3/22/2022

Nagendra Mirajkar

Zensar AES Practices

JavaScript DOM

Document Object Model

Table of Contents

[Document Object Model in JavaScript 10](#_Toc98888128)

[What is Document Object Model (DOM) 10](#_Toc98888129)

[A document as a hierarchy of nodes 10](#_Toc98888130)

[**Node Types** 11](#_Toc98888131)

[**The nodeName and nodeValue properties** 11](#_Toc98888132)

[**Node and Element** 12](#_Toc98888133)

[**Node Relationships** 13](#_Toc98888134)

[Summary 13](#_Toc98888135)

[JavaScript getElementById 13](#_Toc98888136)

[Introduction to JavaScript getElementById() method 14](#_Toc98888137)

[JavaScript getElementById() method example 14](#_Toc98888138)

[Summary 15](#_Toc98888139)

[JavaScript getElementsByName 15](#_Toc98888140)

[Introduction to JavaScript getElementsByName() method 15](#_Toc98888141)

[JavaScript getElementsByName() example 15](#_Toc98888142)

[Summary 17](#_Toc98888143)

[JavaScript getElementsByName 17](#_Toc98888144)

[Introduction to JavaScript getElementsByName() method 17](#_Toc98888145)

[JavaScript getElementsByName() example 17](#_Toc98888146)

[Summary 18](#_Toc98888147)

[JavaScript getElementsByTagName 18](#_Toc98888148)

[Introduction to JavaScript getElementsByTagName() method 19](#_Toc98888149)

[JavaScript getElementsByTagName() example 19](#_Toc98888150)

[Summary 20](#_Toc98888151)

[JavaScript getElementsByClassName 20](#_Toc98888152)

[Introduction to the getElementsByClassName() method 20](#_Toc98888153)

[JavaScript getElementsByClassName() method examples 20](#_Toc98888154)

[**1) Calling JavaScript getElementsByClassName() on an element example** 21](#_Toc98888155)

[**2) Calling JavaScript getElementsByClassName() on the document example** 22](#_Toc98888156)

[Summary 22](#_Toc98888157)

[JavaScript querySelector 22](#_Toc98888158)

[Introduction to JavaScript querySelector() and querySelectorAll() methods 22](#_Toc98888159)

[Basic selectors 23](#_Toc98888160)

[**1) Universal selector** 24](#_Toc98888161)

[**2) Type selector** 24](#_Toc98888162)

[**3) Class selector** 24](#_Toc98888163)

[**4) ID Selector** 25](#_Toc98888164)

[**5) Attribute selector** 25](#_Toc98888165)

[Grouping selectors 25](#_Toc98888166)

[Combinators 26](#_Toc98888167)

[**1) descendant combinator** 26](#_Toc98888168)

[**2) Child combinator** 26](#_Toc98888169)

[**3) General sibling combinator** 26](#_Toc98888170)

[**4) Adjacent sibling combinator** 26](#_Toc98888171)

[Pseudo 27](#_Toc98888172)

[**1) Pseudo-classes** 27](#_Toc98888173)

[**2) Pseudo-elements** 27](#_Toc98888174)

[Summary 27](#_Toc98888175)

[JavaScript Get the Parent Element parentNode 27](#_Toc98888176)

[Introduction to parentNode attribute 27](#_Toc98888177)

[JavaScript parentNode example 28](#_Toc98888178)

[Summary 28](#_Toc98888179)

[Getting Child Elements of a Node in JavaScript 28](#_Toc98888180)

[Get the first child element 29](#_Toc98888181)

[Get the last child element 30](#_Toc98888182)

[Get all child elements 30](#_Toc98888183)

[Summary 31](#_Toc98888184)

[JavasScript Siblings 31](#_Toc98888185)

[Get the next siblings 31](#_Toc98888186)

[Get the previous siblings 32](#_Toc98888187)

[Get all siblings of an element 33](#_Toc98888188)

[Summary 34](#_Toc98888189)

[JavaScript CreateElement 34](#_Toc98888190)

[1) Creating a new div example 35](#_Toc98888191)

[**Adding an id to the div** 35](#_Toc98888192)

[**Adding a class to the div** 36](#_Toc98888193)

[**Adding text to a div** 36](#_Toc98888194)

[**Adding an element to a div** 36](#_Toc98888195)

[2) Creating new list items (li) example 36](#_Toc98888196)

[3) Creating a script element example 37](#_Toc98888197)

[Summary 38](#_Toc98888198)

[JavaScript appendChild 38](#_Toc98888199)

[Introduction to the JavaScript appendChild() method 38](#_Toc98888200)

[JavaScript appendChild() examples 38](#_Toc98888201)

[**1) Simple appendChild() example** 38](#_Toc98888202)

[**2) Moving a node within the document example** 40](#_Toc98888203)

[Summary 41](#_Toc98888204)

[JavaScript textContent 41](#_Toc98888205)

[Reading textContent from a node 41](#_Toc98888206)

[**textContent vs. innerText** 42](#_Toc98888207)

[Setting textContent for a node 42](#_Toc98888208)

[Summary 42](#_Toc98888209)

[JavaScript innerHTML 43](#_Toc98888210)

[Reading the innerHTML property of an element 43](#_Toc98888211)

[**1) Simple innerHTML example** 43](#_Toc98888212)

[**2) Examining the current HTML source** 43](#_Toc98888213)

[Setting the innerHTML property of an element 44](#_Toc98888214)

[**⚠️ Security Risk** 45](#_Toc98888215)

[Summary 46](#_Toc98888216)

[JavaScript innerHTML vs createElement 46](#_Toc98888217)

[#1) createElement is more performant 46](#_Toc98888218)

[#2) createElement is more secure 46](#_Toc98888219)

[#3) Using DocumentFragment for composing DOM Nodes 47](#_Toc98888220)

[JavaScript DocumentFragment 47](#_Toc98888221)

[Introduction to the JavaScript DocumentFragment interface 48](#_Toc98888222)

[JavaScript DocumentFragment example 48](#_Toc98888223)

[Summary 50](#_Toc98888224)

[JavaScript insertBefore 50](#_Toc98888225)

[Introduction to JavaScript insertBefore() method 50](#_Toc98888226)

[JavaScript insertBefore() helper function 50](#_Toc98888227)

[JavaScript insertBefore() example 50](#_Toc98888228)

[Summary 51](#_Toc98888229)

[JavaScript insertAfter 51](#_Toc98888230)

[Summary 53](#_Toc98888231)

[JavaScript append 53](#_Toc98888232)

[Introduction to JavaScript append() method 53](#_Toc98888233)

[JavaScript append() method examples 54](#_Toc98888234)

[**1) Using the append() method to append an element example** 54](#_Toc98888235)

[**2) Using the append() method to append text to an element example** 54](#_Toc98888236)

[append vs. appendChild() 55](#_Toc98888237)

[Summary 55](#_Toc98888238)

[JavaScript prepend 55](#_Toc98888239)

[Introduction to JavaScript prepend() method 55](#_Toc98888240)

[JavaScript prepend() method examples 56](#_Toc98888241)

[**1) Using the prepend() method to prepend an element example** 56](#_Toc98888242)

[**2) Using the prepend() method to prepend text to an element example** 56](#_Toc98888243)

[Summary 57](#_Toc98888244)

[JavaScript insertAdjacentHTML 57](#_Toc98888245)

[Introduction to JavaScript insertAdjacentHTML() method 57](#_Toc98888246)

[**1) position** 57](#_Toc98888247)

[**2) text** 58](#_Toc98888248)

[**Security consideration** 58](#_Toc98888249)

[JavaScript insertAdjacentHTML() method example 58](#_Toc98888250)

[Summary 59](#_Toc98888251)

[JavaScript replaceChild 59](#_Toc98888252)

[Summary 60](#_Toc98888253)

[JavaScript cloneNode 60](#_Toc98888254)

[**Pamaraters** 60](#_Toc98888255)

[**deep** 60](#_Toc98888256)

[**originalNode** 61](#_Toc98888257)

[**Return value** 61](#_Toc98888258)

[**Usage notes** 61](#_Toc98888259)

[JavaScript cloneNode() example 61](#_Toc98888260)

[Summary 62](#_Toc98888261)

[JavaScript removeChild 62](#_Toc98888262)

[Introduction to JavaScript removeChild() method 62](#_Toc98888263)

[JavaScript removeChild() example 63](#_Toc98888264)

[Removing all child nodes of an element 64](#_Toc98888265)

[Summary 64](#_Toc98888266)

[Understanding Relationships Between HTML Attributes & DOM Object’s Properties 64](#_Toc98888267)

[Attribute methods 65](#_Toc98888268)

[element.attributes 65](#_Toc98888269)

[Attribute-property synchronization 66](#_Toc98888270)

[DOM properties are typed 66](#_Toc98888271)

[The data-\* attributes 67](#_Toc98888272)

[Summary 67](#_Toc98888273)

[JavaScript setAttribute 68](#_Toc98888274)

[Introduction to the JavaScript setAttribute() method 68](#_Toc98888275)

[**Parameters** 68](#_Toc98888276)

[**Return value** 68](#_Toc98888277)

[JavaScript setAttribute() example 68](#_Toc98888278)

[Summary 69](#_Toc98888279)

[JavaScript getAttribute 69](#_Toc98888280)

[Introduction to the JavaScript getAttribute() method 69](#_Toc98888281)

[**Parameters** 69](#_Toc98888282)

[**Return value** 69](#_Toc98888283)

[JavaScript getAttribute() example 70](#_Toc98888284)

[Summary 71](#_Toc98888285)

[JavaScript removeAttribute 71](#_Toc98888286)

[Introduction to JavaScript removeAttribute() method 71](#_Toc98888287)

[**Parameters** 71](#_Toc98888288)

[**Return value** 71](#_Toc98888289)

[**Usage notes** 71](#_Toc98888290)

[JavaScript removeAttribute() example 71](#_Toc98888291)

[Summary 72](#_Toc98888292)

[JavaScript hasAttribute 72](#_Toc98888293)

[Introduction to the JavaScript hasAttribute() method 72](#_Toc98888294)

[**Parameters** 72](#_Toc98888295)

[**Return value** 72](#_Toc98888296)

[JavaScript hasAttribute() example 73](#_Toc98888297)

[Summary 73](#_Toc98888298)

[JavaScript Style 73](#_Toc98888299)

[Setting inline styles 73](#_Toc98888300)

[Getting inline styles 79](#_Toc98888301)

[Summary 79](#_Toc98888302)

[JavaScript getComputedStyle 79](#_Toc98888303)

[Introduction to JavaScript getComputedStyle() method 79](#_Toc98888304)

[**Parameters** 79](#_Toc98888305)

[**Return value** 79](#_Toc98888306)

[JavaScript getComputedStyle() examples 80](#_Toc98888307)

[**1) Simple getComputedStyle() example** 80](#_Toc98888308)

[**2) The getComputedStyle() for pseudo-elements example** 81](#_Toc98888309)

[Summary 81](#_Toc98888310)

[JavaScript className 82](#_Toc98888311)

[JavaScript classList 83](#_Toc98888312)

[Introduction to JavaScript classList property 83](#_Toc98888313)

[JavaScript classList examples 83](#_Toc98888314)

[**1) Get the CSS classes of an element** 83](#_Toc98888315)

[**2) Add one or more classes to the class list of an element** 84](#_Toc98888316)

[**3) Remove element’s classes** 84](#_Toc98888317)

[**4) Replace a class of an element** 84](#_Toc98888318)

[**5) Check if an element has a specified class** 84](#_Toc98888319)

[**6) Toggle a class** 85](#_Toc98888320)

[**Summary** 85](#_Toc98888321)

[Getting the Width and Height of an Element 85](#_Toc98888322)

[clientWidth & clientHeight 88](#_Toc98888323)

[Summary 89](#_Toc98888324)

[JavaScript Events 89](#_Toc98888325)

[Introduction to JavaScript events 89](#_Toc98888326)

[Event flow 90](#_Toc98888327)

[**Event bubbling** 90](#_Toc98888328)

[**Event capturing** 91](#_Toc98888329)

[**DOM Level 2 Event flow** 92](#_Toc98888330)

[Event object 93](#_Toc98888331)

[**preventDefault()** 94](#_Toc98888332)

[**stopPropagation()** 95](#_Toc98888333)

[Summary 95](#_Toc98888334)

[Handling Events in JavaScript 95](#_Toc98888335)

[1) HTML event handler attributes 96](#_Toc98888336)

[**Important notes** 96](#_Toc98888337)

[**Disadvantages of using HTML event handler attributes** 97](#_Toc98888338)

[2) DOM Level 0 event handlers 97](#_Toc98888339)

[3) DOM Level 2 event handlers 98](#_Toc98888340)

[The addEventListener() method 98](#_Toc98888341)

[The removeEventListener() method 98](#_Toc98888342)

[Summary 99](#_Toc98888343)

[JavaScript Page Load Events 99](#_Toc98888344)

[Overview of JavaScript page load events 99](#_Toc98888345)

[Handling JavaScript page load events 100](#_Toc98888346)

[JavaScript onload 101](#_Toc98888347)

[The window’s load event 101](#_Toc98888348)

[The image’s load event 101](#_Toc98888349)

[The script’s load event 102](#_Toc98888350)

[Summary 103](#_Toc98888351)

[JavaScript DOMContentLoaded 103](#_Toc98888352)

[Summary 105](#_Toc98888353)

[JavaScript beforeunload Event 105](#_Toc98888354)

[Introduction to JavaScript beforeunload event 105](#_Toc98888355)

[JavaScript beforeunload event example 106](#_Toc98888356)

[Summary 106](#_Toc98888357)

[JavaScript Mouse Events 106](#_Toc98888358)

[Introduction to JavaScript mouse events 107](#_Toc98888359)

[**mousedown, mouseup, and click** 107](#_Toc98888360)

[**dblclick** 107](#_Toc98888361)

[**mousemove** 108](#_Toc98888362)

[**mouseover / mouseout** 109](#_Toc98888363)

[**mouseenter / mouseleave** 109](#_Toc98888364)

[Registering mouse event handlers 109](#_Toc98888365)

[Detecting mouse buttons 110](#_Toc98888366)

[Modifier keys 111](#_Toc98888367)

[Getting Screen Coordinates 113](#_Toc98888368)

[Summary 114](#_Toc98888369)

[JavaScript Keyboard Events 114](#_Toc98888370)

[Introduction to JavaScript keyboard events 115](#_Toc98888371)

[Handling keyboard events 115](#_Toc98888372)

[The keyboard event properties 116](#_Toc98888373)

[Summary 116](#_Toc98888374)

[JavaScript Scroll Events 117](#_Toc98888375)

[Introduction to the JavaScript scroll events 117](#_Toc98888376)

[Scrolling the document 117](#_Toc98888377)

[**Scroll offsets** 118](#_Toc98888378)

[Scrolling an element 118](#_Toc98888379)

[The better ways to handle the scroll events 119](#_Toc98888380)

[**Event throttling** 119](#_Toc98888381)

[**Passive events** 120](#_Toc98888382)

[Summary 120](#_Toc98888383)

[JavaScript scrollIntoView 121](#_Toc98888384)

[**alignToTop** 121](#_Toc98888385)

[**options** 121](#_Toc98888386)

[JavaScript scrollIntoView() Example 121](#_Toc98888387)

[JavaScript Focus Events 123](#_Toc98888388)

[Introduction to JavaScript focus events 123](#_Toc98888389)

[JavaScript focus event examples 123](#_Toc98888390)

[Summary 124](#_Toc98888391)

[JavaScript hashchange Event 124](#_Toc98888392)

[Introduction to the JavaScript hashchange event 124](#_Toc98888393)

[Summary 125](#_Toc98888394)

[JavaScript Event Delegation 125](#_Toc98888395)

[Introduction to JavaScript Event Delegation 125](#_Toc98888396)

[JavaScript event delegation benefits 127](#_Toc98888397)

[Summary 127](#_Toc98888398)

[JavaScript dispatchEvent 127](#_Toc98888399)

[Event constructor 127](#_Toc98888400)

[**type** 128](#_Toc98888401)

[**options** 128](#_Toc98888402)

[dispatchEvent method 128](#_Toc98888403)

[Summary 129](#_Toc98888404)

[JavaScript Custom Events 129](#_Toc98888405)

[Introduction to JavaScript custom events 129](#_Toc98888406)

[**Creating JavaScript custom events** 130](#_Toc98888407)

[**Dispatching JavaScript custom events** 131](#_Toc98888408)

[JavaScript custom event example 131](#_Toc98888409)

[Why using custom events 132](#_Toc98888410)

[Summary 132](#_Toc98888411)

[JavaScript MutationObserver 132](#_Toc98888412)

[Introduction to the JavaScript MutationObserver API 132](#_Toc98888413)

[The MutationObserver options 133](#_Toc98888414)

[**Observing changes to child elements** 133](#_Toc98888415)

[**Observing for changes to attributes** 136](#_Toc98888416)

[**Observing for changes to a subtree** 137](#_Toc98888417)

[**Observing for changes to character data** 137](#_Toc98888418)

[**Accessing old values** 137](#_Toc98888419)

[JavaScript Form 137](#_Toc98888420)

[Form basics 138](#_Toc98888421)

[Referencing forms 138](#_Toc98888422)

[Submitting a form 139](#_Toc98888423)

[Accessing form fields 139](#_Toc98888424)

[Put it all together: signup form 141](#_Toc98888425)

[**The showMessage() function** 144](#_Toc98888426)

[**The showError() and showSuccess() functions** 145](#_Toc98888427)

[**The hasValue() function** 145](#_Toc98888428)

[**The validateEmail() function** 145](#_Toc98888429)

[**The submit event handler** 146](#_Toc98888430)

[Summary 146](#_Toc98888431)

[JavaScript Radio Button 147](#_Toc98888432)

[Introduction to the JavaScript Radio Button 147](#_Toc98888433)

[Radio button’s change event 150](#_Toc98888434)

[Summary 152](#_Toc98888435)

[JavaScript select Element 152](#_Toc98888436)

[Introduction to the HTML select elements 152](#_Toc98888437)

[The HTMLSelectElement type 153](#_Toc98888438)

[**The selectedIndex property** 153](#_Toc98888439)

[**The value property** 154](#_Toc98888440)

[The HTMLOptionElement type 155](#_Toc98888441)

[Summary 156](#_Toc98888442)

[JavaScript: Dynamically Add & Remove Options 157](#_Toc98888443)

[Adding options 157](#_Toc98888444)

[**1) Using the Option constructor and add() method** 157](#_Toc98888445)

[**2) Using the DOM methods** 157](#_Toc98888446)

[Removing Options 158](#_Toc98888447)

[Put it all together 158](#_Toc98888448)

[JavaScript change Event 161](#_Toc98888449)

[Using JavaScript change event for input elements 161](#_Toc98888450)

[Using JavaScript change event for radio buttons 162](#_Toc98888451)

[Using JavaScript change event for checkboxes 163](#_Toc98888452)

[Using JavaScript change event for the select element 164](#_Toc98888453)

[Summary 165](#_Toc98888454)

[JavaScript input Event 165](#_Toc98888455)

[Introduction to the JavaScript input event 165](#_Toc98888456)

Document Object Model in JavaScript

**Summary**: in this tutorial, you will learn about the Document Object Model in JavaScript.

## **What is Document Object Model (DOM)**

The Document Object Model (DOM) is an application programming interface (API) for manipulating HTML documents.

The DOM represents an HTML document as a tree of nodes. The DOM provides functions that allow you to add, remove, and modify parts of the document effectively.

Note that the DOM is cross-platform and language-independent ways of manipulating HTML and XML documents.

## **A document as a hierarchy of nodes**

The DOM represents an HTML document as a hierarchy of nodes. Consider the following HTML document:

**<html>**

**<head>**

**<title>**JavaScript DOM**</title>**

**</head>**

**<body>**

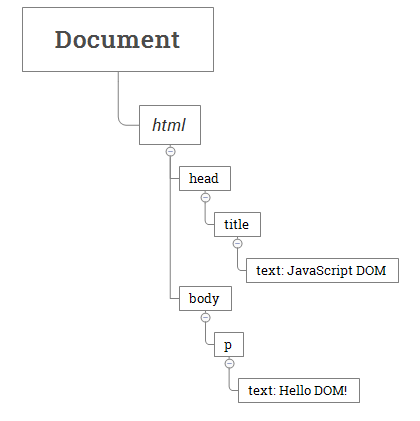
**<p>**Hello DOM!**</p>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

The following tree represents the above HTML document:



In this DOM tree, the document is the root node. The root node has one child node which is the <html> element. The <html> element is called the document element.

Each document can have only one document element. In an HTML document, the document element is the <html> element. Each markup can be represented by a node in the tree.

### **Node Types**

Each node in the DOM tree is identified by a node type. JavaScript uses integer numbers to determine the node types. The following table illustrates the node type constants:

| **Constant** | **Value** | **Description** |
| --- | --- | --- |
| Node.ELEMENT\_NODE | 1 | An Element node like <p> or <div>. |
| Node.TEXT\_NODE | 3 | The actual Text inside an Element or Attr. |
| Node.CDATA\_SECTION\_NODE | 4 | A CDATASection, such as <!CDATA[[ … ]]>. |
| Node.PROCESSING\_INSTRUCTION\_NODE | 7 | A ProcessingInstruction of an XML document, such as <?xml-stylesheet … ?>. |
| Node.COMMENT\_NODE | 8 | A Comment node, such as <!-- … -->. |
| Node.DOCUMENT\_NODE | 9 | A Document node. |
| Node.DOCUMENT\_TYPE\_NODE | 10 | A DocumentType node, such as <!DOCTYPE html>. |
| Node.DOCUMENT\_FRAGMENT\_NODE | 11 | A DocumentFragment node. |

To get the type of node, you use the nodeType property:

node.nodeType

Code language: CSS (css)

You can compare the nodeType property with the above constants to determine the node type. For example:

if (node.nodeType == Node.ELEMENT\_NODE) {

*// node is the element node*

}

Code language: JavaScript (javascript)

### **The nodeName and nodeValue properties**

A node has two important properties: nodeName and nodeValue that provide specific information about the node.

The values of these properties depend on the node type. For example, if the node type is the element node, the nodeName is always the same as the element’s tag name and nodeValue is always null.

For this reason, it’s better to test node type before using these properties:

if (node.nodeType == Node.ELEMENT\_NODE) {

let name = node.nodeName; *// tag name like <p>*

}

Code language: JavaScript (javascript)

### **Node and Element**

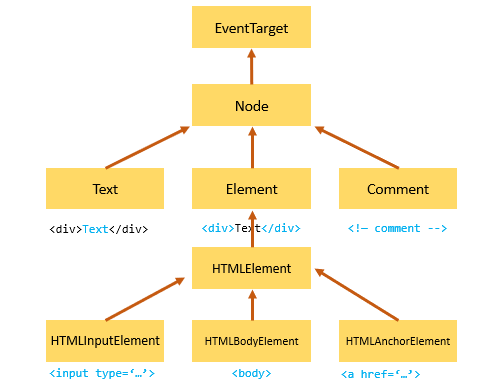
Sometimes it’s easy to confuse between the Node and the Element.

A node is a generic name of any object in the DOM tree. It can be any built-in DOM element such as the document. Or it can be any HTML tag specified in the HTML document like <div> or <p>.

An element is a node with a specific node type Node.ELEMENT\_NODE, which is equal to 1.

In other words, the node is the generic type of element. The element is a specific type of the node with the node type Node.ELEMENT\_NODE.

The following picture illustrates the relationship between the Node and Element types:



Note that the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) and [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) returns an object with the Element type while [getElementsByTagName()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementsbytagname/) or [querySelectorAll()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) returns NodeList which is a collection of nodes.

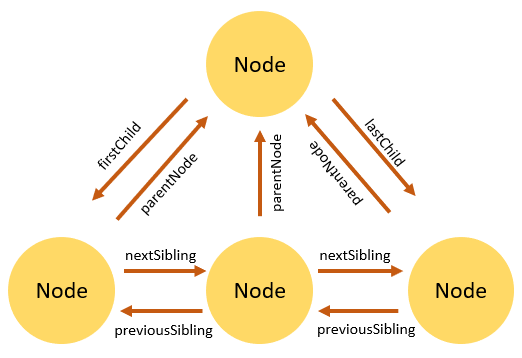
### **Node Relationships**

Any node has relationships to other nodes in the DOM tree. The relationships are the same as the ones described in a traditional family tree.

For example, <body> is a [child node](https://www.javascripttutorial.net/javascript-dom/javascript-get-child-element/) of the <html> node, and <html> is the [parent](https://www.javascripttutorial.net/javascript-dom/javascript-get-parent-element-parentnode/) of the <body> node.

The <body> node is the [sibling](https://www.javascripttutorial.net/javascript-dom/javascript-siblings/) of the <head> node because they share the same immediate parent, which is the <html> element.

The following picture illustrates the relationships between nodes:



## **Summary**

* An HTML or XML document can be represented as a tree of nodes, like a traditional family tree.
* Each markup can be represented as a node with a specific node type.
* Element is a specific type of node with the node type Node.ELEMENT\_NODE.
* In the DOM tree, a node has relationships with other nodes.

# JavaScript getElementById

**Summary**: in this tutorial, you will learn how to use the JavaScript getElementById() to select an element by a specified id.

## **Introduction to JavaScript getElementById() method**

The document.getElementById() method returns an Element object that represents an HTML element with an id that matches a specified string.

If the document has no element with the specified id, the document.getElementById() returns null.

Because the id of an element is unique within an HTML document, the document.getElementById() is a quick way to access an element.

Unlike the querySelector() method, the getElementById() is only available on the document object, not other elements.

The following shows the syntax of the getElementById() method:

const element = document.getElementById(id);

Code language: JavaScript (javascript)

In this syntax, the id is a string that represents the id of the element to select. The id is case-sensitive. For example, the 'root' and 'Root' are totally different.

The id is unique within an HTML document. However, HTML is a forgiving language. If the HTML document has multiple elements with the same id, the document.getElementById() method returns the first element it encounters.

## **JavaScript getElementById() method example**

Suppose you have the following HTML document:

**<html>**

**<head>**

**<title>**JavaScript getElementById() Method**</title>**

**</head>**

**<body>**

**<p id="message">**A paragraph**</p>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

The document contains a <p> element that has the id attribute with the value message:

const p = document.getElementById('message');

console.log(p);

Code language: JavaScript (javascript)

Output:

**<p id="message">**A paragraph**</p>**

Code language: HTML, XML (xml)

After selecting an element, you can [add styles to the element](https://www.javascripttutorial.net/dom/css/add-styles-to-an-element/), manipulate its [attributes](https://www.javascripttutorial.net/dom/attributes/set-the-value-of-an-attribute/), and traverse to [parent](https://www.javascripttutorial.net/javascript-dom/javascript-get-parent-element-parentnode/) and [child elements](https://www.javascripttutorial.net/javascript-dom/javascript-get-child-element/).

## **Summary**

* The document.getElementById() returns a DOM element specified by an id or null if no matching element found.
* If multiple elements have the same id, even though it is invalid, the getElementById() returns the first element it encounters.

# JavaScript getElementsByName

**Summary**: in this tutorial, you will learn how to use the JavaScript getElementsByName() method to get elements with a given name in a document.

