>
        // Access the h1 tag
        const element = document.getElementById('gfg');

        // Set text color
        element.style.color = "green";

        // Set text alignment
        element.style.textAlign = "center";

        // Optional styling
        element.style.margin = "30px";
        element.style.fontSize = "30px";
    </script>
</body>

</html>
```

2. Using getElementsByClassName

This method selects elements based on their class attribute. It returns a collection of elements with the specified class name.

Syntax:

```
document.getElementsByClassName('class')
```

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, initial-scale=1.0">
    <title>getElementsByClassName Example</title>
</head>

<body>
    <h1 class="selector">GeeksForGeeks</h1>
    <h2 class="selector">DOM selector in JavaScript</h2>

    <script>
        // Corrected: document (not documSizeent)
        const elements = document.getElementsByClassName('selector');

        elements[0].style.color = "green";
        elements[1].style.color = "red";

        elements[0].style.textAlign = "center";
        elements[1].style.textAlign = "center";

        elements[0].style.marginTop = "60px";
    </script>
</body>

</html>
```

3. Using getElementsByTagName

This method selects elements based on their tag name. It returns a collection of elements with the specified tag name.

Syntax:

```
document.getElementsByTagName('tag')
```

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

<head>
  <meta charset="UTF-8">
  <meta name="viewport"
      content="width=device-width, initial-scale=1.0">
  <title>getElementsByTagName Example</title>
</head>

<body>
  <p>Thanks for visiting GFG</p>
  <p>This is showing how to select DOM element using tag name</p>

  <script>
    const paragraphs = document.getElementsByTagName('p');
    paragraphs[0].style.color = "green";
    paragraphs[1].style.color = "blue";
    paragraphs[0].style.fontSize = "25px";
    paragraphs[0].style.textAlign = "center";
    paragraphs[1].style.textAlign = "center";
    paragraphs[0].style.marginTop = "60px";
  </script>
</body>

</html>
```

4. Using querySelector

This method selects the first element that matches a specified CSS selector. It returns only one element.

Syntax:

```
document.querySelector('selector')
```

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

<head>
   <meta charset="UTF-8">
   <meta name="viewport"
        content="width=device-width,
                initial-scale=1.0">
   <title>querySelector Example</title>
</head>

<body>
   <div class="gfg">GeeksForGeeks</div>

   <script>
      const element =
      document.querySelector('.gfg');
      element.style.color = "green";
      element.style.textAlign = "center";
      element.style.margin = "30px";
      element.style.fontSize = "30px";
   </script>
</body>

</html>
```

5. Using querySelectorAll

Similar to querySelector, but it returns a NodeList containing all elements that match the specified CSS selector.

Syntax:

```
document.querySelectorAll('selector')
```

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, initial-scale=1.0">
    <title>querySelectorAll Example</title>
</head>

<body>
    <h1 class="selector">GeeksForGeeks</h1>
    <p class="selector">
        This is showing how to select DOM element
        using tag name
    </p>

    <script>
        const elements = document.querySelectorAll('.selector');
        elements[0].style.color = "green";
        elements[1].style.color = "red";
        elements[0].style.textAlign = "center";
        elements[1].style.textAlign = "center";
        elements[0].style.marginTop = "60px";
    </script>
</body>
</html>
```

innerHTML property

The innerHTML property can be used to get or change any HTML element, including <html> and <body>.

Example:

```
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>

</body>
</html>
```

Changing the HTML content

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

<h1 id="id01">Old Heading</h1>

<script>
const element = document.getElementById("id01");
element.innerHTML = "New Heading";
</script>

<p>JavaScript changed "Old Heading" to "New Heading".</p>

</body>
</html>
```

- The HTML document above contains an <h1> element with id="id01"
- We use the HTML DOM to get the element with id="id01"
- A JavaScript changes the content (innerHTML) of that element to "New Heading"

Changing the Value of an Attribute

syntax:

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

Example:

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

<img id="myImage" src="smiley.gif">

<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>

</body>
</html>
```

- The HTML document above contains an <img> element with id="myImage"
- We use the HTML DOM to get the element with id="myImage"
- A JavaScript changes the src attribute of that element from "smiley.gif" to "landscape.jpg"

Dynamic HTML content

Date():

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

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Date : " + Date();
</script>

</body>
</html>
```

HTML Event

All event objects in the HTML DOM are based on the Event Object.