## **Introduction to JavaScript getElementsByName() method**

Every element on an HTML document may have a name attribute:

**<input type="radio" name="language" value="JavaScript">**

Code language: HTML, XML (xml)

Unlike the id attribute, multiple HTML elements can share the same value of the name attribute like this:

**<input type="radio" name="language" value="JavaScript">**

**<input type="radio" name="language" value="TypeScript">**

Code language: HTML, XML (xml)

To get all elements with a specified name, you use the getElementsByName() method of the document object:

let elements = document.getElementsByName(name);

Code language: JavaScript (javascript)

The getElementsByName() accepts a name which is the value of the name attribute of elements and returns a live NodeList of elements.

The return collection of elements is live. It means that the return elements are automatically updated when elements with the same name are [inserted](https://www.javascripttutorial.net/dom/manipulating/insert-an-element-before-an-existing-element/) and/or [removed](https://www.javascripttutorial.net/javascript-dom/javascript-removechild/) from the document.

## **JavaScript getElementsByName() example**

The following example shows a radio group that consists of [radio buttons](https://www.javascripttutorial.net/javascript-dom/javascript-radio-button/) that have the same name (rate).

When you select a radio button and click the submit button, the page will show the selected value such as Very Poor, Poor, OK, Good, or Very Good:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript getElementsByName Demo**</title>**

**</head>**

**<body>**

**<p>**Please rate the service:**</p>**

**<p>**

**<label for="very-poor">**

**<input type="radio" name="rate" value="Very poor" id="very-poor">** Very poor

**</label>**

**<label for="poor">**

**<input type="radio" name="rate" value="Poor" id="poor">** Poor

**</label>**

**<label for="ok">**

**<input type="radio" name="rate" value="OK" id="ok">** OK

**</label>**

**<label for="good">**

**<input type="radio" name="rate" value="Good">** Good

**</label>**

**<label for="very-good">**

**<input type="radio" name="rate" value="Very Good" id="very-good">** Very Good

**</label>**

**</p>**

**<p>**

**<button id="btnRate">**Submit**</button>**

**</p>**

**<p id="output"></p>**

**<script>**

let btn = document.getElementById('btnRate');

let output = document.getElementById('output');

btn.addEventListener('click', () => {

let rates = document.getElementsByName('rate');

rates.forEach((rate) => {

if (rate.checked) {

output.innerText = `You selected: ${rate.value}`;

}

});

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, select the submit button by its id btnRate using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Second, listen to the [click](https://www.javascripttutorial.net/javascript-dom/javascript-mouse-events/) event of the submit button.
* Third, get all the radio buttons using the getElementsByName() and show the selected value in the output element.

Notice that you will learn about [events](https://www.javascripttutorial.net/javascript-dom/javascript-events/) like click later. For now, you just need to focus on the getElementsByName() method.

## **Summary**

* The getElementsByName() returns a live NodeList of elements with a specified name.
* The NodeList is an array-like object, not an [array](https://www.javascripttutorial.net/javascript-array/) object.

# JavaScript getElementsByName

**Summary**: in this tutorial, you will learn how to use the JavaScript getElementsByName() method to get elements with a given name in a document.

## **Introduction to JavaScript getElementsByName() method**

Every element on an HTML document may have a name attribute:

**<input type="radio" name="language" value="JavaScript">**

Code language: HTML, XML (xml)

Unlike the id attribute, multiple HTML elements can share the same value of the name attribute like this:

**<input type="radio" name="language" value="JavaScript">**

**<input type="radio" name="language" value="TypeScript">**

Code language: HTML, XML (xml)

To get all elements with a specified name, you use the getElementsByName() method of the document object:

let elements = document.getElementsByName(name);

Code language: JavaScript (javascript)

The getElementsByName() accepts a name which is the value of the name attribute of elements and returns a live NodeList of elements.

The return collection of elements is live. It means that the return elements are automatically updated when elements with the same name are [inserted](https://www.javascripttutorial.net/dom/manipulating/insert-an-element-before-an-existing-element/) and/or [removed](https://www.javascripttutorial.net/javascript-dom/javascript-removechild/) from the document.

## **JavaScript getElementsByName() example**

The following example shows a radio group that consists of [radio buttons](https://www.javascripttutorial.net/javascript-dom/javascript-radio-button/) that have the same name (rate).

When you select a radio button and click the submit button, the page will show the selected value such as Very Poor, Poor, OK, Good, or Very Good:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript getElementsByName Demo**</title>**

**</head>**

**<body>**

**<p>**Please rate the service:**</p>**

**<p>**

**<label for="very-poor">**

**<input type="radio" name="rate" value="Very poor" id="very-poor">** Very poor

**</label>**

**<label for="poor">**

**<input type="radio" name="rate" value="Poor" id="poor">** Poor

**</label>**

**<label for="ok">**

**<input type="radio" name="rate" value="OK" id="ok">** OK

**</label>**

**<label for="good">**

**<input type="radio" name="rate" value="Good">** Good

**</label>**

**<label for="very-good">**

**<input type="radio" name="rate" value="Very Good" id="very-good">** Very Good

**</label>**

**</p>**

**<p>**

**<button id="btnRate">**Submit**</button>**

**</p>**

**<p id="output"></p>**

**<script>**

let btn = document.getElementById('btnRate');

let output = document.getElementById('output');

btn.addEventListener('click', () => {

let rates = document.getElementsByName('rate');

rates.forEach((rate) => {

if (rate.checked) {

output.innerText = `You selected: ${rate.value}`;

}

});

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, select the submit button by its id btnRate using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Second, listen to the [click](https://www.javascripttutorial.net/javascript-dom/javascript-mouse-events/) event of the submit button.
* Third, get all the radio buttons using the getElementsByName() and show the selected value in the output element.

Notice that you will learn about [events](https://www.javascripttutorial.net/javascript-dom/javascript-events/) like click later. For now, you just need to focus on the getElementsByName() method.

## **Summary**

* The getElementsByName() returns a live NodeList of elements with a specified name.
* The NodeList is an array-like object, not an [array](https://www.javascripttutorial.net/javascript-array/) object.

# JavaScript getElementsByTagName

**Summary**: in this tutorial, you will learn how to use the JavaScript getElementsByTagName() to select elements with a given tag name.

## **Introduction to JavaScript getElementsByTagName() method**

The getElementsByTagName() is a method of the document object or a specific DOM element.

The getElementsByTagName() method accepts a tag name and returns a live HTMLCollection of elements with the matching tag name in the order which they appear in the document.

The following illustrates the syntax of the getElementsByTagName():

let elements = document.getElementsByTagName(tagName);

Code language: JavaScript (javascript)

The return collection of the getElementsByTagName() is live, meaning that it is automatically updated when elements with the matching tag name are added and/or removed from the document.

Note that the HTMLCollection is an array-like object, like arguments object of a function.

## **JavaScript getElementsByTagName() example**

The following example illustrates how to use the getElementsByTagName() method to get the number of H2 tags in the document.

When you click the **Count H2** button, the page shows the number of H2 tags:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JavaScript getElementsByTagName() Demo**</title>**

**</head>**

**<body>**

**<h1>**JavaScript getElementsByTagName() Demo**</h1>**

**<h2>**First heading**</h2>**

**<p>**This is the first paragraph.**</p>**

**<h2>**Second heading**</h2>**

**<p>**This is the second paragraph.**</p>**

**<h2>**Third heading**</h2>**

**<p>**This is the third paragraph.**</p>**

**<button id="btnCount">**Count H2**</button>**

**<script>**

let btn = document.getElementById('btnCount');

btn.addEventListener('click', () => {

let headings = document.getElementsByTagName('h2');

alert(`The number of H2 tags: ${headings.length}`);

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, select the button **Count H2** by using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Second, hook the click event of the button to an anonymous function.
* Third, in the anonymous function, use the document.getElementsByTagName() to get a list of H2 tags.
* Finally, show the number of H2 tags using the alert() function.

## **Summary**

* The getElementsByTagName() is a method of the document or element object.
* The getElementsByTagName() accepts a tag name and returns a list of elements with the matching tag name.
* The getElementsByTagName() returns a live HTMLCollection of elements. The HTMLCollection is an array-like object.

# JavaScript getElementsByClassName

**Summary**: in this tutorial, you will learn how to use the getElementsByClassName() method to select elements by class name.

## **Introduction to the getElementsByClassName() method**

The getElementsByClassName() method returns an array-like of objects of the child elements with a specified class name. The getElementsByClassName() method is available on the document element or any other elements.

When calling the method on the document element, it searches the entire document and returns the child elements of the document:

let elements = document.getElementsByClassName(names);

Code language: JavaScript (javascript)

However, when calling the method on a specific element, it returns the descendants of that specific element with the given class name:

let elements = rootElement.getElementsByClassName(names);

Code language: JavaScript (javascript)

The method returns the elements which is a live HTMLCollection of the matches elements.

The names parameter is a string that represents one or more class names to match; To use multiple class names, you separate them by space.

## **JavaScript getElementsByClassName() method examples**

Let’s take some examples of using the getElementsByClassName() method.

Suppose that you have the following HTML document:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript getElementsByClassName**</title>**

**</head>**

**<body>**

**<header>**

**<nav>**

**<ul id="menu">**

**<li class="item">**HTML**</li>**

**<li class="item">**CSS**</li>**

**<li class="item highlight">**JavaScript**</li>**

**<li class="item">**TypeScript**</li>**

**</ul>**

**</nav>**

**<h1>**getElementsByClassName Demo**</h1>**

**</header>**

**<section>**

**<article>**

**<h2 class="secondary">**Example 1**</h2>**

**</article>**

**<article>**

**<h2 class="secondary">**Example 2**</h2>**

**</article>**

**</section>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

### **1) Calling JavaScript getElementsByClassName() on an element example**

The following example illustrates how to use the getElementsByClassName() method to select the <li> items which are the descendants of the <ul> element:

let menu = document.getElementById('menu');

let items = menu.getElementsByClassName('item');

let data = [].map.call(items, item => item.textContent);

console.log(data);

Code language: JavaScript (javascript)

Output:

['HTML', 'CSS', 'JavaScript', 'TypeScript']

Code language: JavaScript (javascript)

How it works:

* First, select the <ul> element with the class name menu using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Then, select <li> elements, which are the descendants of the <ul> element, using the getElementsByClassName() method.
* Finally, create an array of the text content of <li> elements by borrowing the map() method of the Array object.

### **2) Calling JavaScript getElementsByClassName() on the document example**

To search for the element with the class heading-secondary, you use the following code:

let elements = document.getElementsByClassName('secondary');

let data = [].map.call(elements, elem => elem.textContent);

console.log(data);

Code language: JavaScript (javascript)

Output:

["Example 1", "Example 2"]

Code language: JavaScript (javascript)

This example calls the getElementsByClassName() method on the document object. Therefore, it searches for the elements with the class secondary in the entire document.

## **Summary**

* Use the JavaScript getElementsByClassName() method to select the child elements of an element that has one or more give class names.

# JavaScript querySelector

**Summary**: in this tutorial, you will learn how to use the JavaScript querySelector() and querySelectorAll() to find elements based on CSS selectors.

## **Introduction to JavaScript querySelector() and querySelectorAll() methods**

The querySelector() is a method of the Element interface. The querySelector() method allows you to select the first element that matches one or more CSS selectors.

The following illustrates the syntax of the querySelector() method:

let element = parentNode.querySelector(selector);

Code language: JavaScript (javascript)

In this syntax, the selector is a CSS selector or a group of CSS selectors to match the descendant elements of the parentNode.

If the selector is not valid CSS syntax, the method will raise a SyntaxError exception.

If no element matches the CSS selectors, the querySelector() returns null.

The querySelector() method is available on the document object or any Element object.

Besides the querySelector(), you can use the querySelectorAll() method to select all elements that match a CSS selector or a group of CSS selectors:

let elementList = parentNode.querySelectorAll(selector);

Code language: JavaScript (javascript)

The querySelectorAll() method returns a static NodeList of elements that match the CSS selector. If no element matches, it returns an empty NodeList.

Note that the NodeList is an array-like object, not an array object. However, in modern web browsers, you can use the [forEach()](https://www.javascripttutorial.net/javascript-array-foreach/) method or the [for...of](https://www.javascripttutorial.net/es6/javascript-for-of/) loop.

To convert the NodeList to an array, you use the Array.from() method like this:

let nodeList = document.querySelectorAll(selector);

let elements = Array.from(nodeList);

Code language: JavaScript (javascript)

## **Basic selectors**

Suppose that you have the following HTML document:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<title>**querySelector() Demo**</title>**

**</head>**

**<body>**

**<header>**

**<div id="logo">**

**<img src="img/logo.jpg" alt="Logo" id="logo">**

**</div>**

**<nav class="primary-nav">**

**<ul>**

**<li class="menu-item current"><a href="#home">**Home**</a></li>**

**<li class="menu-item"><a href="#services">**Services**</a></li>**

**<li class="menu-item"><a href="#about">**About**</a></li>**

**<li class="menu-item"><a href="#contact">**Contact**</a></li>**

**</ul>**

**</nav>**

**</header>**

**<main>**

**<h1>**Welcome to the JS Dev Agency**</h1>**

**<div class="container">**

**<section class="section-a">**

**<h2>**UI/UX**</h2>**

**<p>**Lorem ipsum dolor sit amet, consectetur adipisicing elit. Autem placeat, atque accusamus voluptas

laudantium facilis iure adipisci ab veritatis eos neque culpa id nostrum tempora tempore minima.

Adipisci, obcaecati repellat.**</p>**

**<button>**Read More**</button>**

**</section>**

**<section class="section-b">**

**<h2>**PWA Development**</h2>**

**<p>**Lorem ipsum dolor sit, amet consectetur adipisicing elit. Magni fugiat similique illo nobis quibusdam

commodi aspernatur, tempora doloribus quod, consectetur deserunt, facilis natus optio. Iure

provident labore nihil in earum.**</p>**

**<button>**Read More**</button>**

**</section>**

**<section class="section-c">**

**<h2>**Mobile App Dev**</h2>**

**<p>**Lorem ipsum dolor sit amet consectetur adipisicing elit. Animi eos culpa laudantium consequatur ea!

Quibusdam, iure obcaecati. Adipisci deserunt, alias repellat eligendi odit labore! Fugit iste sit

laborum debitis eos?**</p>**

**<button>**Read More**</button>**

**</section>**

**</div>**

**</main>**

**<script src="js/main.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

### **1) Universal selector**

The universal selector is denoted by \* that matches all elements of any type:

\*

The following example uses the querySelector() selects the first element in the document:

let element = document.querySelector('\*');

Code language: JavaScript (javascript)

And this select all elements in the document:

let elements = document.querySelectorAll('\*');

Code language: JavaScript (javascript)

### **2) Type selector**

To select elements by node name, you use the type selector e.g., a selects all <a> elements:

elementName

The following example finds the first h1 element in the document:

let firstHeading = document.querySelector('h1');

Code language: JavaScript (javascript)

And the following example finds all h2 elements:

let heading2 = document.querySelectorAll('h2');

Code language: JavaScript (javascript)

### **3) Class selector**

To find the element with a given CSS class, you use the class selector syntax:

.className

Code language: CSS (css)

The following example finds the first element with the menu-item class:

let note = document.querySelector('.menu-item');

Code language: JavaScript (javascript)

And the following example finds all elements with the menu class:

let notes = document.querySelectorAll('.menu-item');

Code language: JavaScript (javascript)

### **4) ID Selector**

To select an element based on the value of its id, you use the id selector syntax:

#id

Code language: CSS (css)

The following example finds the first element with the id #logo:

let logo = document.querySelector('#logo');

Code language: JavaScript (javascript)

Since the id should be unique in the document, the querySelectorAll() is not relevant.

### **5) Attribute selector**

To select all elements that have a given attribute, you use one of the following attribute selector syntaxes:

[attribute]

[attribute=value]

[attribute~=value]

[attribute|=value]

[attribute^=value]

[attribute$=value]

[attribute\*$\*=value]

Code language: JSON / JSON with Comments (json)

The following example finds the first element with the attribute [autoplay] with any value:

let autoplay = document.querySelector('[autoplay]');

Code language: JavaScript (javascript)

And the following example finds all elements that have [autoplay] attribute with any value:

let autoplays = document.querySelectorAll('[autoplay]');

Code language: JavaScript (javascript)

## **Grouping selectors**

To group multiple selectors, you use the following syntax:

selector, selector, ...

The selector list will match any element with one of the selectors in the group.

The following example finds all <div> and <p> elements:

let elements = document.querySelectorAll('div, p');

Code language: JavaScript (javascript)

## **Combinators**

### **1) descendant combinator**

To find descendants of a node, you use the space ( ) descendant combinator syntax:

selector selector

For example p a will match all <a> elements inside the p element:

let links = document.querySelector('p a');

Code language: JavaScript (javascript)

### **2) Child combinator**

The > child combinator finds all elements that are direct children of the first element:

selector > selector

The following example finds all li elements that are directly inside a <ul> element:

let listItems = document.querySelectorAll('ul > li');

Code language: JavaScript (javascript)

To select all li elements that are directly inside a <ul> element with the class nav:

let listItems = document.querySelectorAll('ul.nav > li');

Code language: JavaScript (javascript)

### **3) General sibling combinator**

The ~ combinator selects siblings that share the same parent:

selector ~ selector

For example, p ~ a will match all <a> elements that follow the p element, immediately or not:

let links = document.querySelectorAll('p ~ a');

Code language: JavaScript (javascript)

### **4) Adjacent sibling combinator**

The + adjacent sibling combinator selects adjacent siblings:

selector + selector

For example, h1 + a matches all elements that directly follow an h1:

let links = document.querySelectorAll('h1 + a');

Code language: JavaScript (javascript)

And select the first <a> that directly follows an h1:

let links = document.querySelector('h1 + a');

Code language: JavaScript (javascript)

## **Pseudo**

### **1) Pseudo-classes**

The : pseudo matches elements based on their states:

element:state

Code language: CSS (css)

For example, the li:nth-child(2) selects the second <li> element in a list:

let listItem = document.querySelectorAll('li:nth-child(2)');

Code language: JavaScript (javascript)

### **2) Pseudo-elements**

The :: represent entities that are not included in the document.

For example, p::first-line matches the first-line of all div elements:

let links = document.querySelector('p::first-line');

Code language: JavaScript (javascript)

## **Summary**

* The querySelector() finds the first element that matches a CSS selector or a group of CSS selectors.
* The querySelectorAll() finds all elements that match a CSS selector or a group of CSS selectors.
* A CSS selector defines elements to which a CSS rule applies.

# JavaScript Get the Parent Element parentNode

**Summary**: in this tutorial, you will learn how to get the parent node of an element by using the JavaScript parentNode attribute of the Node object.

## **Introduction to parentNode attribute**

To get the parent node of a specified node in the DOM tree, you use the parentNode property:

let parent = node.parentNode;

Code language: JavaScript (javascript)

The parentNode is read-only.

The Document and DocumentFragment nodes do not have a parent. Therefore, the parentNode will always be null.

If you create a new node but haven’t attached it to the DOM tree, the parentNode of that node will also be null.

## **JavaScript parentNode example**

See the following HTML document:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript parentNode**</title>**

**</head>**

**<body>**

**<div id="main">**

**<p class="note">**This is a note!**</p>**

**</div>**

**<script>**

let note = document.querySelector('.note');

console.log(note.parentNode);

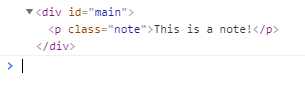
**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

The following picture shows the output in the Console:



How it works:

* First, select the element with the .note class by using the querySelector() method.
* Second, find the parent node of the element.

## **Summary**

* The node.parentNode returns the read-only parent node of a specified node or null if it does not exist.
* The document and DocumentFragment do not have a parent node.

# Getting Child Elements of a Node in JavaScript

**Summary**: in this tutorial, you will learn how to get the first child element, last child element, and all children of a specified element.

Suppose that you have the following HTML fragment:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JS Get Child Elements**</title>**

**</head>**

**<body>**

**<ul id="menu">**

**<li class="first">**Home**</li>**

**<li>**Products**</li>**

**<li class="current">**Customer Support**</li>**

**<li>**Careers**</li>**

**<li>**Investors**</li>**

**<li>**News**</li>**

**<li class="last">**About Us**</li>**

**</ul>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **Get the first child element**

To get the first child element of a specified element, you use the firstChild property of the element:

let firstChild = parentElement.firstChild;

Code language: JavaScript (javascript)

If the parentElement does not have any child element, the firstChild returns null. The firstChild property returns a child node which can be any node type such as an element node, a text node, or a comment node. The following script shows the first child of the #menu element:

let content = document.getElementById('menu');

let firstChild = content.firstChild.nodeName;

console.log(firstChild);

Code language: JavaScript (javascript)

Output:

#text

Code language: CSS (css)

The Console window show #text because a text node is inserted to maintain the whitespace between the openning <ul> and <li> tags. This whitespace creates a #text node.

Note that any whitespace such as a single space, multiple spaces, returns, and tabs will create a #text node. To remove the #text node, you can remove the whitespaces as follows:

**<article id="content"><h2>**Heading**</h2><p>**First paragraph**</p></article>**

Code language: HTML, XML (xml)

Or to get the first child with the Element node only, you can use the firstElementChild property:

let firstElementChild = parentElement.firstElementChild;

Code language: JavaScript (javascript)

The following code returns the first list item which is the first child element of the menu:

let content = document.getElementById('menu');

console.log(content.firstElementChild);

Code language: JavaScript (javascript)

Output:

**<li class="first">**Home**</li>**

Code language: HTML, XML (xml)

In this example:

* First, select the #menu element by using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Second,  get the first child element by using the firstElementChild property.

## **Get the last child element**

To get the last child element of a node, you use the lastChild property:

let lastChild = parentElement.lastChild;

Code language: JavaScript (javascript)

In case the parentElement does not have any child element, the lastChild returns null. Similar to the the firstChild property, the lastChild property returns the first element node, text node, or comment node. If you want to select only the last child element with the element node type, you use the lastElementChild property:

let lastChild = parentElement.lastElementChild;

Code language: JavaScript (javascript)

The following code returns the list item which is the last child element of the menu:

let menu = document.getElementById('menu');

console.log(main.lastElementChild);

Code language: JavaScript (javascript)

Output:

**<li class="last">**About Us**</li>**

Code language: HTML, XML (xml)

## **Get all child elements**

To get a live NodeList of child elements of a specified element, you use the childNodes property:

let children = parentElement.childNodes;

Code language: JavaScript (javascript)

The childNodes property returns all child elements with any node type. To get the child element with only the element node type, you use the children property:

let children = parentElement.children;

Code language: JavaScript (javascript)

The following example selects all child elements of the element with the Id main:

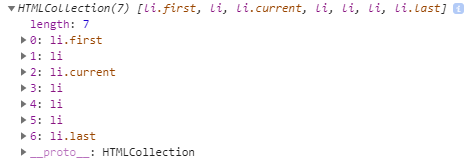
let menu = document.getElementById('menu');

let children = menu.children;

console.log(children);

Code language: JavaScript (javascript)

Output:



## **Summary**

* The firstChild and lastChild return the first and last child of a node, which can be any node type including text node, comment node, and element node.
* The firstElementChild and lastElementChild return the first and last child Element node.
* The childNodes returns a live NodeList of all child nodes of any node type of a specified node. The children return all child Element nodes of a specified node.

# JavasScript Siblings

**Summary**: in this tutorial, you will learn how to select the next siblings, previous siblings, and all siblings of an element. Let’s say you have the following list of items:

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**Products**</li>**

**<li class="current">**Customer Support**</li>**

**<li>**Careers**</li>**

**<li>**Investors**</li>**

**<li>**News**</li>**

**<li>**About Us**</li>**

**</ul>**

Code language: HTML, XML (xml)

## **Get the next siblings**

To get the next sibling of an element, you use the nextElementSibling attribute:

let nextSibling = currentNode.nextElementSibling;

Code language: JavaScript (javascript)

The nextElementSibling returns null if the specified element is the first one in the list. The following example uses the nextElementSibling property to get the next sibling of the list item that has the current class:

let current = document.querySelector('.current');

let nextSibling = current.nextElementSibling;

console.log(nextSibling);

Code language: JavaScript (javascript)

Output:

**<li>**Careers**</li>**

Code language: HTML, XML (xml)

In this example:

* First, select the list item whose class is current using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-siblings/).
* Second, get the next sibling of that list item using the nextElementSibling property.

To get all the next siblings of an element, you can use the following code:

let current = document.querySelector('.current');

let nextSibling = current.nextElementSibling;

while(nextSibling) {

console.log(nextSibling);

nextSibling = nextSibling.nextElementSibling;

}

Code language: JavaScript (javascript)

## **Get the previous siblings**

To get the previous siblings of an element, you use the previousElementSibling attribute:

let current = document.querySelector('.current');

let prevSibling = currentNode.previousElementSibling;

Code language: JavaScript (javascript)

The previousElementSibling property returns null if the current element is the first one in the list.

The following example uses the previousElementSibling property to get the previous siblings of the list item that has the current class:

let current = document.querySelector('.current');

let prevSiblings = current.previousElementSibling;

console.log(prevSiblings);

Code language: JavaScript (javascript)

And the following example selects all the previous siblings of the list item that has the current class:

let current = document.querySelector('.current');

let prevSibling = current.previousElementSibling;

while(prevSibling) {

console.log(prevSibling);

prevSibling = current.previousElementSibling;

}

Code language: JavaScript (javascript)

## **Get all siblings of an element**

To get all siblings of an element, we’ll use the logic:

* First, select the parent of the element whose siblings that you want to find.
* Second, select the first child element of that parent element.
* Third, add the first element to an array of siblings.
* Fourth, select the next sibling of the first element.
* Finally, repeat the 3rd and 4th steps until there are no siblings left. In case the sibling is the original element, skip the 3rd step.

The following function illustrates the steps:

let getSiblings = function (e) {

*// for collecting siblings*

let siblings = [];

*// if no parent, return no sibling*

if(!e.parentNode) {

return siblings;

}

*// first child of the parent node*

let sibling = e.parentNode.firstChild;

*// collecting siblings*

while (sibling) {

if (sibling.nodeType === 1 && sibling !== e) {

siblings.push(sibling);

}

sibling = sibling.nextSibling;

}

return siblings;

};

Code language: JavaScript (javascript)

Put it all together:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript Siblings**</title>**

**</head>**

**<body>**

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**Products**</li>**

**<li class="current">**Customer Support**</li>**

**<li>**Careers**</li>**

**<li>**Investors**</li>**

**<li>**News**</li>**

**<li>**About Us**</li>**

**</ul>**

**<script>**

let getSiblings = function (e) {

*// for collecting siblings*

let siblings = [];

*// if no parent, return no sibling*

if(!e.parentNode) {

return siblings;

}

*// first child of the parent node*

let sibling = e.parentNode.firstChild;

*// collecting siblings*

while (sibling) {

if (sibling.nodeType === 1 && sibling !== e) {

siblings.push(sibling);

}

sibling = sibling.nextSibling;

}

return siblings;

};

let siblings = getSiblings(document.querySelector('.current'));

siblingText = siblings.map(e => e.innerHTML);

console.log(siblingText);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Output:

["Home", "Products", "Careers", "Investors", "News", "About Us"]

Code language: JSON / JSON with Comments (json)

## **Summary**

* The nextElementSibling returns the next sibling of an element or null if the element is the last one in the list.
* The previousElementSibling returns the previous sibling of an element or null if the element is the first one in the list.
* To get all siblings of an element, you can use a helper function that utilizes the nextElementSibling property.

# JavaScript CreateElement

**Summary**: in this tutorial, you will learn how to use the JavaScript document.createElement() to create a new HTML element and attach it to the DOM tree.

To create an HTML element, you use the document.createElement() method:

let element = document.createElement(htmlTag);

Code language: JavaScript (javascript)

The document.createElement() accepts an HTML tag name and returns a new Node with the Element type.

## **1) Creating a new div example**

Suppose that you have the following HTML document:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JS CreateElement Demo**</title>**

**</head>**

**<body>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

The following example uses the document.createElement() to create a new <div> element:

let div = document.createElement('div');

Code language: JavaScript (javascript)

And add an HTML snippet to the div:

div.innerHTML = '**<p>**CreateElement example**</p>**';

Code language: HTML, XML (xml)

To attach the div to the document, you use the appendChild() method:

document.body.appendChild(div);

Code language: CSS (css)

Put it all together:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JS CreateElement Demo**</title>**

**</head>**

**<body>**

**<script>**

let div = document.createElement('div');

div.id = 'content';

div.innerHTML = '<p>CreateElement example</p>';

document.body.appendChild(div);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

### **Adding an id to the div**

If you want to add an id to a div, you set the id attribute of the element to a value, like this:

let div = document.createElement('div');

div.id = 'content';

div.innerHTML = '<p>CreateElement example</p>';

document.body.appendChild(div);

Code language: JavaScript (javascript)

### **Adding a class to the div**

The following example set the CSS class of a new div note:

let div = document.createElement('div');

div.id = 'content';

div.className = 'note';

div.innerHTML = '<p>CreateElement example</p>';

document.body.appendChild(div);

Code language: JavaScript (javascript)

### **Adding text to a div**

To add a piece of text to a <div>, you can use the innerHTML property as the above example, or create a new Text node and append it to the div:

*// create a new div and set its attributes*

let div = document.createElement('div');

div.id = 'content';

div.className = 'note';

*// create a new text node and add it to the div*

let text = document.createTextNode('CreateElement example');

div.appendChild(text);

*// add div to the document*

document.body.appendChild(div);

Code language: JavaScript (javascript)

### **Adding an element to a div**

To add an element to a div, you create an element and append it to the div using the appendChild() method:

let div = document.createElement('div');

div.id = 'content';

div.className = 'note';

*// create a new heading and add it to the div*

let h2 = document.createElement('h2');

h2.textContent = 'Add h2 element to the div';

div.appendChild(h2);

*// add div to the document*

document.body.appendChild(div);

Code language: JavaScript (javascript)

## **2) Creating new list items (li) example**

Let’s say you have a list of items:

**<ul id="menu">**

**<li>**Home**</li>**

**</ul>**

Code language: HTML, XML (xml)

The following code adds two li elements to the ul:

let li = document.createElement('li');

li.textContent = 'Products';

menu.appendChild(li);

li = document.createElement('li');

li.textContent = 'About Us';

*// select the ul menu element*

const menu = document.querySelector('#menu');

menu.appendChild(li);

Code language: JavaScript (javascript)

Output:

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**Products**</li>**

**<li>**About Us**</li>**

**</ul>**

Code language: HTML, XML (xml)

## **3) Creating a script element example**

Sometimes, you may want to load a JavaScript file dynamically. To do this, you can use the document.createElement() to create the script element and add it to the document.

The following example illustrates how to create a new script element and loads the /lib.js file to the document:

let script = document.createElement('script');

script.src = '/lib.js';

document.body.appendChild(script);

Code language: JavaScript (javascript)

You can first create a new helper function that loads a JavaScript file from an URL:

function loadJS(url) {

let script = document.createElement('script');

script.src = url;

document.body.appendChild(script);

}

Code language: JavaScript (javascript)

And then use the helper function to load the /lib.js file:

loadJS('/lib.js');

Code language: JavaScript (javascript)

To load a JavaScript file asynchronously, you set the async attribute of the script element to true:

function loadJSAsync(url) {

let script = document.createElement('script');

script.src = url;

script.async = true;

document.body.appendChild(script);

}

Code language: JavaScript (javascript)

## **Summary**

* The document.createElement() creates a new HTML element.
* The element.appendChild() appends an HTML element to an existing element.

# JavaScript appendChild

**Summary**: in this tutorial, you will learn how to use the JavaScript appendChild() method to add a node to the end of the list of child nodes of a specified parent node.

## **Introduction to the JavaScript appendChild() method**

The appendChild() is a method of the Node interface. The appendChild() method allows you to add a node to the end of the list of child nodes of a specified parent node.

The following illustrates the syntax of the appendChild() method:

parentNode.appendChild(childNode);

Code language: CSS (css)

In this method, the childNode is the node to append to the given parent node. The appendChild() returns the appended child.

If the childNode is a reference to an existing node in the document, the appendChild() method moves the childNode from its current position to the new position.

## **JavaScript appendChild() examples**

Let’s take some examples of using the appendChild() method.

### **1) Simple appendChild() example**

Suppose that you have the following HTML markup:

**<ul id="menu">**

**</ul>**

Code language: HTML, XML (xml)

The following example uses the appendChild() method to add three list items to the <ul> element:

function createMenuItem(name) {

let li = document.createElement('li');

li.textContent = name;

return li;

}

*// get the ul#menu*

const menu = document.querySelector('#menu');

*// add menu item*

menu.appendChild(createMenuItem('Home'));

menu.appendChild(createMenuItem('Services'));

menu.appendChild(createMenuItem('About Us'));

Code language: JavaScript (javascript)

How it works:

* First, the createMenuItem() function create a new list item element with a specified name by using the [createElement()](https://www.javascripttutorial.net/javascript-dom/javascript-createelement/) method.
* Second, select the <ul> element with id menu using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method.
* Third, call the createMenuItem() function to create a new menu item and use the appendChild() method to append the menu item to the <ul> element

Output:

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**Services**</li>**

**<li>**About Us**</li>**

**</ul>**

Code language: HTML, XML (xml)

Put it all together:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript appendChild() Demo**</title>**

**</head>**

**<body>**

**<ul id="menu">**

**</ul>**

**<script>**

function createMenuItem(name) {

let li = document.createElement('li');

li.textContent = name;

return li;

}

*// get the ul#menu*

const menu = document.querySelector('#menu');

*// add menu item*

menu.appendChild(createMenuItem('Home'));

menu.appendChild(createMenuItem('Services'));

menu.appendChild(createMenuItem('About Us'));

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

### **2) Moving a node within the document example**

Assuming that you have two lists of items:

**<ul id="first-list">**

**<li>**Everest**</li>**

**<li>**Fuji**</li>**

**<li>**Kilimanjaro**</li>**

**</ul>**

**<ul id="second-list">**

**<li>**Karakoram Range**</li>**

**<li>**Denali**</li>**

**<li>**Mont Blanc**</li>**

**</ul>**

Code language: HTML, XML (xml)

The following example uses the appendChild() to move the first child element from the first list to the second list:

*// get the first list*

const firstList = document.querySelector('#first-list');

*// take the first child element*

const everest = firstList.firstElementChild;

*// get the second list*

const secondList = document.querySelector('#second-list');

*// append the everest to the second list*

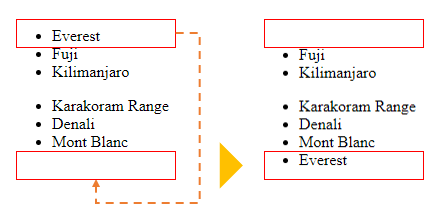
secondList.appendChild(everest)

Code language: JavaScript (javascript)

How it works:

* First, select the first element by its id (first-list) using the querySelector() method.
* Second, select the first child element from the first list.
* Third, select the second element by its id (second-list) using the querySelector() method.
* Finally, append the first child element from the first list to the second list using the appendChild() method.

Here are the list before and after moving:



## **Summary**

* Use appendChild() method to add a node to the end of the list of child nodes of a specified parent node.
* The appendChild() can be used to move an existing child node to the new position within the document.

# JavaScript textContent

**Summary**: in this tutorial, you will learn how to use the JavaScript textContent property to get the text content of a node and its descendants.

## **Reading textContent from a node**

To get the text content of a node and its descendants, you use the textContent property:

let text = node.textContent;

Code language: JavaScript (javascript)

Suppose that you have the following HTML snippet:

**<div id="note">**

JavaScript textContent Demo!

**<span style="display:none">**Hidden Text!**</span>**

*<!-- my comment -->*

**</div>**

Code language: HTML, XML (xml)

The following example uses the textContent property to get the text of the <div> element:

let note = document.getElementById('note');

console.log(note.textContent);

Code language: JavaScript (javascript)

How it works.

* First, select the div element with the id note by using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Then, display the text of the node by accessing the textContent property.

Output:

JavaScript textContent Demo!

Hidden Text!

As you can see clearly from the output, the textContent property returns the concatenation of the textContent of every [child node](https://www.javascripttutorial.net/javascript-dom/javascript-get-child-element/), excluding comments (and also processing instructions).

### **textContent vs. innerText**

On the other hand, the innerText takes the CSS style into account and returns only human-readable text. For example:

let note = document.getElementById('note');

console.log(note.innerText);

Code language: JavaScript (javascript)

Output:

JavaScript textContent Demo!

As you can see, the hidden text and comments are not returned.

Since the innerText property uses the up-to-date CSS to compute the text, accessing it will trigger a reflow, which is computationally expensive.

A **reflow** occurs when a web brower needs to process and draw parts or all of a webpage again.

## **Setting textContent for a node**

Besides reading textContent, you can also use the textContent property to set the text for a node:

node.textContent = newText;

When you set textContent on a node, all the node’s children will be removed and replaced by a single text node with the newText value. For example:

let note = document.getElementById('note');

note.textContent = 'This is a note';

Code language: JavaScript (javascript)

## **Summary**

* Use the textContent property to return the concatenation of the textContent of every child node. You can use it to set a text for a node.
* The innerText returns the human-readable text that takes CSS into account.

# JavaScript innerHTML

**Summary**: in this tutorial, you will learn how to use the JavaScript innerHTML property of an element to get or set an HTML markup contained in the element.

The innerHTML is a property of the Element that allows you to get or set the HTML markup contained within the element.

## **Reading the innerHTML property of an element**

To get the HTML markup contained within an element, you use the following syntax:

let content = element.innerHTML;

Code language: JavaScript (javascript)

When you read the innerHTML of an element, the web browser has to serialize the HTML fragment of the element’s descendants.

### **1) Simple innerHTML example**

Suppose that you have the following markup:

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**Services**</li>**

**</ul>**

Code language: HTML, XML (xml)

The following example uses the innerHTML property to get the content of the <ul> element:

let menu = document.getElementById('menu');

console.log(menu.innerHTML);

Code language: JavaScript (javascript)

How it works:

* First, select the <ul> element by its id (menu) using the getElementById() method.
* Then, get the HTML content of the <ul> element using the innerHTML.

Output:

**<li>**Home**</li>**

**<li>**Services**</li>**

Code language: HTML, XML (xml)

### **2) Examining the current HTML source**

The innerHTML property returns the current HTML source of the document, including all changes have been made since the page was loaded.

See the following example:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript innerHTML**</title>**

**</head>**

**<body>**

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**Services**</li>**

**</ul>**

**<script>**

let menu = document.getElementById('menu');

*// create new li element*

let li = document.createElement('li');

li.textContent = 'About Us';

*// add it to the ul element*

menu.appendChild(li);

console.log(menu.innerHTML);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Output:

**<li>**Home**</li>**

**<li>**Services**</li>**

**<li>**About Us**</li>**

Code language: HTML, XML (xml)

How it works.

* First, get the <ul> element with the id menu using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Second, create a new <li> element and add it to the <ul> element using the [createElement()](https://www.javascripttutorial.net/javascript-dom/javascript-createelement/) and appendChild() methods.
* Third, get the HTML of the <ul> element using the innerHTML property of the <ul> element. The contents of the <ul> element includes the initial content and the dynamic content created by JavaScript.

## **Setting the innerHTML property of an element**

To set the value of innerHTML property, you use this syntax:

element.innerHTML = newHTML;

The setting will replace the existing content of an element with the new content.

For example, you can remove the entire contents of the document by clearing contents of the document.body element:

document.body.innerHTML = '';

Code language: JavaScript (javascript)

### **⚠️ Security Risk**

HTML5 specifies that a <script> tag inserted with innerHTML should not execute. See the following example:

index.html document:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript innerHTML**</title>**

**</head>**

**<body>**

**<div id="main"></div>**

**<script src="app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

app.js

const main = document.getElementById('main');

const scriptHTML = '<script>alert("Alert from innerHTML");</script>';

main.innerHTML = scriptHTML;

Code language: JavaScript (javascript)

In this example, the alert() inside the <script> tag will not execute.

However, if you change source code of the app.js to the following:

const main = document.getElementById('main');

const externalHTML = `<img src='1' onerror='alert("Error loading image")'>`;

*// shows the alert*

main.innerHTML = externalHTML;

Code language: JavaScript (javascript)

The [alert()](https://www.javascripttutorial.net/javascript-bom/javascript-alert/) will show because the image cannot be loaded successfully. It causes the onerror handler executes. And this handler can execute any malicious code, not just a simple alert.

Therefore, you should not set the innerHTML to the content that you have no control over or you will face a potential security risk.

Because of this, if you want to insert plain text into the document, you use the [textContent](https://www.javascripttutorial.net/javascript-dom/javascript-textcontent/) property instead of the innerHTML. The textContent will not be parsed as the HTML, but as the raw text.

## **Summary**

* Use innerHTML property of an element to get or set HTML contained within the element.
* The innerHTML property returns the current HTML source of the element, including any change that has been made since the page was loaded.
* Do not use innerHTML to set new content that you have no control over to avoid a security risk.

# JavaScript innerHTML vs createElement

**Summary**: in this tutorial, you’ll learn the difference between the innerHTML and createElement() when it comes to creating new elements in the DOM tree.

## **#1) createElement is more performant**

Suppose that you have a div element with the class container:

**<div class="container"></div>**

Code language: HTML, XML (xml)

You can new elements to the div element by creating an element and appending it:

let div = document.querySelector('.container');

let p = document.createElement('p');

p.textContent = 'JS DOM';

div.appendChild(p);

Code language: JavaScript (javascript)

You can also manipulate an element’s HTML directly using innerHTML like this:

let div = document.querySelector('.container');

div.innerHTML += '<p>JS DOM</p>';

Code language: JavaScript (javascript)

Using innerHTML is cleaner and shorter when you want to add attributes to the element:

let div = document.querySelector('.container');

div.innerHTML += '<p class="note">JS DOM</p>';

Code language: JavaScript (javascript)

However, using the innerHTML causes the web browsers to reparse and recreate all DOM nodes inside the div element. Therefore, it is less efficient than creating a new element and appending to the div. In other words, creating a new element and appending it to the DOM tree provides better performance than the innerHTML.

## **#2) createElement is more secure**

As mentioned in the innerHTML tutorial, you should use it only when the data comes from a trusted source like a database.

If you set the contents that you have no control over to the innerHTML, the malicious code may be injected and executed.

## **#3) Using DocumentFragment for composing DOM Nodes**

Assuming that you have a list of elements and you need in each iteration:

let div = document.querySelector('.container');

for (let i = 0; i < 1000; i++) {

let p = document.createElement('p');

p.textContent = `Paragraph ${i}`;

div.appendChild(p);

}

Code language: JavaScript (javascript)

This code results in recalculation of styles, painting, and layout every iteration. This is not very efficient.

To overcome this, you typically use a DocumentFragment to compose DOM nodes and append it to the DOM tree:

let div = document.querySelector('.container');

*// compose DOM nodes*

let fragment = document.createDocumentFragment();

for (let i = 0; i < 1000; i++) {

let p = document.createElement('p');

p.textContent = `Paragraph ${i}`;

fragment.appendChild(p);

}

*// append the fragment to the DOM tree*

div.appendChild(fragment);

Code language: JavaScript (javascript)

In this example, we composed the DOM nodes by using the DocumentFragment object and append the fragment to the active DOM tree once at the end.

A document fragment does not link to the active DOM tree, therefore, it doesn’t incur any performance.

Check it out the [DocumentFragment tutorial](https://www.javascripttutorial.net/javascript-dom/javascript-documentfragment/) for more detail.

# JavaScript DocumentFragment

**Summary**: in this tutorial, you’ll learn about the JavaScript DocumentFragment interface to compose DOM nodes and update them to the active DOM tree.

## **Introduction to the JavaScript DocumentFragment interface**

The DocumentFragment interface is a lightweight version of the Document that stores a piece of document structure like a standard document. However, a DocumentFragment isn’t part of the active DOM tree.

If you make changes to the document fragment, it doesn’t affect the document or incurs any performance.

Typically, you use the DocumentFragment to compose DOM nodes and append or insert it to the active DOM tree using [appendChild()](https://www.javascripttutorial.net/javascript-dom/javascript-appendchild/) or [insertBefore()](https://www.javascripttutorial.net/javascript-dom/javascript-insertbefore/) method.

To create a new document fragment, you use the DocumentFragment constructor like this:

let fragment = new DocumentFragment();

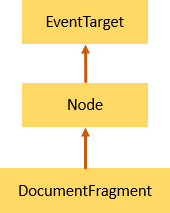
Code language: JavaScript (javascript)

Or you can use the createDocumentFragment() method of the Document object:

let fragment = document.createDocumentFragment();

Code language: JavaScript (javascript)

This DocumentFragment inherits the methods of its parent, Node, and also implements those of the ParentNode interface such as [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) and querySelectorAll().



## **JavaScript DocumentFragment example**

Suppose that you have a <ul> element with the id language:

**<ul id="language"></ul>**

Code language: HTML, XML (xml)

The following code creates a list of <li> elements (<li>) and append each to the <ul> element using the DocumentFragment:

let languages = ['JS', 'TypeScript', 'Elm', 'Dart','Scala'];

let langEl = document.querySelector('#language')

let fragment = new DocumentFragment();

languages.forEach((language) => {

let li = document.createElement('li');

li.innerHTML = language;

fragment.appendChild(li);

})

langEl.appendChild(fragment);

Code language: JavaScript (javascript)

How it works:

* First, select the <ul> element by its id using the querySelector() method.
* Second, create a new document fragment.
* Third, for each element in the languages array, create a list item element, assign the list item’s innerHTML to the language, and append all the newly created list items to the document fragment.
* Finally, append the document fragment to the <ul> element.

Put it all together:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**DocumentFragment Demo**</title>**

**</head>**

**<body>**

**<ul id="language"></ul>**

**<script>**

let languages = ['JS', 'TypeScript', 'Elm', 'Dart', 'Scala'];

let langEl = document.querySelector('#language');

let fragment = new DocumentFragment();

languages.forEach((language) => {

let li = document.createElement('li');

li.innerHTML = language;

fragment.appendChild(li);

})

langEl.appendChild(fragment);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **Summary**

* Use the DocumentFragment to compose DOM nodes before updating them to the active DOM tree to get better performance.

# JavaScript insertBefore

**Summary**: in this tutorial, you will learn how to use the JavaScript insertBefore() to insert a node before another node as a child node of a specified parent node.

## **Introduction to JavaScript insertBefore() method**

To insert a node before another node as a child node of a parent node, you use the parentNode.insertBefore() method:

parentNode.insertBefore(newNode, existingNode);

Code language: CSS (css)

In this method:

* The newNode is the new node to be inserted.
* The existingNode is the node before which the new node is inserted. If the existingNode is null, the insertBefore() inserts the newNode at the end of the parentNode‘s child nodes.

The insertBefore() returns the inserted child node.

## **JavaScript insertBefore() helper function**

The following insertBefore() function inserts a new node before a specified node:

function insertBefore(newNode, existingNode) {

existingNode.parentNode.insertBefore(newNode, existingNode);

}

Code language: JavaScript (javascript)

## **JavaScript insertBefore() example**

Suppose that you have the following list of items:

**<ul id="menu">**

**<li>**Services**</li>**

**<li>**About**</li>**

**<li>**Contact**</li>**

**</ul>**

Code language: HTML, XML (xml)

The following example uses the insertBefore() method to insert a new node as the first list item:

let menu = document.getElementById('menu');

*// create a new li node*

let li = document.createElement('li');

li.textContent = 'Home';

*// insert a new node before the first list item*

menu.insertBefore(li, menu.firstElementChild);

Code language: JavaScript (javascript)

How it works.

* First, get the menu element using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Second, create a new list item using the [createElement()](https://www.javascripttutorial.net/javascript-dom/javascript-createelement/) method.
* Third, insert the list item element before the first child element of the menu element using the insertBefore() method.

Put it all together.

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript insertBefore()**</title>**

**</head>**

**<body>**

**<ul id="menu">**

**<li>**Services**</li>**

**<li>**About**</li>**

**<li>**Contact**</li>**

**</ul>**

**<script>**

let menu = document.getElementById('menu');

*// create a new li node*

let li = document.createElement('li');

li.textContent = 'Home';

*// insert a new node before the first list item*

menu.insertBefore(li, menu.firstElementChild);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **Summary**

* Use the parentNode.insertBefore() to insert a new node before an existing node as a child node of a parent node.

# JavaScript insertAfter

**Summary**: in this tutorial, you will learn how to insert a new node after an existing node as a child node of a parent node.

JavaScript DOM provides the [insertBefore()](https://www.javascripttutorial.net/javascript-dom/javascript-insertbefore/) method that allows you to insert a new node after an existing node as a child node. However, it hasn’t supported the insertAfter() method yet.

To insert a new node after an existing node as a child node, you can use the following approach:

* First, [select the next sibling node](https://www.javascripttutorial.net/javascript-dom/javascript-get-parent-element-parentnode/) of the existing node.
* Then, [select the parent node](https://www.javascripttutorial.net/javascript-dom/javascript-get-parent-element-parentnode/) of the existing node and call the [insertBefore()](https://www.javascripttutorial.net/javascript-dom/javascript-insertbefore/) method on the parent node to insert a new node before that immediate sibling node.

The following insertAfter() function illustrates the logic:

function insertAfter(newNode, existingNode) {

existingNode.parentNode.insertBefore(newNode, existingNode.nextSibling);

}

Code language: JavaScript (javascript)

Suppose that you have the following list of items:

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**About**</li>**

**<li>**Contact**</li>**

**</ul>**

Code language: HTML, XML (xml)

The following inserts a new node after the last list item:

let menu = document.getElementById('menu');

*// create a new li node*

let li = document.createElement('li');

li.textContent = 'Services';

*// insert a new node after the last list item*

insertAfter(li, menu.lastElementChild);

Code language: JavaScript (javascript)

How it works:

* First, select the ul element by its id (menu) using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Second, create a new list item using the [createElement()](https://www.javascripttutorial.net/javascript-dom/javascript-createelement/) method.
* Third, use the insertAfter() method to insert a list item element after the last list item element.

Put it all together.

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript insertAfter() Demo**</title>**

**</head>**

**<body>**

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**About**</li>**

**<li>**Contact**</li>**

**</ul>**

**<script>**

function insertAfter(newNode, existingNode) {

existingNode.parentNode.insertBefore(newNode, existingNode.nextSibling);

}

let menu = document.getElementById('menu');

*// create a new li node*

let li = document.createElement('li');

li.textContent = 'Services';

insertAfter(li, menu.lastElementChild);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

The menu looks like the following after the insert:

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**About**</li>**

**<li>**Contact**</li>**

**<li>**Services**</li>**

**</ul>**

Code language: HTML, XML (xml)

## **Summary**

* JavaScript DOM hasn’t supported the insertAfter() method yet.
* Use the insertBefore() method and the nextSibling property to insert a new before an existing node as a child of a parent node.

# JavaScript append

**Summary**: in this tutorial, you’ll learn how to use the JavaScript append() method to insert a set of Node objects or DOMString objects after the last child of a parent node.

## **Introduction to JavaScript append() method**

The parentNode.append() method inserts a set of Node objects or DOMString objects after the last child of a [parent node](https://www.javascripttutorial.net/javascript-dom/javascript-get-parent-element-parentnode/):

parentNode.append(...nodes);

parentNode.append(...DOMStrings);

Code language: JavaScript (javascript)

The append() method will insert DOMString objects as Text nodes.

Note that a DOMString is a UTF-16 string that maps directly to a string.

The append() method has no return value. It means that the append() method implicitly returns undefined.

## **JavaScript append() method examples**

Let’s take some examples of using the append() method.

### **1) Using the append() method to append an element example**

Suppose that you have the following ul element:

**<ul id="app">**

**<li>**JavaScript**</li>**

**</ul>**

Code language: HTML, XML (xml)

The following example shows how to create a list of li elements and append them to the ul element:

let app = document.querySelector('#app');

let langs = ['TypeScript','HTML','CSS'];

let nodes = langs.map(lang => {

let li = document.createElement('li');

li.textContent = lang;

return li;

});

app.append(...nodes);

Code language: JavaScript (javascript)

Output HTML:

**<ul id="app">**

**<li>**JavaScript**</li>**

**<li>**TypeScript**</li>**

**<li>**HTML**</li>**

**<li>**CSS**</li>**

**</ul>**

Code language: HTML, XML (xml)

How it works:

* First, select the ul element by its id by using the querySelector() method.
* Second, declare an array of languages.
* Third, for each language, create a new li element with the textContent is assigned to the language.
* Finally, append li elements to the ul element by using the append() method.

### **2) Using the append() method to append text to an element example**

Assume that you have the following HTML:

**<div id="app"></div>**

Code language: HTML, XML (xml)

You can use the append() method to append a text to an element:

let app = document.querySelector('#app');

app.append('append() Text Demo');

console.log(app.textContent);

Code language: JavaScript (javascript)

Output HTML:

**<div id="app">**append() Text Demo**</div>**

Code language: HTML, XML (xml)

## **append vs. appendChild()**

Here are the differences between append() and appendChild():

|  |  |  |
| --- | --- | --- |
| Differences | append() | appendChild() |
| Return value | undefined | The appended Node object |
| Input | Multiple Node Objects | A single Node object |
| Parameter Types | Accept Node and DOMString | Only Node |

## **Summary**

* Use the parentNode.append() method to append a set of Node objects or DOMString objects after the last child node of the parentNode.

# JavaScript prepend

**Summary**: in this tutorial, you’ll learn about the JavaScript prepend() method that inserts Node objects or DOMString objects before the first child of a parent node.

## **Introduction to JavaScript prepend() method**

The prepend() method inserts a set of Node objects or DOMString objects after the first child of a parent node:

parentNode.prepend(...nodes);

parentNode.prepend(...DOMStrings);

Code language: JavaScript (javascript)

The prepend() method inserts DOMString objects as Text nodes. Note that a DOMString is a UTF-16 string that directly maps to a string.

The prepend() method returns undefined.

## **JavaScript prepend() method examples**

Let’s take some examples of using the prepend() method.

### **1) Using the prepend() method to prepend an element example**

Let’s say you have the following ul element:

**<ul id="app">**

**<li>**HTML**</li>**

**</ul>**

Code language: HTML, XML (xml)

This example shows how to create a list of li elements and prepend them to the ul element:

let app = document.querySelector('#app');

let langs = ['CSS','JavaScript','TypeScript'];

let nodes = langs.map(lang => {

let li = document.createElement('li');

li.textContent = lang;

return li;

});

app.prepend(...nodes);

Code language: JavaScript (javascript)

Output HTML:

**<ul id="app">**

**<li>**CSS**</li>**

**<li>**JavaScript**</li>**

**<li>**TypeScript**</li>**

**<li>**HTML**</li>**

**</ul>**

Code language: HTML, XML (xml)

How it works:

* First, select the ul element by its id by using the querySelector() method.
* Second, declare an array of strings.
* Third, for each element in an array, create a new li element with the textContent is assigned to the array element.
* Finally, prepend the li elements to the ul parent element by using the prepend() method.