All event objects (like MouseEvent and KeyboardEvent) has access to the Event Object's properties and methods.

Event	Occurs When
abort	The loading of a media is aborted
afterprint	A page has started printing
beforeprint	A page is about to be printed
beforeunload	Before the document is about to be unloaded
canplay	The browser can start playing a media (has buffered enough to begin)
canplaythrough	The browser can play through a media without stopping for buffering
change	The content of a form element have changed
ended	A media has reach the end ("thanks for listening")
error	An error has occurred while loading a file
fullscreenchange	An element is displayed in fullscreen mode

fullscreenerror	An element can not be displayed in fullscreen mode
input	An element gets user input
invalid	An element is invalid
load	An object has loaded
loadeddata	Media data is loaded
loadedmetadata	Meta data (like dimensions and duration) are loaded
message	A message is received through the event source
offline	The browser starts working offline
online	the browser starts working online
open	A connection with the event source is opened
pause	A media is paused
scroll	A scrollbar is being scrolled
search	Something is written in a search field

play	A media has started or is no longer paused
playing	A media is playing after beeing paused or buffered
progress	The browser is downloading media data
ratechange	The playing speed of a media is changed
resize	The document view is resized
reset	A form is reset
play	A media has started or is no longer paused

1. The FocusEvent Object

The FocusEvent Object handles events that occur when elements gets or loses focus.

Event	Occurs When
onblur	An element loses focus
onfocus	An element gets focus
onfocusin	An element is about to get focus
onfocusout	An element is about to lose focus

Property	Returns
relatedTarget	The element that triggered the event

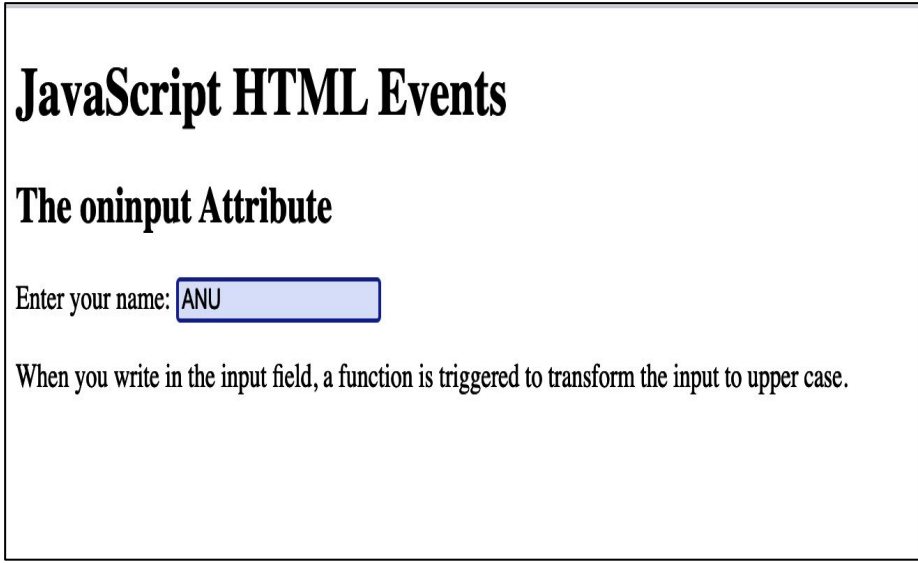
2. The InputEvent Object

The InputEvent Object handles events that occur when an input element is changed.

Event	Occurs When
oninput	The content (value) of an input element is changed

Property	Returns
data	The inserted characters
dataTransfer	An object containing information about the inserted/deleted data
inputType	The type of the change (i.e "inserting" or "deleting")
isComposing	If the state of the event is composing or not

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The oninput Attribute</h2>

Enter your name: <input type="text" id="fname" oninput="upperCase()">
<p>When you write in the input field, a function is triggered to transform the input to upper case.</p>

<script>
function upperCase() {
  const x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>