### **2) Using the prepend() method to prepend text to an element example**

Suppose that you have the following element:

**<div id="app"></div>**

Code language: HTML, XML (xml)

The following shows how to use the prepend() method to prepend a text to the above div element:

let app = document.querySelector('#app');

app.prepend('prepend() Text Demo');

console.log(app.textContent);

Code language: JavaScript (javascript)

Output HTML:

**<div id="app">**prepend() Text Demo**</div>**

Code language: HTML, XML (xml)

## **Summary**

* Use the parentNode.prepend() method to prepend a list of Node objects or DOMString objects before the first child node of the parent node.

# JavaScript insertAdjacentHTML

**Summary**: in this tutorial, you’ll learn how to use the insertAdjacentHTML() method to insert HTML into the document.

## **Introduction to JavaScript insertAdjacentHTML() method**

The insertAdjacentHTML() is a method of the Element interface so that you can invoke it from any element.

The insertAdjacentHTML() method parses a piece of HTML text and inserts the resulting nodes into the DOM tree at a specified position:

element.insertAdjacentHTML(positionName, text);

Code language: JavaScript (javascript)

The insertAdjacentHTML() method has two parameters:

### **1) position**

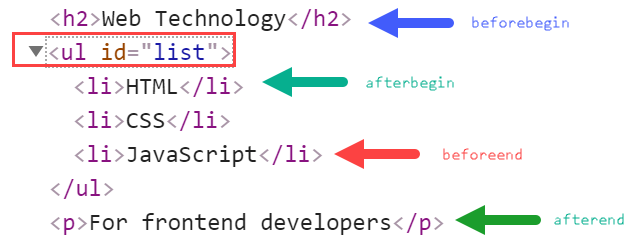
The positionName is a string that represents the position relative to the element. The positionName accepts one of the following four values:

* 'beforebegin': before the element
* 'afterbegin': before its first child of the element.
* 'beforeend': after the last child of the element
* 'afterend': after the element

Note that the 'beforebegin' and 'afterend' are only relevant if the element is in the DOM tree and has a parent element.

The insertAdjacentHTML() method has no return value, or undefined by default.

The following visualization illustrates the position name:



### **2) text**

The text parameter is a string that the insertAdjacentHTML() method parses as HTML or XML. It cannot be Node objects

### **Security consideration**

Like the [innerHTML](https://www.javascripttutorial.net/javascript-dom/javascript-innerhtml/), if you use the user input to pass into the insertAdjacentHTML() method, you should always escape it to avoid security risk.

## **JavaScript insertAdjacentHTML() method example**

The following JavaScript example uses the insertAdjacentHTML() method to insert various elements into the page with the positions relative to the ul element:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**insertAdjacentHTML() Demo**</title>**

**</head>**

**<body>**

**<ul id="list">**

**<li>**CSS**</li>**

**</ul>**

**<script>**

let list = document.querySelector('#list');

list.insertAdjacentHTML('beforebegin', '<h2>Web Technology</h2>');

list.insertAdjacentHTML('afterbegin', '<li>HTML</li>');

list.insertAdjacentHTML('beforeend', '<li>JavaScript</li>');

list.insertAdjacentHTML('afterend', '<p>For frontend developers</p>');

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, select the ul element by its id list using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method.
* Next, use the insertAdjacentHTML() method to insert a heading 2 element before the ul element. The position is 'beforebegin'.
* Then, use the insertAdjacentHTML() method to insert a new list item element before the first child of the ul element. The position is 'afterbegin'.
* After that, use the insertAdjacentHTML() method to insert a new list item element after the last child of the ul element with the position 'beforeend'.
* Finally, insert use the insertAdjacentHTML() method to insert a new paragraph element after the ul element with the position 'afterend'.

## **Summary**

* Use the insertAdjacentHTML() method to insert a text as Nodes into the DOM tree at a specified position.
* Always escape the user input text that you pass into the insertAdjacentHTML() method to avoid security risk.

# JavaScript replaceChild

**Summary**: in this tutorial, you will learn how to use the JavaScript Node.replaceChild() method to replace an HTML element by a new one.

To replace an HTML element, you use the node.replaceChild() method:

parentNode.replaceChild(newChild, oldChild);

Code language: CSS (css)

In this method, the newChild is the new node to replace the oldChild node which is the old child node to be replaced.

Suppose that you have the following list of items:

**<ul id="menu">**

**<li>**Homepage**</li>**

**<li>**Services**</li>**

**<li>**About**</li>**

**<li>**Contact**</li>**

**</ul>**

Code language: HTML, XML (xml)

The following example creates a new list item element and replaces the first list item element in the menu by the new one:

let menu = document.getElementById('menu');

*// create a new node*

let li = document.createElement('li');

li.textContent = 'Home';

*// replace the first list item*

menu.replaceChild(li, menu.firstElementChild);

Code language: JavaScript (javascript)

Put it all together.

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript DOM: Replace Elements**</title>**

**</head>**

**<body>**

**<ul id="menu">**

**<li>**Homepage**</li>**

**<li>**Services**</li>**

**<li>**About**</li>**

**<li>**Contact**</li>**

**</ul>**

**<script>**

let menu = document.getElementById('menu');

*// create a new node*

let li = document.createElement('li');

li.textContent = 'Home';

*// replace the first list item*

menu.replaceChild(li, menu.firstElementChild);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **Summary**

* Use Node.replaceChild() to replace a child element of a node by a new element.

# JavaScript cloneNode

**Summary**: in this tutorial, you will learn how to use the JavaScript cloneNode() method to clone an element.

The cloneNode() is a method of the Node interface that allows you to clone an element:

let clonedNode = originalNode.cloneNode(deep);

Code language: JavaScript (javascript)

### **Pamaraters**

### **deep**

The cloneNode() method accepts an optional parameter deep:

* If the deep is true, then the original node and all of its descendants are cloned.
* If the deep is false, only the original node will be cloned. All the node’s descendants will not be cloned.

The deep parameter defaults to false if you omit it.

### **originalNode**

The originalNode is the element to be cloned.

### **Return value**

The cloneNode() returns a copy of the originalNode.

### **Usage notes**

Besides the DOM structure, the cloneNode() copies all attributes and inline listeners of the original node. However, it doesn’t copy the event listeners added via addEventListener() or assignment to element’s properties such as originalNode.onclick = eventHandler().

If you clone a node with an id attribute and place the cloned node in the same document, the id will be duplicate. In this case, you need to change the id before adding it to the DOM tree.

## **JavaScript cloneNode() example**

The following example uses the cloneNode() method to copy the <ul> element and place it in the same document:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript cloneNode() Demo**</title>**

**</head>**

**<body>**

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**Services**</li>**

**<li>**About**</li>**

**<li>**Contact**</li>**

**</ul>**

**<script>**

let menu = document.querySelector('#menu');

let clonedMenu = menu.cloneNode(true);

clonedMenu.id = 'menu-mobile';

document.body.appendChild(clonedMenu);

**</script>**

**</body>**

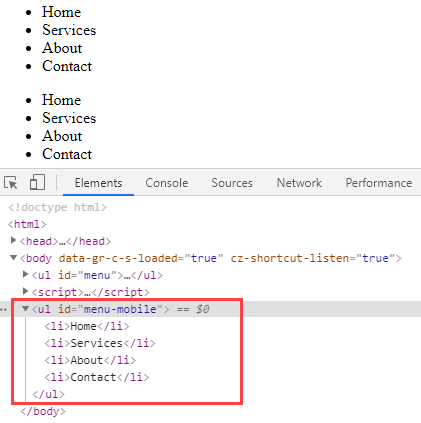
**</html>**

Code language: HTML, XML (xml)

How it works.

* First, select the <ul> with the id menu by using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method.
* Second, create a deep clone of the <ul> element using the cloneNode() method.
* Third, change the id of the cloned element to avoid duplicate.
* Finally, append the cloned element to the child nodes of the document.body using the [appendChild()](https://www.javascripttutorial.net/javascript-dom/javascript-appendchild/) method.

Output:



## **Summary**

* Use the node.cloneNode() method to clone the node.
* Pass true into the cloneNode() method to create a deep clone of a DOM element.

# JavaScript removeChild

**Summary**: in this tutorial, you will learn how to use the JavaScript removeChild() method to remove a child node from a parent node.

## **Introduction to JavaScript removeChild() method**

To remove a child element of a node, you use the removeChild() method:

let childNode = parentNode.removeChild(childNode);

Code language: JavaScript (javascript)

The childNode is the child node of the parentNode that you want to remove. If the childNode is not the child node of the parentNode, the method throws an exception.

The removeChild() returns the removed child node from the DOM tree but keeps it in the memory, which can be used later.

If you don’t want to keep the removed child node in the memory, you use the following syntax:

parentNode.removeChild(childNode);

Code language: CSS (css)

The child node will be in the memory until it is destroyed by the JavaScript garbage collector.

## **JavaScript removeChild() example**

Suppose you have the following list of items:

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**Products**</li>**

**<li>**About Us**</li>**

**</ul>**

Code language: HTML, XML (xml)

The following example uses the removeChild() to remove the last list item:

let menu = document.getElementById('menu');

menu.removeChild(menu.lastElementChild);

Code language: JavaScript (javascript)

How it works:

* First, get the ul element with the id menu by using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Then, remove the last element of the ul element by using the removeChild() method. The [menu.lastElementChild](https://www.javascripttutorial.net/javascript-dom/javascript-get-child-element/) property returns the last child element of the menu.

Put it all together.

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JavaScript removeChild()**</title>**

**</head>**

**<body>**

**<ul id="menu">**

**<li>**Home**</li>**

**<li>**Products**</li>**

**<li>**About Us**</li>**

**</ul>**

**<script>**

let menu = document.getElementById('menu');

menu.removeChild(menu.lastElementChild);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **Removing all child nodes of an element**

To remove all child nodes of an element, you use the following steps:

* Get the first node of the element using the firstChild property.
* Repeatedly removing the child node until there are no child nodes left.

The following code shows how to remove all list items of the menu element:

let menu = document.getElementById('menu');

while (menu.firstChild) {

menu.removeChild(menu.firstChild);

}

Code language: JavaScript (javascript)

You can remove all child nodes of an element by setting the innerHTML property of the parent node to blank:

let menu = document.getElementById('menu');

menu.innerHTML = '';

Code language: JavaScript (javascript)

## **Summary**

* Use parentNode.removeChild() to remove a child node of a parent node.
* The parentNode.removeChild() throws an exception if the child node cannot be found in the parent node.

# Understanding Relationships Between HTML Attributes & DOM Object’s Properties

**Summary**: in this tutorial, you will learn the relationships between HTML attributes and DOM object’s properties.

When the web browser [loads an HTML page](https://www.javascripttutorial.net/javascript-dom/javascript-page-load-events/), it generates the corresponding DOM objects based on the DOM nodes of the document.

For example, if a page contains the following input element:

<input type="text" id="username">

Code language: JavaScript (javascript)

The web browser will generate an HTMLInputElement object.

The input element has two attributes:

* The type attribute with the value text.
* The id attribute with the value username.

The generated HTMLInputElement object will have the corresponding properties:

* The input.type with the value text.
* The input.id with the value username.

In other words, the web browser will automatically convert attributes of HTML elements to properties of DOM objects.

However, the web browser only converts the standard attributes to the DOM object’s properties. The standard attributes of an element are listed on the element’s specification.

Attribute-property mapping is not always one-to-one. For example:

<input type="text" id="username" secured="true">

Code language: JavaScript (javascript)

In this example, the secured is a non-standard attribute:

let input = document.querySelector('#username');

console.log(input.secured); *// undefined*

Code language: JavaScript (javascript)

## **Attribute methods**

To access both standard and non-standard attributes, you use the following methods:

* element.getAttribute(name) – [get the attribute value](https://www.javascripttutorial.net/javascript-dom/javascript-getattribute/)
* element.setAttribute(name, value) – [set the value for the attribute](https://www.javascripttutorial.net/javascript-dom/javascript-setattribute/)
* element.hasAttribute(name) – [check for the existence of an attribute](https://www.javascripttutorial.net/javascript-dom/javascript-hasattribute/)
* element.removeAttribute(name) – [remove the attribute](https://www.javascripttutorial.net/javascript-dom/javascript-removeattribute/)

## **element.attributes**

The element.attributes property provides a live collection of attributes available on a specific element. For example:

let input = document.querySelector('#username');

for(let attr of input.attributes) {

console.log(`${attr.name} = ${attr.value}` )

}

Code language: JavaScript (javascript)

Output:

type = text

id = username

secure = true

Code language: JavaScript (javascript)

Note that element.attributes is a NamedNodeMap, not an [Array](https://www.javascripttutorial.net/javascript-array/), therefore, it has no Array’s methods.

## **Attribute-property synchronization**

When a standard attribute changes, the corresponding property is auto-updated with some exceptions and vice versa.

Suppose that you have the following input element:

<input type="text" id="username" tabindex="1">

Code language: JavaScript (javascript)

The following example illustrates the change of the tabindex attribute is propagated to the tabIndex property and vice versa:

let input = document.querySelector('#username');

*// attribute -> property*

input.setAttribute('tabindex', 2);

console.log(input.tabIndex); *// 2*

*// property -> attribute*

input.tabIndex = 3;

console.log(input.getAttribute('tabIndex')); *// 3*

Code language: JavaScript (javascript)

The following example shows when the value attribute changes, it reflects in the value property, but not the other way around:

let input = document.querySelector('#username');

*// attribute -> property: OK*

input.setAttribute('value','guest');

console.log(input.value); *// guest*

*// property -> attribute: doesn't change*

input.value = 'admin';

console.log(input.getAttribute('value')); *// guest*

Code language: JavaScript (javascript)

## **DOM properties are typed**

The value of an attribute is always a string. However, when the attribute is converted to the property of a DOM object, the property value can be a string, a boolean, an object, etc.

The following checkbox element has the checked attribute. When the checked attribute is converted to the property, it is a boolean value:

<input type="checkbox" id="chkAccept" checked> Accept

let checkbox = document.querySelector('#chkAccept');

console.log(checkbox.checked); *// true*

Code language: JavaScript (javascript)

The following shows an input element with the style attribute:

<input type="password" id="password" style="color:red;with:100%">

Code language: JavaScript (javascript)

The style attribute is a string while the style property is an object:

let input = document.querySelector('#password');

let styleAttr = input.getAttribute('style');

console.log(styleAttr);

console.dir(input.style);

Code language: JavaScript (javascript)

Output:

[object CSSStyleDeclaration]

Code language: JavaScript (javascript)

## **The data-\* attributes**

If you want to add a custom attribute to an element, you should prefix it with the data- e.g., data-secured because all attributes start with data- are reserved for the developer’s uses.

To access data-\* attributes, you can use the dataset property. For example, we have the following div element with custom attributes:

<div id="main" data-progress="pending" data-value="10%"></div>

Code language: JavaScript (javascript)

The following shows how to access the data-\* attributes via the dataset property:

let bar = document.querySelector('#main');

console.log(bar.dataset);

Code language: JavaScript (javascript)

Output:

[object DOMStringMap] {

progress: "pending",

value: "10%"

}

Code language: JavaScript (javascript)

## **Summary**

* Attributes are specified in HTML elements.
* Properties are specified DOM objects.
* Attributes are converted to properties respectively.
* Use the element.attributes property to access standard and custom attributes of an element.
* Use the element.dataset property to access the data-\* attributes.

# JavaScript setAttribute

**Summary**: in this tutorial, you will learn how to use the JavaScript setAttribute() method to set a value for an attribute on a specified element.

## **Introduction to the JavaScript setAttribute() method**

To set a value of an attribute on a specified element, you use the setAttribute() method:

element.setAttribute(name, value);

Code language: CSS (css)

### **Parameters**

The name specifies the attribute name whose value is set. It’s automatically converted to lowercase if you call the setAttribute() on an HTML element.

The value specifies the value to assign to the attribute. It’s automatically converted to a string if you pass a non-string value to the method.

### **Return value**

The setAttribute() returns [undefined](https://www.javascripttutorial.net/javascript-undefined/).

Note that if the attribute already exists on the element, the setAttribute() method updates the value; otherwise, it adds a new attribute with the specified name and value.

Typically, you use the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) or [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) to select an element before calling the setAttribute() on the selected element.

To get the current value of an attribute, you use the getAttribute() method. To remove an attribute, you call the removeAttribute() method.

## **JavaScript setAttribute() example**

See the following example:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JS setAttribute() Demo**</title>**

**</head>**

**<body>**

**<button id="btnSend">**Send**</button>**

**<script>**

let btnSend = document.querySelector('#btnSend');

if (btnSend) {

btnSend.setAttribute('name', 'send');

btnSend.setAttribute('disabled', '');

}

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, select the button with the id btnSend by using the querySelector() method.
* Second, set the value of the name attribute to send using the setAttribute() method.
* Third, set the value of the disabled attribute so that when users click the button, it will do nothing.

Note that the disabled attribute is special because it is a Boolean attribute. If a Boolean attribute is present, whatever value it has, the value is considered to be true. To set the value of a Boolean attribute to false, you need to remove the entire attribute using the removeAttribute() method.

## **Summary**

* Use the setAttribute() to set the value of an attribute on a specified element.
* Setting the value of a Boolean attribute to whatever value, that value will be considered to be true.

# JavaScript getAttribute

**Summary**: in this tutorial, you will learn how to use the JavaScript getAttribute() method to get the value of a specified attribute on an element.

## **Introduction to the JavaScript getAttribute() method**

To get the value of an attribute on a specified element, you call the getAttribute() method of the element:

let value = element.getAttribute(name);

Code language: JavaScript (javascript)

### **Parameters**

The getAttribute() accepts an argument which is the name of the attribute from which you want to return the value.

### **Return value**

If the attribute exists on the element, the getAttribute() returns a string that represents the value of the attribute. In case the attribute does not exist, the getAttribute() returns null.

Note that you can use the hasAttribute() method to check if the attribute exists on the element before getting its value.

## **JavaScript getAttribute() example**

Consider the following example:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JS getAttribute() Demo**</title>**

**</head>**

**<body>**

**<a href="https://www.javascripttutorial.net"**

**target="\_blank"**

**id="js">**JavaScript Tutorial

**</a>**

**<script>**

let link = document.querySelector('#js');

if (link) {

let target = link.getAttribute('target');

console.log(target);

}

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Output

\_blank

How it works:

* First, select the link element with the id js using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method.
* Second, get the target attribute of the link by calling the getAttribute() of the selected link element.
* Third, show the value of the target on the Console window.

The following example uses the getAttribute() method to get the value of the title attribute of the link element with the id js:

let link = document.querySelector('#js');

if (link) {

let title = link.getAttribute('title');

console.log(title);

}

Code language: JavaScript (javascript)

Output:

null

Code language: JavaScript (javascript)

## **Summary**

* Get the value of an attribute of a specified element by calling the getAttribute() method on the element.
* The getAttribute() returns null if the attribute does not exist.

# JavaScript removeAttribute

**Summary**: in this tutorial, you will learn how to use the JavaScript removeAttribute() to remove the attribute with the specified name from the element.

## **Introduction to JavaScript removeAttribute() method**

The removeAttribute() removes an attribute with a specified name from an element:

element.removeAttribute(name);

Code language: CSS (css)

### **Parameters**

The removeAttribute() accepts an argument which is the name of the attribute that you want to remove. If the attribute does not exist, the removeAttribute()  method wil not raise an error.

### **Return value**

The removeAttribute() returns a value of undefined.

### **Usage notes**

HTML elements have some attributes which are Boolean attributes. To set false to the Boolean attributes, you cannot simply use the [setAttribute()](https://www.javascripttutorial.net/javascript-dom/javascript-setattribute/) method, but you have to remove the attribute entirely using the removeAttribute() method.

For example, the values of the disabled attributes are true in the following cases:

**<button disabled>**Save Draft**</button>**

**<button disabled="">**Save**</button>**

**<button disabled="disabled">**Cancel**</button>**

Code language: HTML, XML (xml)

Similarly, the values of the following readonly attributes are true:

**<input type="text" readonly>**

**<textarea type="text" readonly="">**

**<textarea type="text" readonly="readonly">**

Code language: HTML, XML (xml)

## **JavaScript removeAttribute() example**

The following example uses the removeAttribute() method to remove the target attribute from the link element with the id js:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JS removeAttribute() Demo**</title>**

**</head>**

**<body>**

**<a href="https://www.javascripttutorial.net"**

**target="\_blank"**

**id="js">**JavaScript Tutorial**</a>**

**<script>**

let link = document.querySelector('#js');

if (link) {

link.removeAttribute('target');

}

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* Select the link element with id js using the querySelector() method.
* Remove the target attribute by calling the removeAttribute() on the selected link element.

## **Summary**

* Use the removeAttribute() to remove an attribute from a specified element.
* Setting the value of a Boolean attribute to false will not work; use the removeAttribute() method instead.

# JavaScript hasAttribute

**Summary**: in this tutorial, you will learn how to use the JavaScript hasAttribute() to check if an element has an attribute.

## **Introduction to the JavaScript hasAttribute() method**

To check an element has a specified attribute or not, you use the hasAttribute() method:

let result = element.hasAttribute(name);

Code language: JavaScript (javascript)

### **Parameters**

The hasAttribute() method accepts an argument that specifies the name of the attribute that you want to check.

### **Return value**

The hasAttribute() returns a Boolean value that indicates if the element has the specified attribute.

If the element contains an attribute, the hasAttribute() returns true; otherwise, it returns false.

## **JavaScript hasAttribute() example**

See the following example:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JS hasAttribute() Demo**</title>**

**</head>**

**<body>**

**<button id="btnSend" disabled>**Send**</button>**

**<script>**

let btn = document.querySelector('#btnSend');

if (btn) {

let disabled = btn.hasAttribute('disabled');

console.log(disabled);

}

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Output:

true

Code language: JavaScript (javascript)

How it works:

* Select the button with the id btnSend by using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method.
* Check if the button has the disabled attribute by calling the hasAttribute() method on the button element.

## **Summary**

* Use the hasAttribute() method to check if an element contains a specified attribute.

# JavaScript Style

**Summary**: in this tutorial, you will learn how to use the style property to manipulate the inline style of the HTML elements.

## **Setting inline styles**

To set the inline style of an element, you use the style property of that element:

element.style

Code language: CSS (css)

The style property returns the read-only CSSStyleDeclaration object that contains a list of CSS properties. For example, to set the color of an element to red, you use the following code:

element.style.color = 'red';

Code language: JavaScript (javascript)

If the CSS property contains hyphens (-) for example -webkit-text-stroke you can use the array-like notation ([]) to access the property:

element.style.['-webkit-text-stock'] = 'unset';

Code language: JavaScript (javascript)

The following table shows the common CSS properties:

| **CSS** | **JavaScript** |
| --- | --- |
| background | background |
| background-attachment | backgroundAttachment |
| background-color | backgroundColor |
| background-image | backgroundImage |
| background-position | backgroundPosition |
| background-repeat | backgroundRepeat |
| border | border |
| border-bottom | borderBottom |
| border-bottom-color | borderBottomColor |
| border-bottom-style | borderBottomStyle |
| border-bottom-width | borderBottomWidth |
| border-color | borderColor |
| border-left | borderLeft |
| border-left-color | borderLeftColor |
| border-left-style | borderLeftStyle |
| border-left-width | borderLeftWidth |
| border-right | borderRight |
| border-right-color | borderRightColor |
| border-right-style | borderRightStyle |
| border-right-width | borderRightWidth |
| border-style | borderStyle |
| border-top | borderTop |
| border-top-color | borderTopColor |
| border-top-style | borderTopStyle |
| border-top-width | borderTopWidth |
| border-width | borderWidth |
| clear | clear |
| clip | clip |
| color | color |
| cursor | cursor |
| display | display |
| filter | filter |
| float | cssFloat |
| font | font |
| font-family | fontFamily |
| font-size | fontSize |
| font-variant | fontVariant |
| font-weight | fontWeight |
| height | height |
| left | left |
| letter-spacing | letterSpacing |
| line-height | lineHeight |
| list-style | listStyle |
| list-style-image | listStyleImage |
| list-style-position | listStylePosition |
| list-style-type | listStyleType |
| margin | margin |
| margin-bottom | marginBottom |
| margin-left | marginLeft |
| margin-right | marginRight |
| margin-top | marginTop |
| overflow | overflow |
| padding | padding |
| padding-bottom | paddingBottom |
| padding-left | paddingLeft |
| padding-right | paddingRight |
| padding-top | paddingTop |
| page-break-after | pageBreakAfter |
| page-break-before | pageBreakBefore |
| position | position |
| stroke-dasharray | strokeDasharray |
| stroke-dashoffset | strokeDashoffset |
| stroke-width | strokeWidth |
| text-align | textAlign |
| text-decoration | textDecoration |
| text-indent | textIndent |
| text-transform | textTransform |
| top | top |
| vertical-align | verticalAlign |
| visibility | visibility |
| width | width |
| z-index | zIndex |

To completely override the existing inline style, you set the cssText property of the style object. For example:

element.style.cssText = 'color:red;background-color:yellow';

Code language: JavaScript (javascript)

Or you can use the setAttribute() method:

element.setAttribute('style','color:red;background-color:yellow');

Code language: JavaScript (javascript)

Once setting the inline style, you can modify one or more CSS properties:

element.style.color = 'blue';

Code language: JavaScript (javascript)

If you do not want to completely overwrite the existing CSS properties, you can concatenate the new CSS property to the cssText as follows:

element.style.cssText += 'color:red;background-color:yellow';

Code language: JavaScript (javascript)

In this case, the += operator appends the new style string to the existing one.

The following css() helper function is used to set multiple styles for an element from an object of key-value pairs:

function css(e, styles) {

for (const property in styles)

e.style[property] = styles[property];

}

Code language: JavaScript (javascript)

You can use this css() function to set multiple styles for an element with the id #content as follows:

let content = document.querySelector('#content');

css(content, { background: 'yellow', border: 'solid 1px red'});

Code language: JavaScript (javascript)

The following example uses the style object to set the CSS properties of a paragraph with the id content:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JS Style Demo**</title>**

**</head>**

**<body>**

**<p id="content">**JavaScript Setting Style Demo!**</p>**

**<script>**

let p = document.querySelector('#content');

p.style.color = 'red';

p.style.fontWeight = 'bold';

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, select the paragraph element whose id is content by using the querySelector() method.
* Then, set the color and font-weight properties of the paragraph by setting the color and fontWeight properties of the style object.

## **Getting inline styles**

The style property returns the inline styles of an element. It is not very useful in practice because the style property doesn’t return the rules that come from elsewhere e.g., styles from an external style sheet.

To get all styles applied to an element, you should use the [window.getComputedStyle()](https://www.javascripttutorial.net/javascript-dom/javascript-getcomputedstyle/) method.

## **Summary**

* Use the properties of element.style object to set the inline CSS properties for the HTML element.

# JavaScript getComputedStyle

**Summary**: in this tutorial, you will learn how to use the JavaScript getComputedStyle() to get the computed CSS properties of an element.

## **Introduction to JavaScript getComputedStyle() method**

The getComputedStyle() is a method of the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object, which returns an object that contains the computed style an element:

let style = window.getComputedStyle(element [,pseudoElement]);

Code language: JavaScript (javascript)

### **Parameters**

The getComputedStyle() method accepts two arguments:

* element is the element that you want to return the computed styles. If you pass another node type e.g., Text node, the method will throw an error.
* pseudoElement specifies the pseudo-element to match. It defaults to null.

For example, if you want to get the computed value of all the CSS properties of a link with the hover state, you pass the following arguments to the getComputedStyle() method:

let link = document.querySelector('a');

let style = getComputedStyle(link,':hover');

console.log(style);

Code language: JavaScript (javascript)

Note that window is the global object, therefore, you can omit it when calling get the getComputedStyle() method.

### **Return value**

The getComputedStyle() method returns a live style object which is an instance of the CSSStyleDeclaration object. The style is automatically updated when the styles of the element are changed.

## **JavaScript getComputedStyle() examples**

Let’s take some examples of using the getComputedStyle() method.

### **1) Simple getComputedStyle() example**

Consider the following example:

<!DOCTYPE html>

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>**JS getComputedStyle() Demo**</title>**

**<style type="text/css">**

.message {

background-color: #fff3d4;

border: solid 1px #f6b73c;

padding: 20px;

color: black;

}

**</style>**

**</head>**

**<body>**

**<p class="message" style="color:red">**

This is a JS getComputedStyle() Demo!

**</p>**

**<script>**

let message = document.querySelector('.message');

let style = getComputedStyle(message);

console.log('color:', style.color);

console.log('background color:', style.backgroundColor);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Note that for the demonstration purpose, we mix all CSS and JavaScript with HTML. In practice, you should separate them in different files.

Output:

color: rgb(255, 0, 0)

background color: rgb(255, 243, 212)

How it works:

* First, define CSS rules for the message class in the head section of the HTML file. The text color is black.
* Second, declare a paragraph element whose text color is red as defined in the inline style. This rule will override the one defined in the head section.
* Third, use the getComputedStyle() method to get all the computed style of the paragraph element. The color property is red as indicated in the Console window ( rgb(255, 0, 0)) as expected.

### **2) The getComputedStyle() for pseudo-elements example**

The following example uses the getComputedStyle() method to pull style information from a pseudo-element:

**<html>**

**<head>**

**<title>**JavaScript getComputedStyle() Demo**</title>**

**<style>**

body {

font: arial, sans-serif;

font-size: 1em;

line-height: 1.6;

}

p::first-letter {

font-size: 1.5em;

font-weight: normal

}

**</style>**

**</head>**

**<body>**

**<p id='main'>**JavaScript getComputedStyle() Demo for pseudo-elements**</p>**

**<script>**

let p = document.getElementById('main');

let style = getComputedStyle(p, '::first-letter');

console.log(style.fontSize);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Output:

24px

How it works:

* First, define CSS rules for the first letter of any paragraph element in the head section of the HTML file.
* Then, use the getComputedStyle() method to pull the computed style of the pseudo-element. The font-size of the first letter of the paragraph with the id is 24px.

## **Summary**

* The getComputedStyle() is a method of the window object.
* The getComputedStyle() method returns an object that contains the computed style of an element.

# JavaScript className

**Summary**: in this tutorial, you will learn how to use the JavaScript className property to manipulate CSS classes of an element.

The className is the property of an element. It returns a space-separated list of CSS classes of the element:

element.className

Code language: CSS (css)

Suppose that you have the following ul element:

**<ul id="menu" class="vertical main">**

**<li>**Homepage**</li>**

**<li>**Services**</li>**

**<li>**About**</li>**

**<li>**Contact**</li>**

**</ul>**

Code language: HTML, XML (xml)

The following code shows the classes of the ul element on the Console window:

let menu = document.querySelector('#menu');

console.log(menu.className);

Code language: JavaScript (javascript)

Output:

vertical main

To set a class to an element, you use the following code:

element.className += newClassName;

The += operator adds the newClassName to the existing class list of the element.

To completely overwrite all the classes of an element, you use a simple assignment operator. For example:

element.className = "class1 class2";

Code language: JavaScript (javascript)

To get a complete list of classes of an element, you just need to access the className property:

let classes = element.className;

Code language: JavaScript (javascript)

Because the [class](https://www.javascripttutorial.net/es6/javascript-class/) is a keyword in JavaScript, the name className is used instead of the class.

Also the class is an HTML attribute:

**<div id="note" class="info yellow-bg red-text">**JS className**</div>**

Code language: HTML, XML (xml)

while className is a DOM property of the element:

let note = document.querySelector('#note');

console.log(note.className);

Code language: JavaScript (javascript)

Output:

info yellow-bg red-text

An element has another property that helps you better manipulate its CSS classes called [classList](https://www.javascripttutorial.net/javascript-dom/javascript-classlist/).

# JavaScript classList

**Summary**: in this tutorial, you will learn how to use the JavaScript classList property to work with the CSS classes of an element.

## **Introduction to JavaScript classList property**

The classList is a read-only property of an element that returns a live collection of CSS classes:

const classes = element.classList;

Code language: JavaScript (javascript)

The classList is a DOMTokenList object that represents the contents of the element’s class attribute.

Even though the classList is read-only, but you can manipulate the classes it contains using various methods.

## **JavaScript classList examples**

Let’s take some examples of manipulating CSS classes of the element via the classList‘s interface.

### **1) Get the CSS classes of an element**

Suppose that you have a div element with two classes: main and red.

**<div id="content" class="main red">**JavaScript classList**</div>**

Code language: HTML, XML (xml)

The following code displays the class list of the div element in the Console window:

let div = document.querySelector('#content');

for (let cssClass of div.classList) {

console.log(cssClass);

}

Code language: JavaScript (javascript)

Output:

main

red

How it works:

* First, select the div element with the id content using the querySelector() method.
* Then, iterate over the elements of the classList and show the classes in the Console window.

### **2) Add one or more classes to the class list of an element**

To add one or more CSS classes to the class list of an element, you use the add() method of the classList.

For example, the following code adds the info class to the class list of the div element with the id content:

let div = document.querySelector('#content');

div.classList.add('info');

Code language: JavaScript (javascript)

The following example adds multiple CSS classes to the class list of an element:

let div = document.querySelector('#content');

div.classList.add('info','visible','block');

Code language: JavaScript (javascript)

### **3) Remove element’s classes**

To remove a CSS class from the class list of an element, you use the remove() method:

let div = document.querySelector('#content');

div.classList.remove('visible');

Code language: JavaScript (javascript)

Like the add() method, you can remove multiple classes once:

let div = document.querySelector('#content');

div.classList.remove('block','red');

Code language: JavaScript (javascript)

### **4) Replace a class of an element**

To replace an existing CSS class with a new one, you use the replace() method:

let div = document.querySelector('#content');

div.classList.replace('info','warning');

Code language: JavaScript (javascript)

### **5) Check if an element has a specified class**

To check if the element has a specified class, you use the contains() method:

let div = document.querySelector('#content');

div.classList.contains('warning'); *// true*

Code language: JavaScript (javascript)

The contains() method returns true if the classList contains a specified class; otherwise false.

### **6) Toggle a class**

If the class list of an element contains a specified class name, the toggle() method removes it. If the class list doesn’t contain the class name, the toggle() method adds it to the class list.

The following example uses the toggle() method to toggle the visible class of an element with the id content:

let div = document.querySelector('#content');

div.classList.toggle('visible');

Code language: JavaScript (javascript)

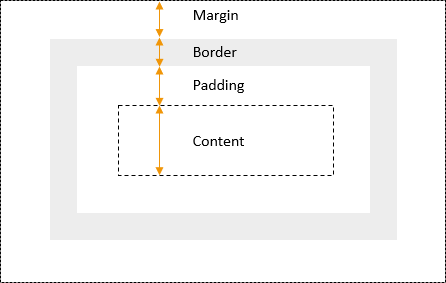
### **Summary**

* The element’s classList property returns the live collection of CSS classes of the element.
* Use add() and remove() to add CSS classes to and remove CSS classes from the class list of an element.
* Use replace() method to replace an existing class with a new one.
* Use contains() method to check if the class list of an element contains a specified class.
* Use the toggle() method to toggle a class.

# Getting the Width and Height of an Element

**Summary**: in this tutorial, you will learn how to get the current computed dimension of an element, including width and height.

The following picture displays the CSS box model that includes a block element with content, padding, border, and margin:



To get the element’s width and height that include the padding and border, you use the offsetWidth and offsetHeight properties of the element:

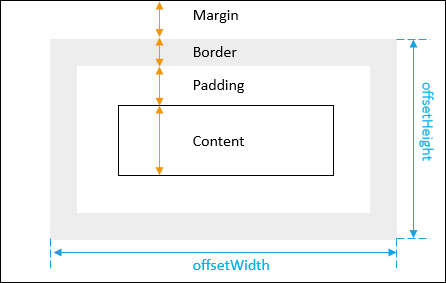
let box = document.querySelector('.box');

let width = box.offsetWidth;

let height = box.offsetHeight;

Code language: JavaScript (javascript)

The following picture illustrates the offsetWidth and offsetHeight of an element:



See the following example:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**Getting the Width and Height of an Element**</title>**

**</head>**

**<body>**

**<div class="box" style="width:100px;height:150px;border:solid 1px #000;padding:10px"></div>**

**<script>**

let box = document.querySelector('.box');

let width = box.offsetWidth;

let height = box.offsetHeight;

console.log({ width, height });

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Output:

{width: 122, height: 172}

Code language: CSS (css)

In this example:

* The width is 100px
* The border is 1px on each side, so 2px for both
* The padding 10px on each side, so 20px for both

Therefore, the total width 12px. Similarly, the height is 172px.

To get the width & height of an element as floating-point after CSS transformation, you use the getBoundingClientRect() method of the DOM element. For example:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**Getting the Width and Height of an Element**</title>**

**</head>**

**<body>**

**<div class="box" style="width:100px;height:150px;border:solid 1px #000;padding:10px"></div>**

**<script>**

let box = document.querySelector('.box');

let width = box.offsetWidth;

let height = box.offsetHeight;

console.log({ width, height });

const domRect = box.getBoundingClientRect();

console.log(domRect);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Output:

{width: 122, height: 172}

DOMRect {x: 7.997685432434082, y: 7.997685432434082, width: 121.95602416992188, height: 171.95602416992188, top: 7.997685432434082, …}

Code language: CSS (css)

## **clientWidth & clientHeight**

To get the element’s width and height that include padding but without the border, you use the clientWidth and clientHeight properties:

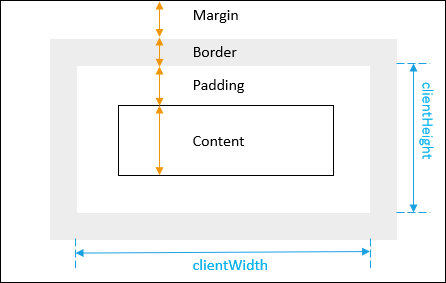
let box = document.querySelector('.box');

let width = box.clientWidth;

let height = box.clientHeight;

Code language: JavaScript (javascript)

The following picture illustrates the clientWidth and clientHeight of an element:



To get the margin of an element, you use the [getComputedStyle()](https://www.javascripttutorial.net/javascript-dom/javascript-getcomputedstyle/) method:

let box = document.querySelector('.box');

let style = getComputedStyle(box);

let marginLeft = parseInt(style.marginLeft);

let marginRight = parseInt(style.marginRight);

let marginTop = parseInt(style.marginTop);

let marginBottom = parseInt(style.marginBottom);

Code language: JavaScript (javascript)

To get the border width of an element, you use the property of the style object returned by the getComputedStyle() method:

let box = document.querySelector('.box');

let style = getComputedStyle(box);

let borderTopWidth = parseInt(style.borderTopWidth) || 0;

let borderLeftWidth = parseInt(style.borderLeftWidth) || 0;

let borderBottomWidth = parseInt(style.borderBottomWidth) || 0;

let borderRightWidth = parseInt(style.borderRightWidth) || 0;

Code language: JavaScript (javascript)

To get the height and width of the window, you use the following code:

let width = window.innerWidth || document.documentElement.clientWidth || document.body.clientWidth;

let height = window.innerHeight || document.documentElement.clientHeight || document.body.clientHeight;

Code language: JavaScript (javascript)

## **Summary**

* Use offsetWidth & offsetHeight properties of the DOM element to get its the width and height.

# JavaScript Events

**Summary**: in this tutorial, you will learn about the JavaScript events, event models, and how to handle events.

## **Introduction to JavaScript events**

An event is an action that occurs in the web browser, which the web browser feedbacks to you so that you can respond to it.

For example, when users click a button on a webpage, you may want to respond to this click event by [displaying a dialog box](https://www.javascripttutorial.net/javascript-bom/javascript-alert/).

Each event may have an event handler which is a block of code that will execute when the event occurs.

An event handler is also known as an event listener. It listens to the event and executes when the event occurs.

Suppose you have a button with the id btn:

**<button id="btn">**Click Me!**</button>**

Code language: HTML, XML (xml)

To define the code that will be executed when the button is clicked, you need to register an event handler using the addEventListener() method:

let btn = document.querySelector('#btn');

function display() {

alert('It was clicked!');

}

btn.addEventListener('click',display);

Code language: JavaScript (javascript)

How it works.

* First, select the button with the id btn by using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method.
* Then, define a [function](https://www.javascripttutorial.net/javascript-function/) called display() as an event handler.
* Finally, register an event handler using the addEventListener() so that when users click the button, the display() function will be executed.

A shorter way to register an event handler is to place all code in an anonymous function, like this:

let btn = document.querySelector('#btn');

btn.addEventListener('click',function() {

alert('It was clicked!');

});

Code language: JavaScript (javascript)

## **Event flow**

Assuming that you have the following HTML document:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS Event Demo**</title>**

**</head>**

**<body>**

**<div id="container">**

**<button id='btn'>**Click Me!**</button>**

**</div>**

**</body>**

Code language: HTML, XML (xml)

When you click the button, you’re clicking not only the button but also the button’s container, the div, and the whole webpage.

Event flow explains the order in which events are received on the page from the element where the event occurs and propagated through the DOM tree.

There are two main event models: event bubbling and event capturing.

### **Event bubbling**

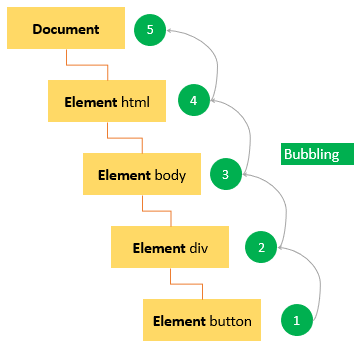
In the event bubbling model, an event starts at the most specific element and then flows upward toward the least specific element (the document or even [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/)).

When you click the button, the click event occurs in the following order:

1. button
2. div with the id container
3. body
4. html
5. document

The click event first occurs on the button which is the element that was clicked. Then the click event goes up the DOM tree, firing on each node along its way until it reaches the document object.

The following picture illustrates the event bubbling effect when users click the button:

[](https://www.javascripttutorial.net/wp-content/uploads/2020/02/JavaScript-event-bubbling.png)

Note that modern web browsers bubble the event up to the window object.

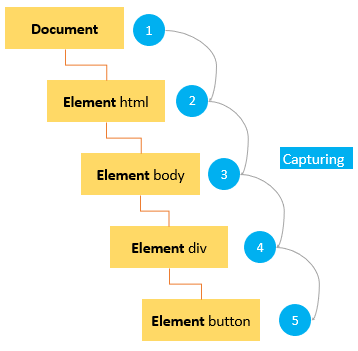
### **Event capturing**

In the event capturing model, an event starts at the least specific element and flows downward toward the most specific element.

When you click the button, the click event occurs in the following order:

1. document
2. html
3. body
4. div with the id container
5. button

The following picture illustrates the event capturing effect:

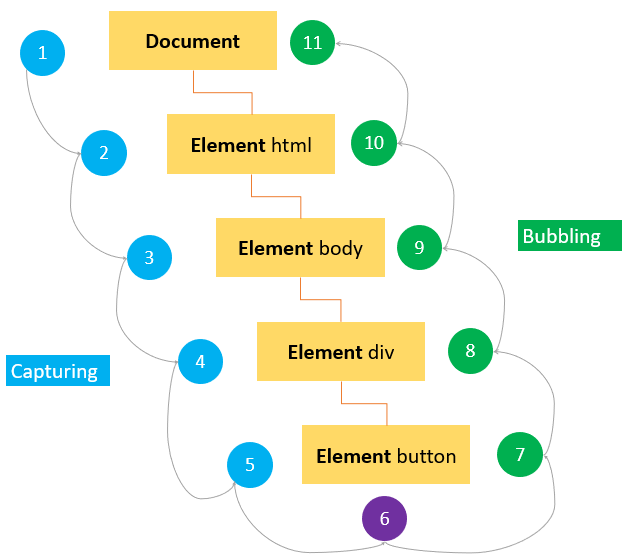
[](https://www.javascripttutorial.net/wp-content/uploads/2020/02/JavaScript-event-capturing.png)

### **DOM Level 2 Event flow**

DOM level 2 events specify that event flow has three phases:

* First, event capturing occurs, which provides the opportunity to intercept the event.
* Then, the actual target receives the event.
* Finally, event bubbling occurs, which allows a final response to the event.

The following picture illustrates the DOM Level 2 event model when users click the button:

[](https://www.javascripttutorial.net/wp-content/uploads/2020/02/JavaScript-DOM-Level-2-Event.png)

## **Event object**

When the event occurs, the web browser passed an Event object to the event handler:

let btn = document.querySelector('#btn');

btn.addEventListener('click', function(event) {

console.log(event.type);

});

Code language: JavaScript (javascript)

Output:

'click'

Code language: JavaScript (javascript)

The following table shows the most commonly-used properties and methods of the event object:

| **Property / Method** | **Description** |
| --- | --- |
| bubbles | true if the event bubbles |
| cancelable | true if the default behavior of the event can be canceled |
| currentTarget | the current element on which the event is firing |
| defaultPrevented | return true if the preventDefault() has been called. |
| detail | more informatio nabout the event |
| eventPhase | 1 for capturing phase, 2 for target, 3 for bubbling |
| preventDefault() | cancel the default behavior for the event. This method is only effective if the cancelable property is true |
| stopPropagation() | cancel any further event capturing or bubbling. This method only can be used if the bubbles property is true. |
| target | the target element of the event |
| type | the type of event that was fired |

Note that the event object is only accessible inside the event handler. Once all the event handlers have been executed, the event object is automatically destroyed.

### **preventDefault()**

To prevent the default behavior of an event, you use the preventDefault() method.

For example, when you click a link, the browser navigates you to the URL specified in the href attribute:

**<a href="https://www.javascripttutorial.net/">**JS Tutorial**</a>**

Code language: HTML, XML (xml)

However, you can prevent this behavior by using the preventDefault() method of the event object:

let link = document.querySelector('a');

link.addEventListener('click',function(event) {

console.log('clicked');

event.preventDefault();

});

Code language: JavaScript (javascript)

Note that the preventDefault() method does not stop the event from bubbling up the DOM. And an event can be canceled when its cancelable property is true.

### **stopPropagation()**

The stopPropagation() method immediately stops the flow of an event through the DOM tree. However, it does not stop the browers default behavior.

See the following example:

let btn = document.querySelector('#btn');

btn.addEventListener('click', function(event) {

console.log('The button was clicked!');

event.stopPropagation();

});

document.body.addEventListener('click',function(event) {

console.log('The body was clicked!');

});

Code language: JavaScript (javascript)

Without the stopPropagation() method, you would see two messages on the Console window.

However, the click event never reaches the body because the stopPropagation() was called on the click event handler of the button.

## **Summary**

* An event is an action that occurs in the web browser e.g., a mouse click.
* Event flow has two main models: event bubbling and event capturing.
* DOM Level 2 Event specifies that the event flow has three phases: event bubbling, the event occurs at the exact element, and event capturing.
* Use addEventListener() to register an event that connects an event to an event listener
* The event object is accessible only within the event listener.
* Use preventDefault() method to prevent the default behavior of an event, but does not stop the event flow.
* Use stopPropagation() method to stop the flow of an event through the DOM tree, but does not cancel the browser default behavior.

# Handling Events in JavaScript

**Summary**: in this tutorial, you will learn the various ways to perform event handling in JavaScript.

When an [event](https://www.javascripttutorial.net/javascript-dom/javascript-events/) occurs, you can create an event handler which is a piece of code that will execute to respond to that event. An event handler is also known as an event listener. It listens to the event and responds accordingly to the event fires.

An event listener is a [function](https://www.javascripttutorial.net/javascript-function/) with an explicit name if it is resuable or an anonymous function in case it is used one time.

An event can be handled by one or multiple event handlers. If an event has multiple event handlers, all the event handlers will be executed when the event is fired.

There are three ways to assign event handlers.

## **1) HTML event handler attributes**

Event handlers typically have names that begin with on, for example, the event handler for the click event is onclick.

To assign an event handler to an event associated with an HTML element, you can use an HTML attribute with the name of the event handler. For example, to execute some code when a button is clicked, you use the following:

**<input type="button" value="Save" onclick="alert('Clicked!')">**

Code language: HTML, XML (xml)

In this case, when the button is clicked, the [alert](https://www.javascripttutorial.net/javascript-bom/javascript-alert/) box is shown.

When you assign JavaScript code as the value of the onclick attribute, you need to escape the HTML characters such as ampersand (&), double quotes ("), less than (<), etc., or you will get a syntax error.

An event handler defined in the HTML can call a function defined in a script. For example:

**<script>**

function showAlert() {

alert('Clicked!');

}

**</script>**

**<input type="button" value="Save" onclick="showAlert()">**

Code language: HTML, XML (xml)

In this example, the button calls the showAlert() function when it is clicked.

The showAlert() is a function defined in a separate <script> element, and could be placed in an external JavaScript file.

### **Important notes**

The following are some important points when you use the event handlers as attributes of the HTML element:

First, the code in the event handler can access the event object without explicitly defining it:

**<input type="button" value="Save" onclick="alert(event.type)">**

Code language: HTML, XML (xml)

Second, the this value inside the event handler is equivalent to the event’s target element:

**<input type="button" value="Save" onclick="alert(this.value)">**

Code language: HTML, XML (xml)

Third, the event handler can access the element’s properties, for example:

**<input type="button" value="Save" onclick="alert(value)">**

Code language: HTML, XML (xml)

### **Disadvantages of using HTML event handler attributes**

Assigning event handlers using HTML event handler attributes are considered as bad practices and should be avoided as much as possible because of the following reasons:

First, the event handler code is mixed with the HTML code, which will make the code more difficult to maintain and extend.

Second, it is a timing issue. If the element is loaded fully before the JavaScript code, users can start interacting with the element on the webpage which will cause an error.

For example, suppose that the following showAlert() function is defined in an external JavaScript file:

**<input type="button" value="Save" onclick="showAlert()">**

Code language: HTML, XML (xml)

And when the page is loaded fully and the JavaScript has not been loaded, the showAlert() function is undefined. If users click the button at this moment, an error will occur.

## **2) DOM Level 0 event handlers**

Each element has event handler properties such as onclick. To assign an event handler, you set the property to a function as shown in the example:

let btn = document.querySelector('#btn');

btn.onclick = function() {

alert('Clicked!');

};

Code language: JavaScript (javascript)

In this case, the anonymous function becomes the method of the button element. Therefore, the this value is equivalent to the element. And you can access the element’s properties inside the event handler:

let btn = document.querySelector('#btn');

btn.onclick = function() {

alert(this.id);

};

Code language: JavaScript (javascript)

Output:

btn

By using the this value inside the event handler, you can access the element’s properties and methods.

To remove the event handler, you set the value of the event handler property to null:

btn.onclick = null;

Code language: JavaScript (javascript)

The DOM Level 0 event handlers are still being used widely because of its simplicity and cross-browser support.

## **3) DOM Level 2 event handlers**

DOM Level 2 Event Handlers provide two main methods for dealing with the registering/deregistering event listeners:

* addEventListener() – register an event handler
* removeEventListener() – remove an event handler

These methods are available in all DOM nodes.

## **The addEventListener() method**

The addEventListener() method accepts three arguments: an event name, an event handler function, and a Boolean value that instructs the method to call the event handler during the capture phase (true) or during the bubble phase (false). For example:

let btn = document.querySelector('#btn');

btn.addEventListener('click',function(event) {

alert(event.type); *// click*

});

Code language: JavaScript (javascript)

It is possible to add multiple event handlers to handle a single event, like this:

let btn = document.querySelector('#btn');

btn.addEventListener('click',function(event) {

alert(event.type); *// click*

});

btn.addEventListener('click',function(event) {

alert('Clicked!');

});

Code language: JavaScript (javascript)

## **The removeEventListener() method**

The removeEventListener() removes an event listener that was added via the addEventListener(). However, you need to pass the same arguments as were passed to the addEventListener(). For example:

let btn = document.querySelector('#btn');

*// add the event listener*

let showAlert = function() {

alert('Clicked!');

};

btn.addEventListener('click', showAlert);

*// remove the event listener*

btn.removeEventListener('click', showAlert);

Code language: JavaScript (javascript)

Using an anonymous event listener as the following will not work:

let btn = document.querySelector('#btn');

btn.addEventListener('click',function() {

alert('Clicked!');

});

*// won't work*

btn.removeEventListener('click', function() {

alert('Clicked!');

});

Code language: JavaScript (javascript)

## **Summary**

* There are three ways to assign an event handler: HTML event handler attribute, element’s event handler property, and addEventListener().
* Assign an event handler via the HTML event handler attribute should be avoided.

# JavaScript Page Load Events

**Summary**: in this tutorial, you will learn about the events that are related to the page load including DOMContentLoaded, load, beforeunload, and unload.

## **Overview of JavaScript page load events**

When you open a page, the following events occur in sequence:

* [DOMContentLoaded](https://www.javascripttutorial.net/javascript-dom/javascript-domcontentloaded/) – the browser fully loaded HTML and completed building the DOM tree. However, it hasn’t loaded external resources like stylesheets and images. In this event, you can start selecting DOM nodes or initialize the interface.
* [load](https://www.javascripttutorial.net/javascript-dom/javascript-onload/) – the browser fully loaded the HTML and also external resources like images and stylesheets.

When you leave the page, the following events fire in sequence:

* [beforeunload](https://www.javascripttutorial.net/javascript-dom/javascript-beforeunload-event/) – fires before the page and resources are unloaded. You can use this event to show a confirmation dialog to confirm if you really want to leave the page. By doing this, you can prevent data loss in case you are filling out a form and accidentally click a link to navigate to another page.
* [unload](https://www.javascripttutorial.net/javascript-dom/javascript-unload/) – fires when the page has completely unloaded. You can use this event to send the analytic data or to clean up resources.

## **Handling JavaScript page load events**

To handle the page events, you can call the addEventListener() method on the document object:

document.addEventListener('DOMContentLoaded',() => {

*// handle DOMContentLoaded event*

});

document.addEventListener('load',() => {

*// handle load event*

});

document.addEventListener('beforeunload',() => {

*// handle beforeunload event*

});

document.addEventListener('unload',() => {

*// handle unload event*

});

Code language: JavaScript (javascript)

The following example illustrates how to handle the page load events:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS Page Load Events**</title>**

**</head>**

**<body>**

**<script>**

addEventListener('DOMContentLoaded', (event) => {

console.log('The DOM is fully loaded.');

});

addEventListener('load', (event) => {

console.log('The page is fully loaded.');

});

addEventListener('beforeunload', (event) => {

*// show the confirmation dialog*

event.preventDefault();

*// Google Chrome requires returnValue to be set.*

event.returnValue = '';

});

addEventListener('unload', (event) => {

*// send analytic data*

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

# JavaScript onload

**Summary**: in this tutorial, you will learn how to handle the load event that fires on the document, image, and script elements in JavaScript.