</body>
</html>
```



3. The KeyboardEvent Object

The KeyboardEvent Object handles events that occur when a user presses a key on the keyboard.

Event	Occurs When
onkeydown	A user presses a key
onkeypress	A user presses a key
onkeyup	A user releases a key

4. The MouseEvent Object

The MouseEvent Object handles events that occur when the mouse interacts with the HTML document.

Event	Occurs When
onclick	A user clicks on an element
oncontextmenu	A user right-clicks on an element
ondblclick	A user double-clicks on an element
onmousedown	A mouse button is pressed over an element
onmouseenter	The mouse pointer moves into an element
onmouseleave	The mouse pointer moves out of an element
onmousemove	The mouse pointer moves over an element
onmouseout	The mouse pointer moves out of an element
onmouseup	A mouse button is released over an element

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript HTML Events</h1>
<h2>The onclick Events</h2>

<p>Click "Try it" to execute the displayDate() function.</p>
<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
document.getElementById("myBtn").onclick = displayDate;

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
</body>
</html>
```

DOM EventListener

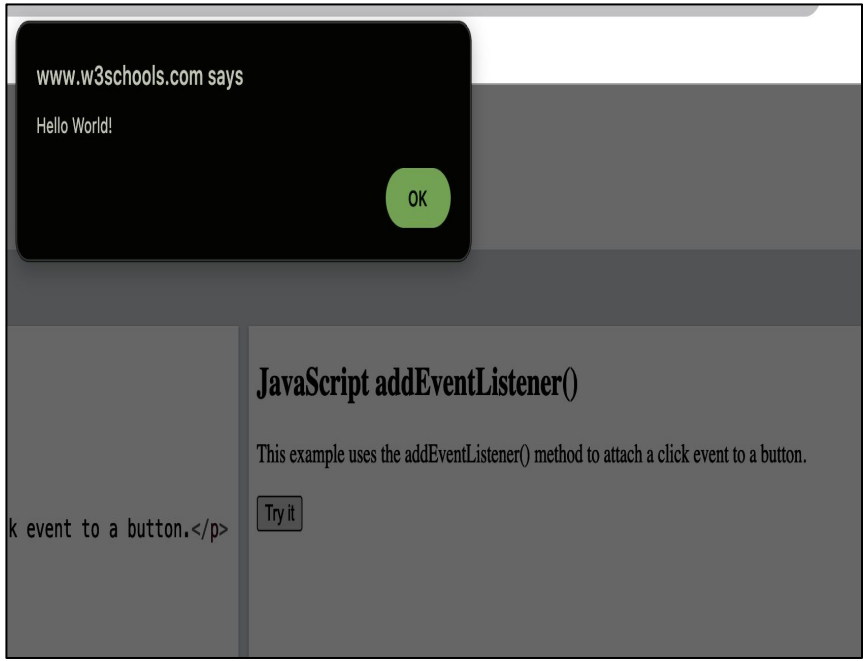
addEventListener(): method attaches an event handler to the specified element.

Syntax:

element.addEventListener(*event, function, useCapture*);

- The first parameter is the type of the event (like "click" or "mousedown" or any other HTML DOM Event.)
- The second parameter is the function we want to call when the event occurs.
- The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

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

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to attach a click event to a button.</p>

<button id="myBtn">Try it</button>

<script>
document.getElementById("myBtn").addEventListener("click", function() {
  alert("Hello World!");
});
</script>

</body>
</html>
```



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

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add two click events to the same button.</p>

<button id="myBtn">Try it</button>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);

function myFunction() {
  alert ("Hello World!");
}

function someOtherFunction() {
  alert ("This function was also executed!");
}
</script>

</body>
</html>
```

The removeEventListener() method

The removeEventListener() method removes event handlers that have been attached with the addEventListener() method.

Syntax:

element.removeEventListener("event", myFunction);

```
<!DOCTYPE html>
<html>
<head>
<style>
#myDIV {
  background-color: coral;
  border: 1px solid;
  padding: 50px;
  color: white;
  font-size: 20px;
}
</style>
</head>
<body>

<h2>JavaScript removeEventListener()</h2>

<div id="myDIV">
  <p>This div element has an onmousemove event handler that displays a random number every time you move your mouse inside this orange field.</p>
  <p>Click the button to remove the div's event handler.</p>
  <button onclick="removeHandler()" id="myBtn">Remove</button>
</div>

<p id="demo"></p>
```

```
<script>
document.getElementById("myDIV").addEventListener("mousemove", myFunction);

function myFunction() {
  document.getElementById("demo").innerHTML = Math.random();
}

function removeHandler() {

document.getElementById("myDIV").removeEventListener("mousemove", myFunction);
}
</script>

</body>
</html>
```

JavaScript Popup Boxes

JavaScript has three kinds of popup boxes:

- Alert box
- Confirm box
- Prompt box

1. Alert Box

An alert box is often used if you want to make sure information comes through to the user.

Syntax

window.alert("*sometext*");

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

<h2>JavaScript Alert</h2>

<button onclick="myFunction()">Try
it</button>

<script>
function myFunction() {
  alert("I am an alert box!");
}
</script>

</body>
</html>
```

2. Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

window.confirm("*sometext*");

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

<h2>JavaScript Confirm Box</h2>


<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var txt;
  if (confirm("Press a button!")) {
    txt = "You pressed OK!";
  } else {
    txt = "You pressed Cancel!";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```

3. Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

window.prompt("*sometext*","*defaultText*");

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

<h2>JavaScript Prompt</h2>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  let text;
  let person = prompt("Please enter your name:",
"Harry Potter");
  if (person == null || person == "") {
    text = "User cancelled the prompt.";
  } else {
    text = "Hello " + person + "! How are you today?";
  }
  document.getElementById("demo").innerHTML =
text;
}
</script>

</body>
</html>
```

What are Cookies?

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

- When a user visits a web page, his/her name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his/her name.

Cookies are saved in name-value pairs like: **username = John Doe**

When a browser requests a web page from a server, cookies belonging to the page are added to the request. This way the server gets the necessary data to "remember" information about users.

code

Create cookies

Delete cookies

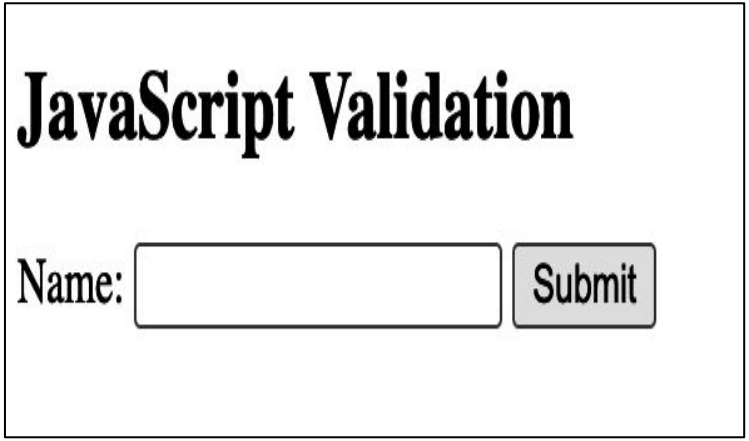
Form Validation

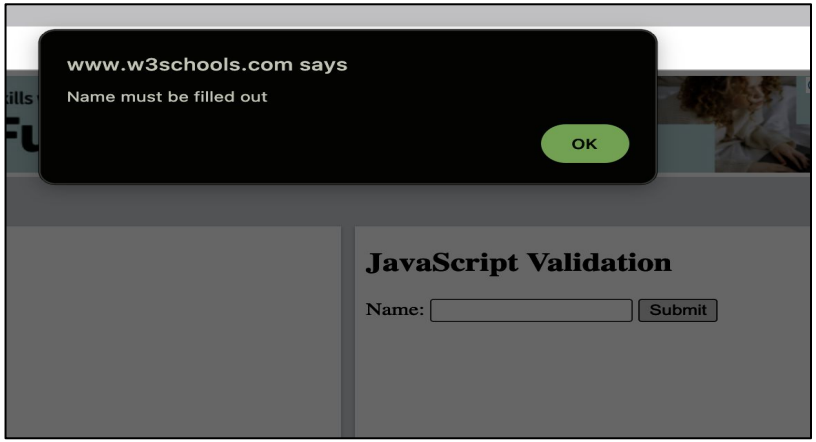
HTML form validation can be done by JavaScript.