## **The window’s load event**

For the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object, the load event is fired when the whole webpage (HTML) has loaded fully, including all dependent resources, including JavaScript files, CSS files, and images.

To handle the load event, you register an event listener using the addEventListener() method:

window.addEventListener('load', (event) => {

console.log('The page has fully loaded');

});

Code language: JavaScript (javascript)

Or use the onload property of the window object:

window.onload = (event) => {

console.log('The page has fully loaded');

};

Code language: JavaScript (javascript)

If you maintain a legacy system, you may find that the load event handler is registered in of the body element of the HTML document, like this:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS load Event Demo**</title>**

**</head>**

**<body onload="console.log('Loaded!')">**

**</body>**

**</html>**

Code language: HTML, XML (xml)

It’s a good practice to use the addEventListener() method to assign the onload event handler whenever possible.

## **The image’s load event**

The load event also fires on images. To handle the load event on images, you use the addEventListener() method of the image elements.

The following example uses the load event handler to determine if an image, which exists in the DOM tree, has been completely loaded:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**Image load Event Demo**</title>**

**</head>**

**<body>**

**<img id="logo">**

**<script>**

let logo = document.querySelector('#logo');

logo.addEventListener('load', (event) => {

console.log('Logo has been loaded!');

});

logo.src = "logo.png";

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

You can assign an onload event handler directly using the onload attribute of the <img> element, like this:

**<img id="logo"**

**src="logo.png"**

**onload="console.log('Logo loaded!')">**

Code language: HTML, XML (xml)

If you create an image element dynamically, you can assign an onload event handler before setting the src property as follows:

window.addEventListener('load' () => {

let logo = document.createElement('img');

*// assign and onload event handler*

logo.addEventListener('load', (event) => {

console.log('The logo has been loaded');

});

*// add logo to the document*

document.body.appendChild(logo);

logo.src = 'logo.png';

});

Code language: JavaScript (javascript)

How it works:

* First, create an image element after the document has been fully loaded by placing the code inside the event handler of the window’s load event.
* Second, assign the onload event handler to the image.
* Third, add the image to the document.
* Finally, assign an image URL to the src attribute. The image will be downloaded to the element as soon as the src property is set.

## **The script’s load event**

The <script> element also supports the load event slightly different from the standard ways. The script’s load event allows you to check if a JavaScript file has been completely loaded.

Unlike images, the web browser starts downloading JavaScript files only after the src property has been assigned and the <script> element has been added to the document.

The following code loads the app.js file after the page has been completely loaded. It assigns an onload event handler to check if the app.js has been fully loaded.

window.addEventListener('load', checkJSLoaded)

function checkJSLoaded() {

*// create the script element*

let script = document.createElement('script');

*// assign an onload event handler*

script.addEventListener('load', (event) => {

console.log('app.js file has been loaded');

});

*// load the script file*

script.src = 'app.js';

document.body.appendChild(script);

}

Code language: JavaScript (javascript)

## **Summary**

* The load event occurs when the document has been completely loaded, including dependent resources like JavaScript files, CSS files, and images.
* The <img> and <script> elements also support the load event.
* Use the addEventListener() method to register an onload event handler.

# JavaScript DOMContentLoaded

**Summary**: in this tutorial, you will learn about the JavaScript DOMContentLoaded event.

The DOMContentLoaded fires when the DOM content is loaded, without waiting for images and stylesheets to finish loading.

You need to handle the DOMContentLoaded event when you place the JavaScript in the head of the page but referencing elements in the body, for example:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS DOMContentLoaded Event**</title>**

**<script>**

let btn = document.getElementById('btn');

btn.addEventListener('click', (e) => {

*// handle the click event*

console.log('clicked');

});

**</script>**

**</head>**

**<body>**

**<button id="btn">**Click Me!**</button>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In this example, we reference the button in the body from the JavaScript in the head.

Because the DOM has not been loaded when the JavaScript engine parses the JavaScript in the head, the button with the id btn does not exist.

To fix this, you place the code inside an DOMContentLoaded event handler, like this:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS DOMContentLoaded Event**</title>**

**<script>**

document.addEventListener('DOMContentLoaded', () => {

let btn = document.getElementById('btn');

btn.addEventListener('click', () => {

*// handle the click event*

console.log('clicked');

});

});

**</script>**

**</head>**

**<body>**

**<button id="btn">**Click Me!**</button>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

When you place JavaScript in the header, it will cause bottlenecks and rendering delays, so it’s better to move the script before the </body> tag. In this case, you don’t need to place the code in the DOMContentLoaded event, like this:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS DOMContentLoaded Event**</title>**

**</head>**

**<body>**

**<button id="btn">**Click Me!**</button>**

**<script>**

document.addEventListener('DOMContentLoaded', () => {

let btn = document.getElementById('btn');

btn.addEventListener('click', () => {

*// handle the click event*

console.log('clicked');

});

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **Summary**

* The DOMContentLoaded event fires when the DOM content is loaded, without waiting for images and stylesheets to finish loading.
* Only handle DOMContentLoaded event if you place the JavaScript code in the head, which references elements in the body section.

# JavaScript beforeunload Event

**Summary**: in this tutorial, you will learn about the JavaScript beforeunload event that allows you to display a confirmation dialog when before you leave a webpage.

## **Introduction to JavaScript beforeunload event**

Before the webpage and its resources are unloaded, the beforeunload event is fired. At this time, the webpage is still visible and you have an opportunity to cancel the event.

To register for the beforeunload event, you use the window.addEventListener() method:

window.addEventListener('beforeunload',(event) =>{

*// do something here*

});

Code language: JavaScript (javascript)

Since the window is the global object, you can omit it like this:

addEventListener('beforeunload',(event) =>{

*// do something here*

});

Code language: PHP (php)

If a webpage has a beforeunload event listener and you are about to leave the page, the beforeunload event will trigger a confirmation dialog to confirm if you really want to leave the page.

If you confirm, the browser navigates you to the new page. Otherwise, it cancels the navigation.

According to the specification, you need to call the preventDefault() method inside the beforeunload event handler in order to show the confirmation dialog. However, not all browsers implement this:

addEventListener('beforeunload',(event) => {

event.preventDefault();

});

Code language: PHP (php)

Historically, some browsers allow you to display a custom message on the confirmation dialog. This was intended to inform the users that they will lose data if they navigate away. Unfortunately, this feature is often used to scam users. As a result, a custom message is no longer supported.

Based on the HTML specification, the calls to [alert()](https://www.javascripttutorial.net/javascript-bom/javascript-alert/), [confirm()](https://www.javascripttutorial.net/javascript-bom/javascript-confirm/), and [prompt()](https://www.javascripttutorial.net/javascript-bom/javascript-prompt/) are ignored in the beforeunload event handler.

## **JavaScript beforeunload event example**

The following example attaches an event handler to the beforeunload event. If you click the link to navigate to another page, the browser will show a confirmation dialog.

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JS beforeunload Event**</title>**

**</head>**

**<body>**

**<a href="https://www.javascripttutorial.net/">**JavaScript Tutorial**</a>**

**<script>**

window.addEventListener('beforeunload', (event) => {

event.preventDefault();

*// Google Chrome requires returnValue to be set.*

event.returnValue = '';

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **Summary**

* The beforeunload event is fired before the webpage and its resources are about to unload.
* Use beforeunload to confirm if users really want to leave the page to prevent data loss.

# JavaScript Mouse Events

**Summary**: in this tutorial, you will learn about the basic mouse events and their properties in JavaScript.

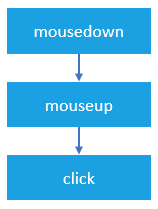
## **Introduction to JavaScript mouse events**

Mouse events fire when you use the mouse to interact with the elements on the page. DOM Level 3 events define nine mouse events.

### **mousedown, mouseup, and click**

When you click an element, there are no less than three mouse events fire in the following sequence:

1. The mousedown fires when you depress the mouse button on the element.
2. The mouseup fires when you release the mouse button on the element.
3. The click fires when one mousedown and one mouseup detected on the element.

[](https://www.javascripttutorial.net/wp-content/uploads/2020/02/javascript-mouse-event-click-event.png)

If you depress the mouse button on an element and move your mouse off the element, and then release the mouse button. The only mousedown event fires on the element.

Likewise, if you depress the mouse button, and move the mouse over the element, and release the mouse button, the only mouseup event fires on the element.

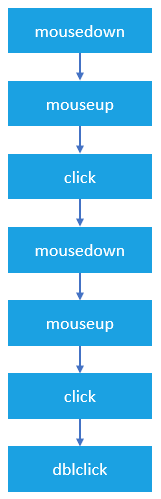
In both cases, the click event never fires.

### **dblclick**

In practice, you rarely use the dblclick event. The dblclick event fires when you double click over an element.

It takes two click events to cause a dblclick event to fire. The dblclick event has four events fired in the following order:

1. mousedown
2. mouseup
3. click
4. mousedown
5. mouseup
6. click
7. dblclick



As you can see, the click events always take place before the dblclick event. If you register both click and dblclick event handlers on the same element, you will not know exactly what user actually has clicked or double-clicked the element.

### **mousemove**

The mousemove event fires repeatedly when you move the mouse cursor around an element. Even when you move the mouse one pixel, the mousemove event still fires. It will cause the page slow, therefore, you only register mousemove event handler only when you need it and immediately remove the event handler as soon as it is no longer used, like this:

element.onmousemove = mouseMoveEventHandler;

*// ...*

*// later, no longer use*

element.onmousemove = null;

Code language: JavaScript (javascript)

### **mouseover / mouseout**

The mouseover fires when the mouse cursor is outside of the element and then move to inside the boundaries of the element.

The mouseout fires when the mouse cursor is over an element and then move another element.

### **mouseenter / mouseleave**

The mouseenter fires when the mouse cursor is outside of an element and then moves to inside the boundaries of the element.

The mouseleave fires when the mouse cursor is over an element and then moves to the outside of the element’s boundaries.

Both mouseenter and mouseleave does not bubble and does not fire when the mouse cursor moves over descendant elements.

## **Registering mouse event handlers**

To register a mouse event, you use these steps:

* First, select the element by using [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) or [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Then, register the mouse event using the addEventListener() method.

For example, suppose that you have the following button:

**<button id="btn">**Click Me!**</button>**

Code language: HTML, XML (xml)

To register a mouse click event handler, you use the following code:

let btn = document.querySelector('#btn');

btn.addEventListener('click',(event) => {

console.log('clicked');

});

Code language: JavaScript (javascript)

or you can assign a mouse event handler to the element’s property:

let btn = document.querySelector('#btn');

btn.onclick = (event) => {

console.log('clicked');

};

Code language: JavaScript (javascript)

In legacy systems, you may find that the event handler is assigned in the HTML attribute of the element:

**<button id="btn" onclick="console.log('clicked')">**Click Me!**</button>**

Code language: HTML, XML (xml)

It’s a good practice to always use the addEventListener() to register a mouse event handler.

## **Detecting mouse buttons**

The event object passed to the mouse event handler has a property called button that indicates which mouse button was pressed on the mouse to trigger the event.

The mouse button is represented by a number:

* 0: the main mouse button pressed, usually the left button.
* 1: the auxiliary button pressed, usually the middle button or the wheel button.
* 2: the secondary button pressed, usually the right button.
* 3: the fourth button pressed, usually the Browser Back button.
* 4: the fifth button pressed, usually the Browser Forward button.



See the following example:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS Mouse Events - Button Demo**</title>**

**</head>**

**<body>**

**<button id="btn">**Click me with any mouse button: left, right, middle, ...**</button>**

**<p id="message"></p>**

**<script>**

let btn = document.querySelector('#btn');

*// disable context menu when right-mouse clicked*

btn.addEventListener('contextmenu', (e) => {

e.preventDefault();

});

*// show the mouse event message*

btn.addEventListener('mouseup', (e) => {

let msg = document.querySelector('#message');

switch (e.button) {

case 0:

msg.textContent = 'Left mouse button clicked.';

break;

case 1:

msg.textContent = 'Middle mouse button clicked.';

break;

case 2:

msg.textContent = 'Right mouse button clicked.';

break;

default:

msg.textContent = `Unknown mouse button code: ${event.button}`;

}

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In this example, when you click the button with your mouse (left-click, right-click, and middle-click), it shows a corresponding message on the <div> element.

Click me with any mouse button: left, right, middle, …

## **Modifier keys**

When you click an element, you may press one or more modifier keys: Shift, Ctrl, Alt, and Meta.

[A close up of a keyboard

Description automatically generated with medium confidence](https://www.javascripttutorial.net/wp-content/uploads/2020/02/javascript-mouse-event-modifier-keys.png)

Note the Meta key is the Windows key on Windows keyboards and the Command key on Apple keyboard.

To detect if these modifier keys have been pressed, you can use the event object passed to the mouse event handler.

The event object has four Boolean properties, where each is set to true if the key is being held down or false if the key is not pressed.

See the following example:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS Modifier Keys Demo**</title>**

**</head>**

**<body>**

**<button id="btnKeys">**Click me with Alt, Shift, Ctrl pressed**</button>**

**<p id="messageKeys"></p>**

**<script>**

let btnKeys = document.querySelector('#btnKeys');

btnKeys.addEventListener('click', (e) => {

let keys = [];

if (e.shiftKey) keys.push('shift');

if (e.ctrlKey) keys.push('ctrl');

if (e.altKey) keys.push('alt');

if (e.metaKey) keys.push('meta');

let msg = document.querySelector('#messageKeys');

msg.textContent = `Keys: ${keys.join('+')}`;

});

**</script>**

**</body>**

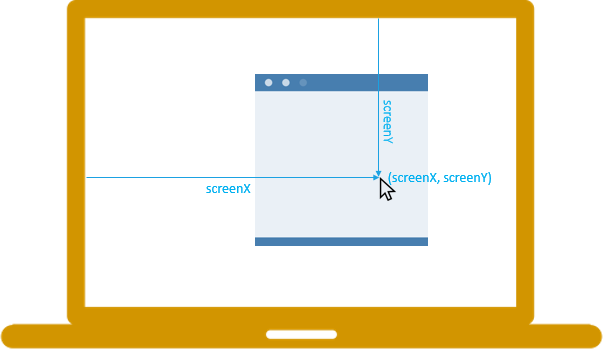
**</html>**

Code language: HTML, XML (xml)

Click me with Alt, Shift, Ctrl pressed

## **Getting Screen Coordinates**

The screenX and screenY properties of the event passed to the mouse event handler return the screen coordinates of the location of the mouse in relation to the entire screen.



On the other hand, the clientX and clientY properties provide the horizontal and vertical coordinates within the application’s client area at which the mouse event occurred:

Diagram

Description automatically generated with low confidence

See the following demo:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS Mouse Location Demo**</title>**

**<style>**

#track {

background-color: goldenrod;

height: 200px;

width: 400px;

}

**</style>**

**</head>**

**<body>**

**<p>**Move your mouse to see its location.**</p>**

**<div id="track"></div>**

**<p id="log"></p>**

**<script>**

let track = document.querySelector('#track');

track.addEventListener('mousemove', (e) => {

let log = document.querySelector('#log');

log.innerText = `

Screen X/Y: (${e.screenX}, ${e.screenY})

Client X/Y: (${e.clientX}, ${e.clientY})`

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Move your mouse to see its location.

## **Summary**

* DOM Level 3 defines nine mouse events.
* Use addEventListener() method to register a mouse event handler.
* The event.button indicates which mouse button was pressed to trigger the mouse event.
* The modifier keys: alt, shift, ctrl, and meta (Mac) can be obtained via properties of the event object passed to the mouse event handler.
* The screenX and screenY properties return the horizontal and vertical coordinates of the mouse pointer in screen coordinates.
* The clientX and clientY properties of the event object return horizontal and vertical coordinates within the application’s client area at which the mouse event occurred.

# JavaScript Keyboard Events

**Summary**: in this tutorial, you will learn how to work with JavaScript keyboard events including keydown, keypress, and keyup.

## **Introduction to JavaScript keyboard events**

When you interact with the keyboard, the keyboard [events](https://www.javascripttutorial.net/javascript-dom/javascript-events/) are fired. There are three main keyboard events:

* keydown – fires when you press a key on the keyboard and it fires repeatedly while you holding down the key.
* keyup – fires when you release a key on the keyboard.
* keypress – fires when you press a character keyboard like a,b, or c, not the left arrow key, home, or end keyboard, … The keypress also fires repeatedly while you hold down the key on the keyboard.

The keyboard events typically fire on the text box, through all elements support them.

When you press a character key once on the keyboard, three keyboard events are fired in the following order:

1. keydown
2. keypress
3. keyup

Both keydown and keypress events are fired before any change made to the text box, whereas the keyup event fires after the changes have made to the text box. If you hold down a character key, the keydown and keypress are fired repeatedly until you release the key.

When you press a non-character key, the keydown event is fired first followed by the keyup event. If you hold down the non-character key, the keydown is fired repeatedly until you release the key.

## **Handling keyboard events**

To handle a keyboard event, you follow these steps:

* First, select the element on which the keyboard event will fire. Typically, it is a text box.
* Then, use the element.addEventListener() to register an event handler.

Suppose that you have the following text box with the id message:

**<input type="text" id="message">**

Code language: HTML, XML (xml)

The following illustrates how to register keyboard event listeners:

let msg = document.getDocumentById('#message');

msg.addEventListener("keydown", (event) => {

*// handle keydown*

});

msg.addEventListener("keypress", (event) => {

*// handle keypress*

});

msg.addEventListener("keyup", (event) => {

*// handle keyup*

});

Code language: PHP (php)

If you press a character key, all three event handlers will be called.

## **The keyboard event properties**

The keyboard event has two important properties: key and code. The key property returns the character that has been pressed whereas the code property returns the physical key code.

For example, if you press the z character key, the event.key returns z and event.code returns KeyZ.

See the following example:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JavaScript Keyboard Events: Key/Code**</title>**

**</head>**

**<body>**

**<input type="text" id="message">**

**<script>**

let textBox = document.getElementById('message');

textBox.addEventListener('keydown', (event) => {

console.log(`key=${event.key},code=${event.code}`);

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

If you type character z, you will see the following message:

key=z,code=KeyZ

How it works:

* First, select the text box with the id message by using the [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) method.
* Then, register a keydown event listener and log the key and code of the key that has been pressed.

## **Summary**

* When you press a character key on the keyboard, the keydown, keypress, and keyup events are fired sequentially. However, if you press a non-character key, only the keydown and keyup events are fired.
* The keyboard event object has two important properties: key and code  properties that allow you to detect which key has been pressed.
* The key property returns the value of the key pressed while the code represents a physical key on the keyboard.

# JavaScript Scroll Events

**Summary**: in this tutorial, you will learn about the JavaScript scroll events and how to handle scroll event properly.

## **Introduction to the JavaScript scroll events**

When you scroll a document or an element, the scroll events fire. You can trigger the scroll events in the following ways, for example:

* Using the scrollbar manually
* Using the mouse wheel
* Clicking an ID link
* Calling functions in JavaScript

To register a scroll event handler, you call the addEventListener() method on the target element, like this:

targetElement.addEventListener('scroll', (event) => {

*// handle the scroll event*

});

Code language: PHP (php)

or assign an event handler to the onscroll property of the target element:

targetElement.onscroll = (event) => {

*// handle the scroll event*

};

Code language: JavaScript (javascript)

## **Scrolling the document**

Typically, you handle the scroll events on the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object to handle the scroll of the whole webpage.

The following shows how to attach an event handler to the scroll event of a page:

window.addEventListener('scroll',(event) => {

console.log('Scrolling...');

});

Code language: JavaScript (javascript)

Or you can use the onscroll property on the window object:

window.onscroll = function(event) {

*//*

};

Code language: JavaScript (javascript)

The onscroll property of the window object is the same as document.body.onscroll and you can use them interchangeably, for example:

document.body.onscroll = null;

console.log(window.onscroll); *// null*

Code language: JavaScript (javascript)

### **Scroll offsets**

The window object has two properties related to the scroll events: scrollX and scrollY.

The scrollX and scrollY properties return the number of pixels that the document is currently scrolled horizontally and vertically. The scrollX and scrollY are double-precision floating-point values so if you need integer values, you can use the Math.round() to round them off.

The scrollX and scrollY are 0 if the document hasn’t been scrolled at all.

The pageXOffset and pageYOffset are aliases of the scrollX and scrollY properties.

## **Scrolling an element**

Like the window object, you can attach a scroll event handler to any HTML element. However, to track the scroll offset, you use the scrollTop and scrollLeft instead of the scrollX and scrollY.

The scrollTop property sets or gets the number of pixels that the element’s content is vertically scrolled. The scrollLeft property gets and sets the number of pixels that an element’s content is scrolled from its left edge.

The following example shows how to handle the scroll event of the div element with the id scrollDemo:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JS Scroll Events**</title>**

**<style>**

#scrollDemo {

height: 200px;

width: 200px;

overflow: auto;

background-color: #f0db4f

}

#scrollDemo p {

*/\* show the scrollbar \*/*

height: 300px;

width: 300px;

}

**</style>**

**</head>**

**<body>**

**<div id="scrollDemo">**

**<p>**JS Scroll Event Demo**</p>**

**</div>**

**<div id="control">**

**<button id="btnScrollLeft">**Scroll Left**</button>**

**<button id="btnScrollTop">**Scroll Top**</button>**

**</div>**

**<script>**

let control = document.querySelector('#control');

control.addEventListener('click', function (e) {

*// get the scrollDemo*

let div = document.getElementById('scrollDemo');

*// get the target*

let target = e.target;

*// handle each button's click*

switch (target.id) {

case 'btnScrollLeft':

div.scrollLeft += 20;

break;

case 'btnScrollTop':

div.scrollTop += 20;

break;

}

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **The better ways to handle the scroll events**

Many scroll events fire while you are scrolling a page or an element. If you attach an event listener to the scroll event, the code in the event handler needs time to execute.

This will cause an issue which is known as the scroll jank. The scroll jank effect causes a delay so that the page doesn’t feel anchored to your finger.

### **Event throttling**

It is much better to keep the scroll event handler lightweight and execute it every N milliseconds by using a timer. So instead of using the following code (and you should never use it):

window.scroll = () => {

*// place the scroll handling logic here*

};

Code language: JavaScript (javascript)

You should use the following code:

let scrolling = false;

window.scroll = () => {

scrolling = true;

};

setInterval(() => {

if (scrolling) {

scrolling = false;

*// place the scroll handling logic here*

}

},300);

Code language: JavaScript (javascript)

How it works:

* First, set the scrolling flag to false. If the scroll event fires set the scrolling flag to true inside the scroll event handler.
* Then, execute the scroll event handler using the [setInterval()](https://www.javascripttutorial.net/javascript-bom/javascript-setinterval/) every 300 milliseconds if the scroll events have fired.

This way of handling the scroll event is called the **event throttling** that throttles an onscroll‘s underlying operation every 300 milliseconds. The throttling slows down the rate of execution of the scroll event handler.

### **Passive events**

Recently, modern web browsers support passive events for the input events like scroll, touchstart, wheel, etc. It allows the UI thread to handle the event immediately before passing over control to your custom event handler.

In the web browsers which support the passive events, you need to add the passive flag to any event listener that does not call preventDefault(), like this:

document.addEventListener(

'scroll',

(event) => {

*// handle scroll event*

},

{ passive: true }

);

Code language: JavaScript (javascript)

Without the passive option, the code in the event handler will always be invoked before the UI thread carries out the scrolling.

Check out which browsers are supporting passive events [here](https://caniuse.com/#feat=passive-event-listener).

## **Summary**

* The scroll event fires when you scroll a webpage or an element.
* For a page, the scrollX and scrollY properties return the number of pixels that the document is currently scrolled horizontally and vertically.
* For an element, the scrollTop and scrollLeft properties set or get the number of pixels that the element’s content is vertically scrolled and scrolled from its left edge.
* Use the event throttling technique to better handle the scroll events. In the modern web browsers, you can use passive event listeners.

# JavaScript scrollIntoView

**Summary**: in this tutorial, you’ll learn how to scroll an element into the view using its scrollIntoView() method.

Suppose you have a list of elements and you want a specific element to be highlighted and scrolled into view.

To achieve this, you can use the element.scrollIntoView() method. The element.scrollIntoView() accepts a boolean value or an object:

element.scrollIntoView(alignToTop);

Code language: JavaScript (javascript)

or

element.scrollIntoView(options);

Code language: CSS (css)

The method accepts one of the following two arguments:

### **alignToTop**

The alignToTop is a boolean value.

If it is set to true, the method will align the top of the element to the top of the viewport or the top of the visible area of the scrollable ancestor.

If the alignToTop is set to false, the method will align the bottom of the element to the bottom of the viewport or the bottom of the visible area of the scrollable ancestor.

By default, the alignToTop is true.

### **options**

The options argument is an object that gives more control over of alignment of the element in the view. However, the web browser support may be slightly different.

The options object has the following properties:

* behavior property defines the transition animation. The behavior property accepts two values: auto or smooth. It defaults to auto.
* block property defines the vertical alignment. It accepts one of four values: start, center, endor nearest. By default, it is start.
* inline property defines horizontal alignment. It also accepts one of four values: start, center, endor nearest. It defaults to nearest.

## **JavaScript scrollIntoView() Example**

Suppose that you have an HTML page with a list of the programming language as follows:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JS scrollIntoView Demo**</title>**

**<link rel="stylesheet" href="style.css">**

**</head>**

**<body>**

**<div class="container">**

**<button class="btn">**Scroll Into View**</button>**

**<ul>**

**<li>**C**</li>**

**<li>**Java**</li>**

**<li>**Python**</li>**

**<li>**C++**</li>**

**<li>**C#**</li>**

**<li>**Go**</li>**

**<li>**Visual Basic**</li>**

**<li>**JavaScript**</li>**

**<li>**PHP**</li>**

**<li>**SQL**</li>**

**<li>**R**</li>**

**<li>**Swift**</li>**

**<li class="special">**JavaScript**</li>**

**<li>**MATLAB**</li>**

**<li>**Assembly language**</li>**

**<li>**Ruby**</li>**

**<li>**PL/SQL**</li>**

**<li>**Classic Visual Basic**</li>**

**<li>**Perl**</li>**

**<li>**Scratch**</li>**

**<li>**Objective-C**</li>**

**</ul>**

**</div>**

**<script src="scrollIntoView.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

Without scrolling, the JavaScript list item, which has a class called special, is not in the viewport. When the button "Scroll Into View" is clicked, the JavaScript list item is scrolled into the view:

let btn = document.querySelector('.btn');

let el = document.querySelector('.special');

btn.addEventListener('click', function () {

el.scrollIntoView(true);

});

Code language: JavaScript (javascript)

How it works:

* First, [select](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) the button with the btn class and list item with the special class.
* Then, attach an event listener to the click event of the button.
* Finally, scroll the JavaScript list item into the viewport by calling the el.scrollIntoView(true) method in the click event handler.

[Here is the JavaScript scrollIntoView() demo.](https://www.javascripttutorial.net/sample/dom/scrollintoview/)

To align the JavaScript list item to the bottom of the view, you pass false value to the scrollIntoView() method:

let btn = document.querySelector('.btn');

let el = document.querySelector('.special');

btn.addEventListener('click', function() {

el.scrollIntoView(false);

});

Code language: JavaScript (javascript)

In this tutorial, you have learned how to use the JavaScript scrollIntoView() method to scroll an element into the viewport.

# JavaScript Focus Events

**Summary**: in this tutorial, you will learn about the JavaScript focus events that keep track of the elements that users focus on.