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

```
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
      alert("Name must be filled out");
    return false;
  }
}
```

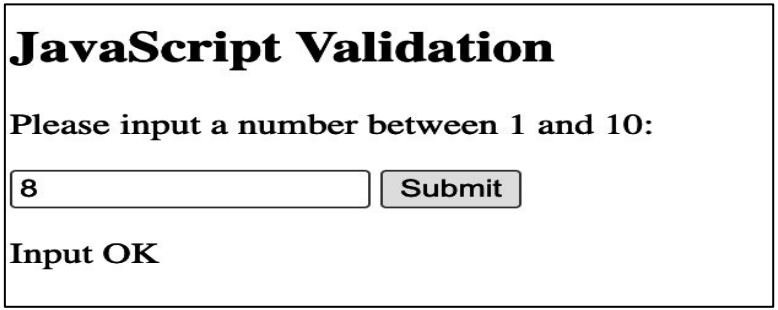
```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
  let x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
<body>
<h2>JavaScript Validation</h2>

<form name="myForm" action="" onsubmit="return
validateForm()" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

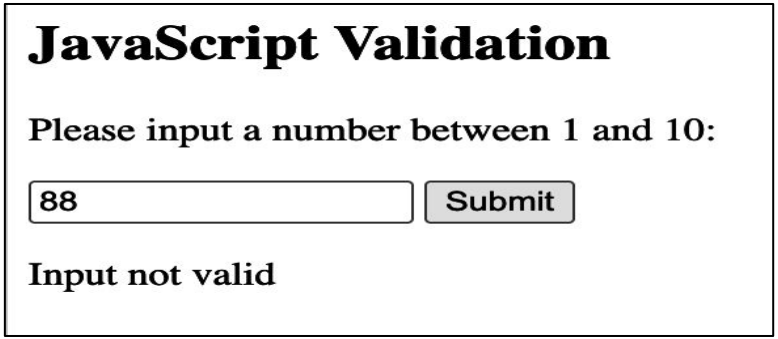




Validate Numeric Input

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

<h2>JavaScript Validation</h2>

<p>Please input a number between 1 and 10:</p>

<input id="numb">

<button type="button" onclick="myFunction()">Submit</button>

<p id="demo"></p>
<script>
function myFunction() {
  // Get the value of the input field with id="numb"
  let x = document.getElementById("numb").value;
  // If x is Not a Number or less than one or greater than 10
  let text;
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input not valid";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

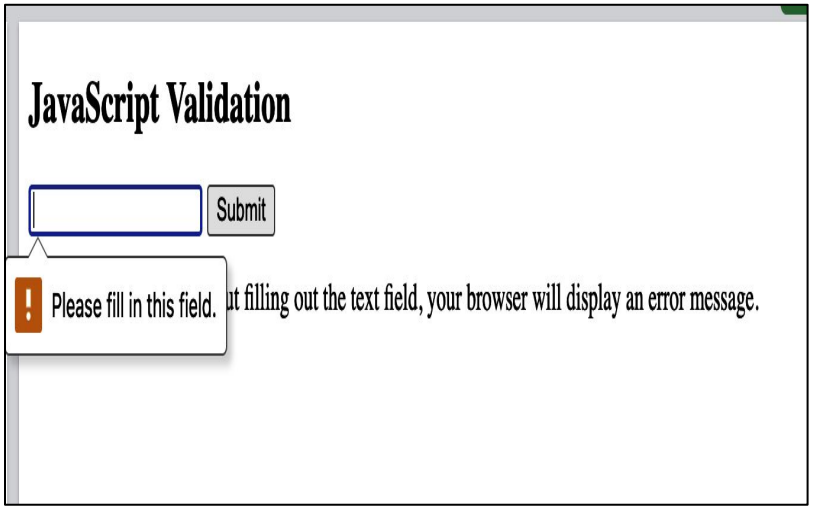




Constraint Validation HTML Input Attributes

Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element
type	Specifies the type of an input element

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

<h2>JavaScript Validation</h2>

<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>

<p>If you click submit, without filling out the text field,
your browser will display an error message.</p>

</body>
</html>
```



References

- Powell, T. A. ,HTML & CSS: The Complete Reference.
- Goodman, D., Morrison, M., Novitski, P., Rayl, T. G., JavaScript Bible.
- https://www.w3schools.com/js/js_htmldom.asp.