## **Introduction to JavaScript focus events**

The focus events fire when an element receives or loses focus. These are the two main focus events:

* focus fires when an element has received focus.
* blur fires when an element has lost focus.

The focusin and focusout fire at the same time as focus and blur, however, they bubble while the focus and blur do not.

The following elements are focusable:

* The **window** gains focus when you bring it forward by using Alt+Tab or clicking on it and loses focus when you send it back.
* **Links** when you use a mouse or a keyboard.
* **Form fields** like input text when you use a keyboard or a mouse.
* Elements with **tabindex**, also when you use a keyboard or a mouse.

## **JavaScript focus event examples**

The following example shows how to handle the focus and blur events. When you move focus to the password field, the background changes to yellow. If you move the mouse to the username field, the background changes to white.

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JavaScript Focus Events**</title>**

**</head>**

**<body>**

**<p>**Move focus to the password field to see the effect:**</p>**

**<form id="form">**

**<input type="text" placeholder="username">**

**<input type="password" placeholder="password">**

**</form>**

**<script>**

const pwd = document.querySelector('input[type="password"]');

pwd.addEventListener('focus', (e) => {

e.target.style.backgroundColor = 'yellow';

});

pwd.addEventListener('blur', (e) => {

e.target.style.backgroundColor = '';

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **Summary**

* Use the focus event to handle the state of element when it has or loses the focus.

# JavaScript hashchange Event

**Summary**: in this tutorial, you will learn about the JavaScript hashchange event and how to handle it effectively.

## **Introduction to the JavaScript hashchange event**

The URL hash is everything that follows the pound sign (#) in the URL. For example, suppose that you have the following URL:

https:*//www.javascripttutorial.net/javascript-dom/javascript-hashchange/#header*

Code language: JavaScript (javascript)

The URL hash is header. If the URL hash changes to footer, like this:

https:*//www.javascripttutorial.net/javascript-dom/javascript-hashchange/#footer*

Code language: JavaScript (javascript)

The hashchange event fires when the URL hash changes from one to another. In this example, it changes from #header to #footer.

To attach an event listener to the hashchange event, you call the addEventListener() method on the [window](https://www.javascripttutorial.net/javascript-bom/javascript-window/) object:

window.addEventListener('hashchange',() =>{

console.log('The URL has has changed');

});

Code language: JavaScript (javascript)

To get the current URL hash, you access the hash property of the location object:

window.addEventListener('hashchange',() => {

console.log(`The current URL hash is ${location.hash}`);

});

Code language: JavaScript (javascript)

Additionally, you can handle the hashchange event by assigning an event listener to the onhashchange property of the window object:

window.onhashchange = () => {

*// handle hashchange event here*

};

Code language: JavaScript (javascript)

## **Summary**

* The hashchange event fires when the URL hash changed.
* To register an event listener, you call the addEventListener() method or assign an event listener to the onhashchange property of the window object.

# JavaScript Event Delegation

**Summary**: in this tutorial, you will learn how to use the JavaScript event delegation that adds a single event handler to the parent element instead of having to register multiple event handlers to the child elements.

## **Introduction to JavaScript Event Delegation**

Suppose that you have the following menu:

**<ul id="menu">**

**<li><a id="home">**home**</a></li>**

**<li><a id="dashboard">**Dashboard**</a></li>**

**<li><a id="report">**report**</a></li>**

**</ul>**

Code language: HTML, XML (xml)

To handle the click event of each menu item, you may add the corresponding click event handlers:

let home = document.querySelector('#home');

home.addEventListener('home',(event) => {

console.log('Home menu item was clicked');

});

let dashboard = document.querySelector('#dashboard');

dashboard.addEventListener('dashboard',(event) => {

console.log('Dashboard menu item was clicked');

});

let report = document.querySelector('#report');

report.addEventListener('report',(event) => {

console.log('Report menu item was clicked');

});

Code language: JavaScript (javascript)

In JavaScript, if you have a large number of [event handlers](https://www.javascripttutorial.net/javascript-dom/handling-events-in-javascript/) on a page, these event handlers will directly impact the performance because of the following reasons:

* First, each event handler is a [function](https://www.javascripttutorial.net/javascript-function/) which is also an [object](https://www.javascripttutorial.net/javascript-objects/) that takes up memory. The more objects in the memory, the slower the performance.
* Second, it takes time to assign all the event handlers, which causes a delay in the interactivity of the page.

To solve this issue, you can leverage the [event bubbling](https://www.javascripttutorial.net/javascript-dom/javascript-events/).

Instead of having multiple event handlers, you can assign a single event handler to handle all the click events:

let menu = document.querySelector('#menu');

menu.addEventListener('click', (event) => {

let target = event.target;

switch(target.id) {

case 'home':

console.log('Home menu item was clicked');

break;

case 'dashboard':

console.log('Dashboard menu item was clicked');

break;

case 'report':

console.log('Report menu item was clicked');

break;

}

});

Code language: JavaScript (javascript)

How it works.

* When you click any <a> element inside the <ul> element with the id menu, the click event bubbles to the parent element which is the <ul> element. So instead of handling the click event of the individual <a> element, you can capture the click event at the parent element.
* In the click event listener, you can access the target property which references the element that dispatches the event. To get the id of the element that the event actually fires, you use the target.id property.
* Once having the id of the element that fires the click event, you can have code that handles the event correspondingly.

The way that we handle the too-many-event-handlers problem is called the event delegation.

The event delegation refers to the technique of levering event bubbling to handle events at a higher level in the DOM than the element on which the event originated.

## **JavaScript event delegation benefits**

When it is possible, you can have a single event handler on the document that will handle all the events of a particular type. By doing this, you gain the following benefits:

* Less memory usage, better performance.
* Less time required to set up event handlers on the page.
* The document object is available immediately. As long as the element is rendered, it can start functioning correctly without delay. You don’t need to wait for the DOMContentLoaded or load events.

## **Summary**

* Having a large number of event handlers will take up memory and degrade the performance of a page.
* The event delegation technique utilizes the event bubbling to handle event at a higher level in the DOM than the element on which the event originated.

# JavaScript dispatchEvent

**Summary**: in this tutorial, you’ll learn how to programmatically create and dispatch events using Event constructor and dispatchEvent() method.

Typically, events are generated by user actions such as [mouse clicks](https://www.javascripttutorial.net/javascript-dom/javascript-mouse-events/) and [key presses](https://www.javascripttutorial.net/javascript-dom/javascript-keyboard-events/). In addition, events can be generated from code.

To generate an event programmatically, you follow these steps:

* First, create a new Event object using Event constructor.
* Then, trigger the event using element.dispatchEvent() method.

## **Event constructor**

To create a new event, you use the Event constructor like this:

let event = new Event(type, [,options]);

Code language: JavaScript (javascript)

The Event constructor accepts two parameters:

### **type**

is a string that specifies the event type such as 'click'.

### **options**

is an object with two optional properties:

* bubbles: is a boolean value that determines if the event bubbles or not. If it is true then the event is bubbled and vice versa.
* cancelable: is also a boolean value that specifies whether the event is cancelable when it is true.

By default, the options object is:

{ bubbles: false, cancelable: false}

Code language: CSS (css)

For example, the following creates a new click event with the default options object:

let clickEvent = new Event('click');

Code language: JavaScript (javascript)

## **dispatchEvent method**

After creating an event, you can fire it on a target element using the dispatchEvent() method like this:

element.dispatchEvent(event);

Code language: CSS (css)

For example, the following code shows how to create the click event and fire it on a button:

HTML:

**<button class="btn">**Test**</button>**

Code language: HTML, XML (xml)

JavaScript:

let btn = document.querySelector('.btn');

btn.addEventListener('click', function () {

alert('Mouse Clicked');

});

let clickEvent = new Event('click');

btn.dispatchEvent(clickEvent);

Code language: JavaScript (javascript)

[Click this link to see the demo](https://www.javascripttutorial.net/sample/dom/dispatchevent/).

In this example, the event handler is executed as if the click event were generated by user actions.

If the event comes from the user actions, the event.isTrusted property is set to true. In case the event is generated by code, the event.isTrusted is false. Therefore, you can examine the value of event.isTrusted property to check the “authenticity” of the event.

The Event is the base type of UIEvent which is the base type of other specific event types such as MouseEvent, TouchEvent, FocusEvent, and KeyboardEvent.

It’s a good practice to use the specialized event constructor like MouseEvent instead of using the generic Event type because these constructors provide more information specific to the events.

For example, the MouseEvent event has many other properties such as clientX and clientY that specify the horizontal and vertical coordinates at which the event occurred relative to the viewport:

let clickEvent = new MouseEvent("click", {

bubbles: true,

cancelable: true,

clientX: 150,

clientY: 150

});

Code language: JavaScript (javascript)

The following link shows the full list of properties of the [MouseEvent](https://www.w3.org/TR/uievents/" \l "idl-mouseevent" \t "_blank)

## **Summary**

* Use the specific event constructor such as MouseEvent and call dispatchEvent() method on an element to generate an event from code.
* Use event.isTrusted to examine whether the event is generated from code or user actions.

# JavaScript Custom Events

**Summary**: in this tutorial, you will learn about the JavaScript custom events such as creating a custom event and dispatching it.

## **Introduction to JavaScript custom events**

The following [function](https://www.javascripttutorial.net/javascript-function/) highlights an element by changing its background color to yellow:

function highlight(elem) {

const bgColor = 'yellow';

elem.style.backgroundColor = bgColor;

}

Code language: JavaScript (javascript)

To execute a piece of code after highlighting the element, you may come up with a [callback](https://www.javascripttutorial.net/javascript-callback/):

function highlight(elem, callback) {

const bgColor = 'yellow';

elem.style.backgroundColor = bgColor;

if(callback && typeof callback === 'function') {

callback(elem);

}

}

Code language: JavaScript (javascript)

The following calls the highlight() function and adds a border to a <div> element:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JS Custom Event Demo**</title>**

**</head>**

**<body>**

**<div class="note">**JS Custom Event Demo**</div>**

**<script>**

function highlight(elem, callback) {

const bgColor = 'yellow';

elem.style.backgroundColor = bgColor;

if (callback && typeof callback === 'function') {

callback(elem);

}

}

let note = document.querySelector('.note');

function addBorder(elem) {

elem.style.border = "solid 1px red";

}

highlight(note, addBorder);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

To make the code more flexible, you can use the custom event.

### **Creating JavaScript custom events**

To create a custom event, you use the CustomEvent() constructor:

let event = new CustomEvent(eventType, options);

Code language: JavaScript (javascript)

The CustomEvent() has two parameters:

* The eventType is a string that represents the name of the event.
* The options is an object has the detail property that contains any custom information about the event.

The following example shows how to create a new custom event called highlight:

let event = new CustomEvent('highlight', {

detail: {backgroundColor: 'yellow'}

});

Code language: JavaScript (javascript)

### **Dispatching JavaScript custom events**

After creating a custom event, you need to attach the event to a DOM element and trigger it by using the [dispatchEvent()](https://www.javascripttutorial.net/javascript-dom/javascript-dispatchevent/) method:

domElement.dispatchEvent(event);

Code language: JavaScript (javascript)

## **JavaScript custom event example**

Put it all together:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript Custom Event**</title>**

**</head>**

**<body>**

**<div class="note">**JS Custom Event**</div>**

**<script>**

function highlight(elem) {

const bgColor = 'yellow';

elem.style.backgroundColor = bgColor;

*// create the event*

let event = new CustomEvent('highlight', {

detail: {

backgroundColor: bgColor

}

});

*// dispatch the event*

elem.dispatchEvent(event);

}

*// Select the div element*

let div = document.querySelector('.note');

*// Add border style*

function addBorder(elem) {

elem.style.border = "solid 1px red";

}

*// Listen to the highlight event*

div.addEventListener('highlight', function (e) {

addBorder(this);

*// examine the background*

console.log(e.detail);

});

*// highlight div element*

highlight(div);

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, declare the highlight() function that highlights an element and triggers the highlight event.
* Second, select the <div> element by using the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method.
* Third, listen to the highlight event. Inside the event listener, call the addBorder() function and show the detail property in the Console.
* Finally, call the highlight() function that will trigger the highlight event.

## **Why using custom events**

The custom events allow you to decouple the code that you want to execute after another piece of code completes. For example, you can separate the event listeners in a separate script. In addition, you can have multiple event listeners to the same custom event.

## **Summary**

* Use the CustomEvent() constructor to create a custom event and dispatchEvent() to trigger the event.

# JavaScript MutationObserver

**Summary**: in this tutorial, you will learn how to use the JavaScript MutationObserver API to watch for changes being made to the DOM tree.

## **Introduction to the JavaScript MutationObserver API**

The MutationObserver API allows you to monitor for changes being made to the DOM tree. When the DOM nodes change, you can invoke a [callback function](https://www.javascripttutorial.net/javascript-callback/) to react to the changes.

The basic steps for using the the MutationObserver API are:

First, define the callback function that will execute when the DOM changes:

function callback(mutations) {

*//*

}

Code language: JavaScript (javascript)

Second, create a MutationObserver object and pass the callback into the MutationObserver() constructor:

let observer = new MutationObserver(callback);

Code language: JavaScript (javascript)

Third, call the observe() method to start observing the DOM changes.

observer.observe(targetNode, observerOptions);

Code language: JavaScript (javascript)

The observe() method has two parameters. The target is the root of the subtree of nodes to monitor for changes. The observerOptions parameter contains properties that specify what DOM changes should be reported to the observer’s callback.

Finally, stop observing for the DOM changes by calling the disconnect() method:

observer.disconnect();

Code language: JavaScript (javascript)

## **The MutationObserver options**

The second argument of the observe() method allows you to specify options to describe the MutationObserver:

let options = {

childList: true,

attributes: true,

characterData: false,

subtree: false,

attributeFilter: ['attr1', 'attr2'],

attributeOldValue: false,

characterDataOldValue: false

};

Code language: JavaScript (javascript)

You don’t need to use all the options. However, to make the MutationObserver works, at least one of childList, attributes, or characterData needs to be set to true, otherwise the observer() method will throw an error.

### **Observing changes to child elements**

Assuming that you have the following list:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**MutationObserver Demo: ChildList**</title>**

**</head>**

**<body>**

**<ul id="language">**

**<li>**HTML**</li>**

**<li>**CSS**</li>**

**<li>**JavaScript**</li>**

**<li>**TypeScript**</li>**

**</ul>**

**<button id="btnStart">**Start Observing**</button>**

**<button id="btnStop">**Stop Observing**</button>**

**<button id="btnAdd">**Add**</button>**

**<button id="btnRemove">**Remove the Last Child**</button>**

**<script src="app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

The following example illustrates how to use the childList property of the mutation options object to monitor for the child node changes.

First, select the elements like the list and buttons using the querySelector() method. By default, the Stop Observing button is disabled.

*// selecting list*

let list = document.querySelector('#language');

*// selecting buttons*

let btnAdd = document.querySelector('#btnAdd');

let btnRemove = document.querySelector('#btnRemove');

let btnStart = document.querySelector('#btnStart');

let btnStop = document.querySelector('#btnStop');

btnStop.disabled = true;

Code language: JavaScript (javascript)

Second, declare a log() function that will be used as a callback for the MutationObserver:

function log(mutations) {

for (let mutation of mutations) {

if (mutation.type === 'childList') {

console.log(mutation);

}

}

}

Code language: JavaScript (javascript)

Third, create a new MutationObserver object:

let observer = new MutationObserver(log);

Code language: JavaScript (javascript)

Fourth, start observing the DOM changes to the child nodes of the list element when the Start Observing button is clicked by calling the observe() method with the childList of the options object is set to true:

btnStart.addEventListener('click', function () {

observer.observe(list, {

childList: true

});

btnStart.disabled = true;

btnStop.disabled = false;

});

Code language: JavaScript (javascript)

Fifth, add a new list item when the add button is clicked:

let counter = 1;

btnAdd.addEventListener('click', function () {

*// create a new item element*

let item = document.createElement('li');

item.textContent = `Item ${counter++}`;

*// append it to the child nodes of list*

list.appendChild(item);

});

Code language: JavaScript (javascript)

Sixth, remove the last child of the list when the Remove button is clicked:

btnRemove.addEventListener('click', function () {

list.lastElementChild ?

list.removeChild(list.lastElementChild) :

console.log('No more child node to remove');

});

Code language: JavaScript (javascript)

Finally, stop observing DOM changes when the Stop Observing button is clicked by calling the disconnect() method of the MutationObserver object:

btnStop.addEventListener('click', function () {

observer.disconnect();

*// set button states*

btnStart.disabled = false;

btnStop.disabled = true;

});

Code language: JavaScript (javascript)

Put it all together:

(function () {

*// selecting the list*

let list = document.querySelector('#language');

*// selecting the buttons*

let btnAdd = document.querySelector('#btnAdd');

let btnRemove = document.querySelector('#btnRemove');

let btnStart = document.querySelector('#btnStart');

*// disable the stop button*

let btnStop = document.querySelector('#btnStop');

btnStop.disabled = true;

function log(mutations) {

for (let mutation of mutations) {

if (mutation.type === 'childList') {

console.log(mutation);

}

}

}

let observer = new MutationObserver(log);

btnStart.addEventListener('click', function () {

observer.observe(list, {

childList: true

});

btnStart.disabled = true;

btnStop.disabled = false;

});

btnStop.addEventListener('click', function () {

observer.disconnect();

*// Set the button state*

btnStart.disabled = false;

btnStop.disabled = true;

});

let counter = 1;

btnAdd.addEventListener('click', function () {

*// create a new item element*

let item = document.createElement('li');

item.textContent = `Item ${counter++}`;

*// append it to the child nodes of list*

list.appendChild(item);

});

btnRemove.addEventListener('click', function () {

list.lastElementChild ?

list.removeChild(list.lastElementChild) :

console.log('No more child node to remove');

});

})();

Code language: JavaScript (javascript)

Notice that we placed all code in an [IIFE](https://www.javascripttutorial.net/javascript-immediately-invoked-function-expression-iife/) (Immediately Invoked Function Expression).

### **Observing for changes to attributes**

To observe for changes to attributes, you use the following attributes property of the options object:

let options = {

attributes: true

}

Code language: JavaScript (javascript)

If you want to observe the changes to one or more specific attributes while ignoring the others, you can use the attributeFilter property:

let options = {

attributes: true,

attributeFilter: ['class', 'style']

}

Code language: JavaScript (javascript)

In this example, the MutationObserver will invoke the callback each time the class or style attribute changes.

### **Observing for changes to a subtree**

To monitor the target node and its subtree of nodes, you set the subtree property of the options object to true:

let options = {

subtree: true

}

Code language: JavaScript (javascript)

### **Observing for changes to character data**

To monitor the node for changes to its textual contents, you set the characterData property of the options object to true:

let options = {

characterData: true

}

Code language: JavaScript (javascript)

### **Accessing old values**

To access the old values of attributes, you set the attributeOldValue property of the options object to true:

let options = {

attributes: true,

attributeOldValue: true

}

Code language: JavaScript (javascript)

Similarly, you can access the old value of character data by setting the characterDataOldValue property of the options object to true:

let options = {

characterData: true,

subtree: true,

characterDataOldValue: true

}

Code language: JavaScript (javascript)

In this tutorial, you have learned about the JavaScript MutationObserver API that monitors the DOM changes and executes a callback every time the change occurs.

# JavaScript Form

**Summary**: in this tutorial, you will learn about JavaScript form API: accessing the form, getting values of the elements, validating form data, and submitting the form.

## **Form basics**

To create a form in HTML, you use the <form> element:

**<form action="/signup" method="post" id="signup">**

**</form>**

Code language: HTML, XML (xml)

The <form> element has two important attributes: action and method.

* The action attribute specifies a URL that will process the form submission. In this example, the action is the /signup URL.
* The method attribute specifies the HTTP method to submit the form with. Usually, the method is either post or get.

Generally, you use the get method when you want to retrieve data from the server and the post method when you want to change data on the server.

JavaScript uses the HTMLFormElement object to represent a form. The HTMLFormElement has the following properties that correspond to the HTML attributes:

* action – is the URL that processes the form data. It is equivalent to the action attribute of the <form> element.
* method – is the HTTP method which is equivalent to the method attribute of the <form> element.

The HTMLFormElement element also provides the following useful methods:

* submit() – submit the form.
* reset() – reset the form.

## **Referencing forms**

To reference the <form> element, you can use DOM selecting methods such as [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/):

const form = document.getElementById('subscribe');

Code language: JavaScript (javascript)

An HTML document can have multiple forms. The document.forms property returns a collection of forms (HTMLFormControlsCollection) on the document:

document.forms

Code language: JavaScript (javascript)

To reference a form, you use an index. For example, the following statement returns the first form of the HTML document:

document.forms[0]

Code language: CSS (css)

## **Submitting a form**

Typically, a form has a submit button. When you click it, the browser sends the form data to the server. To create a submit button, you use <input> or <button> element with the type submit:

**<input type="submit" value="Subscribe">**

Code language: HTML, XML (xml)

Or

**<button type="submit">**Subscribe**</button>**

Code language: HTML, XML (xml)

If the submit button has focus and you press the Enter key, the browser also submits the form data.

When you submit the form, the submit event is fired before the request is sent to the server. This gives you a chance to validate the form data. If the form data is invalid, you can stop submitting the form.

To attach an event listener to the submit event, you use the addEventListener() method of the form element as follows:

const form = document.getElementById('signup');

form.addEventListener('submit', (event) => {

*// handle the form data*

});

Code language: JavaScript (javascript)

To stop the form from being submitted, you call the preventDefault() method of the event object inside the submit event handler like this:

form.addEventListener('submit', (event) => {

*// stop form submission*

event.preventDefault();

});

Code language: PHP (php)

Typically, you call the event.preventDefault() method if the form data is invalid. To submit the form in JavaScript, you call the submit() method of the form object:

form.submit();

Code language: CSS (css)

Note that the form.submit() does not fire the submit event. Therefore, you should always [validate the form](https://www.javascripttutorial.net/javascript-dom/javascript-form-validation/) before calling this method.

## **Accessing form fields**

To access form fields, you can use DOM methods like [getElementsByName()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementsbyname/), [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/), [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/), etc.

Also, you can use the elements property of the form object. The form.elements property stores a collection of the form elements.

JavaScript allows you to access an element by index, id, or name. Suppose that you have the following signup form with two <input> elements:

**<form action="signup.html" method="post" id="signup">**

**<h1>**Sign Up**</h1>**

**<div class="field">**

**<label for="name">**Name:**</label>**

**<input type="text" id="name" name="name" placeholder="Enter your fullname" />**

**<small></small>**

**</div>**

**<div class="field">**

**<label for="email">**Email:**</label>**

**<input type="text" id="email" name="email" placeholder="Enter your email address" />**

**<small></small>**

**</div>**

**<button type="submit">**Subscribe**</button>**

**</form>**

Code language: HTML, XML (xml)

To access the elements of the form, you get the form element first:

const form = document.getElementById('signup');

Code language: JavaScript (javascript)

And use index, id, or name to access the element. The following accesses the first form element:

form.elements[0]; *// by index*

form.elements['name']; *// by name*

form.elements['name']; *// by id (name & id are the same in this case)*

Code language: JavaScript (javascript)

The following accesses the second input element:

form.elements[1]; *// by index*

form.elements['email']; *// by name*

form.elements['email']; *// by id*

Code language: JavaScript (javascript)

After accessing a form field, you can use the value property to access its value, for example:

const form = document.getElementById('signup');

const name = form.elements['name'];

const email = form.elements['email'];

*// getting the element's value*

let fullName = name.value;

let emailAddress = email.value;

Code language: JavaScript (javascript)

## **Put it all together: signup form**

The following illustrates the HTML document that has a signup form. [See the live demo here](https://www.javascripttutorial.net/sample/dom/form/signup.html).

<!DOCTYPE html>

**<html lang="en">**

**<head>**

**<title>**JavaScript Form Demo**</title>**

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

**<link rel="stylesheet" href="css/style.css" />**

**</head>**

**<body>**

**<div class="container">**

**<form action="signup.html" method="post" id="signup">**

**<h1>**Sign Up**</h1>**

**<div class="field">**

**<label for="name">**Name:**</label>**

**<input type="text" id="name" name="name" placeholder="Enter your fullname" />**

**<small></small>**

**</div>**

**<div class="field">**

**<label for="email">**Email:**</label>**

**<input type="text" id="email" name="email" placeholder="Enter your email address" />**

**<small></small>**

**</div>**

**<div class="field">**

**<button type="submit" class="full">**Subscribe**</button>**

**</div>**

**</form>**

**</div>**

**<script src="js/app.js"></script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

The HTML document references the [style.css](https://www.javascripttutorial.net/sample/dom/form/css/style.css) and [app.js](https://www.javascripttutorial.net/sample/dom/form/js/app.js) files. It uses the <small> element to display an error message in case the <input> element has invalid data.

Submitting the form without providing any information will result in the following error:

[Graphical user interface, application

Description automatically generated](https://www.javascripttutorial.net/wp-content/uploads/2021/07/JavaScript-Form-Example.png)

Submitting the form with the name but invalid email address format will result in the following error:

[Graphical user interface, application

Description automatically generated](https://www.javascripttutorial.net/wp-content/uploads/2021/07/JavaScript-Form-validate-email-address.png)

The following shows the complete app.js file:

*// show a message with a type of the input*

function showMessage(input, message, type) {

*// get the small element and set the message*

const msg = input.parentNode.querySelector("small");

msg.innerText = message;

*// update the class for the input*

input.className = type ? "success" : "error";

return type;

}

function showError(input, message) {

return showMessage(input, message, false);

}

function showSuccess(input) {

return showMessage(input, "", true);

}

function hasValue(input, message) {

if (input.value.trim() === "") {

return showError(input, message);

}

return showSuccess(input);

}

function validateEmail(input, requiredMsg, invalidMsg) {

*// check if the value is not empty*

if (!hasValue(input, requiredMsg)) {

return false;

}

*// validate email format*

const emailRegex =

/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;

const email = input.value.trim();

if (!emailRegex.test(email)) {

return showError(input, invalidMsg);

}

return true;

}

const form = document.querySelector("#signup");

const NAME\_REQUIRED = "Please enter your name";

const EMAIL\_REQUIRED = "Please enter your email";

const EMAIL\_INVALID = "Please enter a correct email address format";

form.addEventListener("submit", function (event) {

*// stop form submission*

event.preventDefault();

*// validate the form*

let nameValid = hasValue(form.elements["name"], NAME\_REQUIRED);

let emailValid = validateEmail(form.elements["email"], EMAIL\_REQUIRED, EMAIL\_INVALID);

*// if valid, submit the form.*

if (nameValid && emailValid) {

alert("Demo only. No form was posted.");

}

});

Code language: JavaScript (javascript)

How it works.

### **The showMessage() function**

The showMessage() function accepts an input element, a message, and a type:

*// show a message with a type of the input*

function showMessage(input, message, type) {

*// get the <small> element and set the message*

const msg = input.parentNode.querySelector("small");

msg.innerText = message;

*// update the class for the input*

input.className = type ? "success" : "error";

return type;

}

Code language: JavaScript (javascript)

The following shows the name input field on the form:

**<div class="field">**

**<label for="name">**Name:**</label>**

**<input type="text" id="name" name="name" placeholder="Enter your fullname" />**

**<small></small>**

**</div>**

Code language: HTML, XML (xml)

If the name’s value is blank, you need to get its parent first which is the <div> with the class “field”.

input.parentNode

Code language: CSS (css)

Next, you need to select the <small> element:

const msg = input.parentNode.querySelector("small");

Code language: JavaScript (javascript)

Then, update the message:

msg.innerText = message;

After that, we change the CSS class of the input field based on the value of the type parameter. If the type is true, we change the class of the input to success. Otherwise, we change the class to error.

input.className = type ? "success" : "error";

Code language: JavaScript (javascript)

Finally, return the value of the type:

return type;

Code language: JavaScript (javascript)

### **The showError() and showSuccess() functions**

The the showError() and showSuccess() functions call the showMessage() function. The showError() function always returns false whereas the showSuccess() function always returns true. Also, the showSuccess() function sets the error message to an empty string.

function showError(input, message) {

return showMessage(input, message, false);

}

function showSuccess(input) {

return showMessage(input, "", true);

}

Code language: JavaScript (javascript)

### **The hasValue() function**

The hasValue() function checks if an input element has a value or not and show an error message using the showError() or showSuccess() function accordingly:

function hasValue(input, message) {

if (input.value.trim() === "") {

return showError(input, message);

}

return showSuccess(input);

}

Code language: JavaScript (javascript)

### **The validateEmail() function**

The validateEmail() function validates if an email field contains a valid email address:

function validateEmail(input, requiredMsg, invalidMsg) {

*// check if the value is not empty*

if (!hasValue(input, requiredMsg)) {

return false;

}

*// validate email format*

const emailRegex =

/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;

const email = input.value.trim();

if (!emailRegex.test(email)) {

return showError(input, invalidMsg);

}

return true;

}

Code language: PHP (php)

The validateEmail() function calls the hasValue() function to check if the field value is empty. If the input field is empty, it shows the requiredMsg.

To validate the email, it uses a [regular expression](https://www.javascripttutorial.net/javascript-regular-expression/). If the email is invalid, the validateEmail() function shows the invalidMsg.

### **The submit event handler**

First, select the signup form by its id using the querySelector() method:

const form = document.querySelector("#signup");

Code language: JavaScript (javascript)

Second, define some constants to store the error messages:

const NAME\_REQUIRED = "Please enter your name";

const EMAIL\_REQUIRED = "Please enter your email";

const EMAIL\_INVALID = "Please enter a correct email address format";

Code language: JavaScript (javascript)

Third, add the submit event listener of the signup form using the addEventListener() method:

form.addEventListener("submit", function (event) {

*// stop form submission*

event.preventDefault();

*// validate the form*

let nameValid = hasValue(form.elements["name"], NAME\_REQUIRED);

let emailValid = validateEmail(form.elements["email"], EMAIL\_REQUIRED, EMAIL\_INVALID);

*// if valid, submit the form.*

if (nameValid && emailValid) {

alert("Demo only. No form was posted.");

}

});

Code language: JavaScript (javascript)

In the submit event handler:

1. Stop the form submission by calling the event.preventDefault() method.
2. Validate the name and email fields using the hasValue() and validateEmail() functions.
3. If both name and email are valid, show an alert. In a real-world application, you need to call the form.submit() method to submit the form.

## **Summary**

* Use the <form> element to create an HTML form.
* Use DOM methods such as getElementById() and querySelector() to select a <form> element. The document.forms[index] also returns the form element by a numerical index.
* Use form.elements to access form elements.
* The submit event fires when users click the submit button on the form.

# JavaScript Radio Button

**Summary**: in this tutorial, you will learn how to use JavaScript to check which radio button in a radio group is checked.

## **Introduction to the JavaScript Radio Button**

Radio buttons allow you to select only one of a predefined set of mutually exclusive options. To create a radio button, you use the <input> element with the type radio. A group of radio buttons is called a radio group.

To form a radio group, you use a common name for all the radio buttons. For example:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript Radio Button**</title>**

**</head>**

**<body>**

**<p>**Select your size:**</p>**

**<div>**

**<input type="radio" name="size" value="XS" id="xs">**

**<label for="xs">**XS**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="S" id="s">**

**<label for="s">**S**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="M" id="m">**

**<label for="m">**M**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="L" id="l">**

**<label for="l">**L**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="XL" id="xl">**

**<label for="xl">**XL**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="XXL" id="xxl">**

**<label for="xxl">**XXL**</label>**

**</div>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In this example, all the radio buttons have the same name size but different values. Because of this, you can only select one radio button at a time.

To find the selected radio button, you follow these steps:

* Select all radio buttons by using a DOM method such as [querySelectorAll()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method.
* Get the checked property of the radio button. If the checked property is true, the radio button is checked; otherwise, it is unchecked.

To know which radio button is checked, you use the value attribute. For example:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript Radio Button**</title>**

**</head>**

**<body>**

**<p>**Select your size:**</p>**

**<div>**

**<input type="radio" name="size" value="XS" id="xs">**

**<label for="xs">**XS**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="S" id="s">**

**<label for="s">**S**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="M" id="m">**

**<label for="m">**M**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="L" id="l">**

**<label for="l">**L**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="XL" id="xl">**

**<label for="xl">**XL**</label>**

**</div>**

**<div>**

**<input type="radio" name="size" value="XXL" id="xxl">**

**<label for="xxl">**XXL**</label>**

**</div>**

**<p>**

**<button id="btn">**Show Selected Value**</button>**

**</p>**

**<p id="output"></p>**

**<script>**

const btn = document.querySelector('#btn');

const radioButtons = document.querySelectorAll('input[name="size"]');

btn.addEventListener("click", () => {

let selectedSize;

for (const radioButton of radioButtons) {

if (radioButton.checked) {

selectedSize = radioButton.value;

break;

}

}

*// show the output:*

output.innerText = selectedSize ? `You selected ${selectedSize}` : `You haven't selected any size`;

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works.

First, select the button with #btn id, output element with the #output id, and all the radio buttons with the name size:

const btn = document.querySelector('#btn');

const output = document.querySelector('#output');

const radioButtons = document.querySelectorAll('input[name="size"]');

Code language: JavaScript (javascript)

Second, register a click event listener on the button element:

btn.addEventListener('click', () => {

});

Code language: PHP (php)

Third, iterate over the radio buttons and get the value of the selected radio button:

let selectedSize;

for (const radioButton of radioButtons) {

if (radioButton.checked) {

selectedSize = radioButton.value;

break;

}

}

Code language: JavaScript (javascript)

If a radio button is checked, its checked property is true. Then, we assign the value of the selected radio button to the selectedSize variable.

Since only one radio button in a radio group can be checked at a time, the loop is terminated immediately by the break statement.

Finally, set the message for the output element:

output.innerText = selectedSize ? `You selected ${selectedSize}` : `You haven't selected any size`;

Code language: JavaScript (javascript)

## **Radio button’s change event**

When you check or uncheck a radio button, it fires the change event. To listen to the change event, you use the addEventListener() method like this:

radioButton.addEventListener('change',function(e){

});

Code language: JavaScript (javascript)

Inside the change event handler, you can access the this keyword to access the radio button. To check if the radio button is checked, you can use the checked property:

if(this.checked) {

*//*

}

Code language: JavaScript (javascript)

To get the value of the checked button, you use the value property:

if(this.checked) {

console.log(this.value);

}

Code language: JavaScript (javascript)

It’ll look like this:

radioButton.addEventListener('change', function (e) {

if (this.checked) {

console.log(this.value);

}

});

Code language: JavaScript (javascript)

The following example dynamically generates a radio group and shows the selected value when a radio button is checked:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript Radio Button**</title>**

**</head>**

**<body>**

**<p>**Select your size:**</p>**

**<div id="group">**

**</div>**

**<p id="output"></p>**

**<script>**

const sizes = ['XS', 'S', 'M', 'L', 'XL', 'XXL'];

*// generate the radio groups*

const group = document.querySelector("#group");

group.innerHTML = sizes.map((size) => `<div>

<input type="radio" name="size" value="${size}" id="${size}">

<label for="${size}">${size}</label>

</div>`).join(' ');

*// add an event listener for the change event*

const radioButtons = document.querySelectorAll('input[name="size"]');

for(const radioButton of radioButtons){

radioButton.addEventListener('change', showSelected);

}

function showSelected(e) {

console.log(e);

if (this.checked) {

document.querySelector('#output').innerText = `You selected ${this.value}`;

}

}

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works.

First, define an array of strings that hold the sizes. In practice, you may get these values from a database in the back-end or from the result of an API call:

const sizes = ['XS', 'S', 'M', 'L', 'XL', 'XXL'];

Code language: JavaScript (javascript)

Second, generate the radio groups from the elements of the sizes array:

const group = document.querySelector('#group');

group.innerHTML = sizes

.map(

(size) => `<div>

<input type="radio" name="size" value="${size}" id="${size}">

<label for="${size}">${size}</label>

</div>`

)

.join(' ');

Code language: JavaScript (javascript)

In this code:

1) Select the element with id #group.

2) Generate a radio group using the map() method with template literals; each array element is corresponding to a radio button HTML.

3) Join radio button HTML strings into an HTML string using the join() method.

4) Assign the HTML to the innerHTML of the output element.

Third, select all the radio buttons with the name size and add the change event listener:

const radioButtons = document.querySelectorAll('input[name="size"]');

for (const radioButton of radioButtons) {

radioButton.addEventListener('change', showSelected);

}

Code language: JavaScript (javascript)

Finally, define the change event handler:

function showSelected(e) {

if (this.checked) {

document.querySelector('#output').innerText = `You selected ${this.value}`;

}

}

Code language: JavaScript (javascript)

## **Summary**

* Use the <input> element with the type radio to create a radio button.
* Assign a name to multiple radio buttons to form a radio group. Only one radio button in the group can be selected.
* If the radio button is selected, its checked property is true.

# JavaScript select Element

**Summary**: in this tutorial, you will learn how to handle the <select> element in JavaScript.

## **Introduction to the HTML select elements**

A <select> element provides you with a list of options. A <select> element allows you to select one or multiple options.

To create a <select> element, you use the <select> and <option> elements. For example:

**<select id="framework">**

**<option value="1">**Angular**</option>**

**<option value="2">**React**</option>**

**<option value="3">**Vue.js**</option>**

**<option value="4">**Ember.js**</option>**

**</select>**

Code language: HTML, XML (xml)

AngularReactVue.jsEmber.js

The above <select> element allows you to select a single option at a time.

To enable multiple selections, you add multiple attribute to <select> element as follows:

**<select id="framework" multiple>**

**<option value="1">**Angular**</option>**

**<option value="2">**React**</option>**

**<option value="3">**Vue.js**</option>**

**<option value="4">**Ember.js**</option>**

**</select>**

Code language: HTML, XML (xml)

AngularReactVue.jsEmber.js

## **The HTMLSelectElement type**

To interact with <select> element in JavaScript, you use the HTMLSelectElement type.

The HTMLSelectElement type has the following useful properties:

* selectedIndex – returns the zero-based index of the selected option. The selectedIndex is -1 if no option is selected. If the <select> element allows multiple selections, the selectedIndex returns the value of the first option.
* value – returns the value property of the first selected option element if there is one. Otherwise, it returns an empty string.
* multiple – returns true if the <select> element allows multiple selections. It is equivalent to the multiple attribute.

### **The selectedIndex property**

To select a <select> element, you use the DOM API like [getElementById()](https://www.javascripttutorial.net/javascript-dom/javascript-getelementbyid/) or [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/).

The following example illustrates how to get the index of the selected option:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JavaScript Select Element Demo**</title>**

**<link href="css/selectbox.css" rel="stylesheet">**

**</head>**

**<body>**

**<form>**

**<label for="framework">**Select a JS Framework**</label>**

**<select id="framework">**

**<option value="1">**Angular**</option>**

**<option value="2">**React**</option>**

**<option value="3">**Vue.js**</option>**

**<option value="4">**Ember.js**</option>**

**</select>**

**<button id="btn">**Get the Selected Index**</button>**

**</form>**

**<script>**

const btn = document.querySelector('#btn');

const sb = document.querySelector('#framework')

btn.onclick = (event) => {

event.preventDefault();

*// show the selected index*

alert(sb.selectedIndex);

};

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, select the <button> and <select> elements using the querySelector() method.
* Then, attach a click event listener to the button and show the selected index using the [alert()](https://www.javascripttutorial.net/javascript-bom/javascript-alert/) method when the button is clicked.

### **The value property**

The value property of the <select> element depends on the <option> element and its HTML multiple attribute:

* If no option is selected, the value property of the select box is an empty string.
* If an option is selected and has a value attribute, the value property of the select box is the value of the selected option.
* If an option is selected and has no value attribute, the value property of the select box is the text of the selected option.
* If multiple options are selected, the value property of the select box is derived from the first selected option based on the previous two rules.

See the following example:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JavaScript Selected Value**</title>**

**<link href="css/selectbox.css" rel="stylesheet">**

**</head>**

**<body>**

**<div id="container">**

**<form>**

**<label for="framework">**Select a JS Framework:**</label>**

**<select id="framework">**

**<option value="">**Angular**</option>**

**<option value="1">**React**</option>**

**<option value="2">**Vue.js**</option>**

**<option>**Ember.js**</option>**

**</select>**

**<button id="btn">**Get the Selected Value**</button>**

**</form>**

**</div>**

**<script>**

const btn = document.querySelector('#btn');

const sb = document.querySelector('#framework')

btn.onclick = (event) => {

event.preventDefault();

alert(sb.value);

};

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In this example:

* If you select the first option, the value property of the <select> is empty.
* If you select the last option, the value property of the select box is Ember.js because the selected option has no value attribute.
* If you select the second or third option, the value property will be "1"or "2".

## **The HTMLOptionElement type**

In JavaScript, the HTMLOptionElement type represents the <option> element.

The HTMLOptionElement type has the following handy properties:

* index – the index of the option inside the collection of options.
* selected – returns true when the option is selected. You set this property to true to select an option.
* text – returns the option’s text.
* value – returns the HTML value attribute.

The <select> element has the options property that allows you to access the collection options:

selectBox.options

Code language: CSS (css)

For example, to access the text and value of the second option, you use the following:

const text = selectBox.options[1].text;

const value = selectBox.options[1].value;

Code language: JavaScript (javascript)

To get the selected option of a <select> element with a single selection, you use the following code:

let selectedOption = selectBox.options[selectBox.selectedIndex];

Code language: JavaScript (javascript)

Then you can access the text and value of the selected option via text and value properties:

const selectedText = selectedOption.text;

const selectedValue = selectedOption.value;

Code language: JavaScript (javascript)

When a <select> element allows multiple selections, you can use the selected property to determine which options are selected:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JavaScript Select Box**</title>**

**<link rel="stylesheet" href="css/selectbox.css">**

**</head>**

**<body>**

**<div id="container">**

**<form>**

**<label for="framework">**Select one or more JS Frameworks:**</label>**

**<select id="framework" multiple>**

**<option value="1">**Angular**</option>**

**<option value="2">**React**</option>**

**<option value="3">**Vue.js**</option>**

**<option value="4">**Ember.js**</option>**

**</select>**

**<button id="btn">**Get Selected Frameworks**</button>**

**</form>**

**</div>**

**<script>**

const btn = document.querySelector('#btn');

const sb = document.querySelector('#framework');

btn.onclick = (e) => {

e.preventDefault();

const selectedValues = [].filter

.call(sb.options, option => option.selected)

.map(option => option.text);

alert(selectedValues);

};

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In this example, the sb.options is an array-like object, so it doesn’t have the [filter()](https://www.javascripttutorial.net/javascript-array-filter/) methods like an [Array](https://www.javascripttutorial.net/javascript-array/) object.

To borrow these methods from the Array object, you use the [call()](https://www.javascripttutorial.net/javascript-call/) method. For example, the following returns an array of selected options:

[].filter.call(sb.options, option => option.selected)

Code language: PHP (php)

And to get the text property of the options, you chain the result of the filter() method with the [map()](https://www.javascripttutorial.net/javascript-array-map/) method, like this:

.map(option => option.text);

Code language: JavaScript (javascript)

## **Summary**

* The <select> element allows you to select one or multiple options. Add the multiple attribute to the <select> element to enable multiple selections.
* The HTMLSelectElement represents the <select> element. Use the selectedIndex and value to get the index and value of the selected option.
* The HTMLOptionElement represents the <option> element. If the option is selected, the selected property is true. The selectedText and selectedValue properties return the text and value of the selected option.

# JavaScript: Dynamically Add & Remove Options

**Summary**: in this tutorial, you will learn how to dynamically add options to and remove options from a [select box](https://www.javascripttutorial.net/javascript-dom/javascript-select-box/) in JavaScript.

The HTMLSelectElement type represents the <select> element. It has the add() method that dynamically adds an option to the <select> element and the remove() method that removes an option from the <select> element:

* add(option,existingOption) – adds a new <option> element to the <select> before an existing option.
* remove(index) – removes an option specified by the index from a <select>.

## **Adding options**

To add an option dynamically to a select box, you use these steps:

* First, create a new option.
* Second, add the option to the select box.

There are multiple ways to create an option dynamically and add it to a select box in JavaScript.

### **1) Using the Option constructor and add() method**

First, use the Option constructor to create a new option with the specified option text and value:

let newOption = new Option('Option Text','Option Value');

Code language: JavaScript (javascript)

Then, call the add() method of the select box object:

selectBox.add(newOption,undefined);

Code language: JavaScript (javascript)

The add() method accepts two arguments. The first argument is the new option and the second one is an existing option.

In this example, we pass undefined as the second argument, the method adds the new option to the end of the options list.

### **2) Using the DOM methods**

First, construct a new option using DOM methods:

*// create option using DOM*

const newOption = document.createElement('option');

const optionText = document.createTextNode('Option Text');

*// set option text*

newOption.appendChild(optionText);

*// and option value*

newOption.setAttribute('value','Option Value');

Code language: JavaScript (javascript)

Second, add the new option to the select box using the [appendChild()](https://www.javascripttutorial.net/javascript-dom/javascript-appendchild/) method:

*// add the option to the select box*

selectBox.appendChild(newOption);

Code language: JavaScript (javascript)

## **Removing Options**

There are also multiple ways to dynamically remove options from a select box.

The first way is to use the remove() method of the HTMLSelectElement type. The following  illustrates how to remove the first option:

selectBox.remove(0);

Code language: CSS (css)

The second way to remove an option is to reference the option by its index using the options collection and set its value to null:

selectBox.options[0] = null;

Code language: JavaScript (javascript)

The third way is to use the [removeChild()](https://www.javascripttutorial.net/javascript-dom/javascript-removechild/) method and remove a specified option. The following code removes the first element of a the selectBox:

*// remove the first element:*

selectBox.removeChild(selectBox.options[0]);

Code language: JavaScript (javascript)

To remove all options of a select box, you use the following code:

function removeAll(selectBox) {

while (selectBox.options.length > 0) {

selectBox.remove(0);

}

}

Code language: JavaScript (javascript)

When you remove the first option, the select box moves another option as the first option. The removeAll() function repeatedly removes the first option in the select box, therefore, it removes all the options.

## **Put it all together**

We’ll build a small application that allows users to add a new option from the value of an input text and to remove one or more selected options:

<!DOCTYPE html>

**<html>**

**<head>**

**<title>**JavaScript Selected Value**</title>**

**<link href="css/selectbox.css" rel="stylesheet">**

**</head>**

**<body>**

**<div id="container">**

**<form>**

**<label for="name">**Framework:**</label>**

**<input type="text" id="name" placeholder="Enter a framework" autocomplete="off">**

**<button id="btnAdd">**Add**</button>**

**<label for="list">**Framework List:**</label>**

**<select id="list" name="list" multiple>**

**</select>**

**<button id="btnRemove">**Remove Selected Framework**</button>**

**</form>**

**</div>**

**<script>**

const btnAdd = document.querySelector('#btnAdd');

const btnRemove = document.querySelector('#btnRemove');

const sb = document.querySelector('#list');

const name = document.querySelector('#name');

btnAdd.onclick = (e) => {

e.preventDefault();

*// validate the option*

if (name.value == '') {

alert('Please enter the name.');

return;

}

*// create a new option*

const option = new Option(name.value, name.value);

*// add it to the list*

sb.add(option, undefined);

*// reset the value of the input*

name.value = '';

name.focus();

};

*// remove selected option*

btnRemove.onclick = (e) => {

e.preventDefault();

*// save the selected option*

let selected = [];

for (let i = 0; i < sb.options.length; i++) {

selected[i] = sb.options[i].selected;

}

*// remove all selected option*

let index = sb.options.length;

while (index--) {

if (selected[index]) {

sb.remove(index);

}

}

};

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

First, use the [querySelector()](https://www.javascripttutorial.net/javascript-dom/javascript-queryselector/) method to select elements including the input text, button, and selection box:

const name = document.querySelector('#name');

const btnAdd = document.querySelector('#btnAdd');

const btnRemove = document.querySelector('#btnRemove');

const sb = document.querySelector('#list');

Code language: JavaScript (javascript)

Second, attach the click event listener to the btnAdd button.

If the value of the input text is blank, we show an [alert](https://www.javascripttutorial.net/javascript-bom/javascript-alert/) to inform the users that the name is required.

Otherwise, we create a new option and add it to the selection box.

After adding the option, we reset the entered text of the input text and set the focus to it.

btnAdd.onclick = (e) => {

e.preventDefault();

*// validate the option*

if (name.value == '') {

alert('Please enter the name.');

return;

}

*// create a new option*

const option = new Option(name.value, name.value);

*// add it to the list*

sb.add(option, undefined);

*// reset the value of the input*

name.value = '';

name.focus();

};

Code language: JavaScript (javascript)

Third, register a click event listener to the btnRemove button. In the event listener, we save the selected options in an array first and remove each of them then.

*// remove selected option*

btnRemove.onclick = (e) => {

e.preventDefault();

*// save the selected option*

let selected = [];

for (let i = 0; i < sb.options.length; i++) {

selected[i] = sb.options[i].selected;

}

*// remove all selected option*

let index = sb.options.length;

while (index--) {

if (selected[index]) {

sb.remove(index);

}

}

};

Code language: JavaScript (javascript)

In this tutorial, you have learned how to add options to and remove options from a select box using JavaScript.

# JavaScript change Event

**Summary**: in this tutorial, you’ll learn about the JavaScript change event of the input text, radio button, checkbox, and select elements.

The change event occurs when the element has completed changing.

To attach an event handler to the change event of an element, you can either call the [addEventListener()](https://www.javascripttutorial.net/dom/events/add-an-event-handler/) method:

element.addEventListener('change', function(){

*// handle change*

});

Code language: JavaScript (javascript)

or use the onchange attribute of the element. For example:

**<input type="text" onchange="changeHandler()">**

Code language: HTML, XML (xml)

However, it is a good practice to use the addEventListener() method.

## **Using JavaScript change event for input elements**

The change event of an <input> element fires when the <input> element loses focus. The change event does not fire when you’re tying.

The following example shows the value of the input text when it loses focus.

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript change Event for Input Element**</title>**

**</head>**

**<body>**

**<label for="message">**Message:**</label>**

**<input type="text" class="input" id="message" name="message">**

**<button>**Submit**</button>**

**<p id="result"></p>**

**<script>**

let input = document.querySelector('.input');

let result = document.querySelector('#result');

input.addEventListener('change', function () {

result.textContent = this.value;

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

In this example, if you type some text on the <input> element and move focus to the button, the change event fires to show the entered text.

Note that if you want to handle every change of the value, you use the input event instead.

## **Using JavaScript change event for radio buttons**

A radio button fires the change event after you select it.

The following example shows how to handle the change event of the radio buttons:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript change Event for Radio Buttons**</title>**

**</head>**

**<body>**

**<span>**Status:**</span>**

**<input type="radio" id="pending" name="status">**

**<label for="pending">**Pending**</label>**

**<input type="radio" id="resolved" name="status">**

**<label for="resolved">**Resolved**</label>**

**<input type="radio" id="rejected" name="status">**

**<label for="rejected">**Rejected**</label>**

**<p id="result"></p>**

**<script>**

let result = document.querySelector('#result');

document.body.addEventListener('change', function (e) {

let target = e.target;

let message;

switch (target.id) {

case 'pending':

message = 'The Pending radio button changed';

break;

case 'resolved':

message = 'The Resolved radio button changed';

break;

case 'rejected':

message = 'The Rejected radio button changed';

break;

}

result.textContent = message;

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, register an event handler to the change event of the body. When a radio button is clicked, its change event is bubbled to the body. This technique is called [event delegation](https://www.javascripttutorial.net/javascript-dom/javascript-event-delegation/).
* Then, show a corresponding message based on which radio button is selected.

## **Using JavaScript change event for checkboxes**

Similar to radio buttons, checkboxes fire the change event after selection, whether checked or unchecked. For example:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript change Event for Checkboxes**</title>**

**</head>**

**<body>**

**<label for="status">**Web Technology:**</label>**

**<input type="checkbox" id="html">** HTML

**<input type="checkbox" id="css">** CSS

**<input type="checkbox" id="js">** JavaScript

**<p id="result"></p>**

**<script>**

let result = document.querySelector('#result');

document.body.addEventListener('change', function (e) {

let target = e.target;

let message;

switch (target.id) {

case 'html':

message = 'The HTML checkbox changed';

break;

case 'css':

message = 'The CSS checkbox changed';

break;

case 'js':

message = 'The JavaScript checkbox changed';

break;

}

result.textContent = message;

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

## **Using JavaScript change event for the select element**

The <select> element fires the change event once the selection has completed.

The following example shows how to handle the change event of the <select> element. The <p> element with the id result will display the selected item:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript change Event for Select element**</title>**

**</head>**

**<body>**

**<select id="lang">**

**<option value="">**Select a language**</option>**

**<option value="JavaScript">**JavaScript**</option>**

**<option value="TypeScript">**TypeScript**</option>**

**<option value="PHP">**PHP**</option>**

**<option value="Python">**Python**</option>**

**<option value="Java">**Java**</option>**

**</select>**

**<p id="result"></p>**

**<script>**

let select = document.querySelector('#lang');

let result = document.querySelector('#result');

select.addEventListener('change', function () {

result.textContent = this.value;

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, select the <select> element by its id (lang);
* Then, show the selected value in the <p> element.

## **Summary**

* The <input> element fires the change event once it loses focus.
* The radio button, checkbox, and select elements fire the change event after they have been selected.

# JavaScript input Event

**Summary**: in this tutorial, you’ll learn about the JavaScript input event that fires whenever the value of the <input>, <select>And <textarea> changes.

## **Introduction to the JavaScript input event**

The input event fires every time whenever the value of the <input>, <select>, or <textarea> element changes.

Unlike the [change](https://www.javascripttutorial.net/javascript-dom/javascript-change-event/) event that only fires when the value is committed, the input event fires whenever the value changes.

For example, if you’re typing on the <input> element, the element fire the input event continuously. However, the change event only fires when the <input> element loses focus.

The following example illustrates the input event of the <input> element:

<!DOCTYPE html>

**<html lang="en">**

**<head>**

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

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

**<title>**JavaScript input Event Demo**</title>**

**</head>**

**<body>**

**<label for="message">**Message**</label>**

**<input placeholder="Enter some text" id="message" name="message">**

**<p id="result"></p>**

**<script>**

const message = document.querySelector('#message');

const result = document.querySelector('#result');

message.addEventListener('input', function () {

result.textContent = this.value;

});

**</script>**

**</body>**

**</html>**

Code language: HTML, XML (xml)

How it works:

* First, select the <input> element with the id message and the <p> element with the id result.
* Then, attach an event handler to the input event of the <input> element. Inside the input event handler, update the textContent property of the <p> element.