<https://www.w3schools.com/js/js_htmldom.asp>

**JavaScript HTML DOM**

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

The HTML DOM (Document Object Model)

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

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



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

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

**JavaScript - HTML DOM Methods**

The following example changes the content (the innerHTML) of the <p> element with id="demo":

<html>

<body>

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

<script>

document.getElementById("demo").innerHTML = "Hello World!";

</script>

</body>

</html>

In the example above, getElementById is a method, while innerHTML is a property.

**The getElementById Method**

The most common way to access an HTML element is to use the id of the element.

In the example above the getElementById method used id="demo" to find the element.

**The innerHTML Property**

The easiest way to get the content of an element is by using the innerHTML property.

The innerHTML property is useful for getting or replacing the content of HTML elements.

**The HTML DOM Document Object**

The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

**Finding HTML Elements**

**Method Description**

document.getElementById(id) Find an element by element id

document.getElementsByTagName(name) Find elements by tag name

document.getElementsByClassName(name) Find elements by class name

**Changing HTML Elements**

**Property Description**

element.innerHTML = new html content Change the inner HTML of an element

element.attribute = new value Change the attribute value of an HTML element

element.style.property = new style Change the style of an HTML element

**Method Description**

element.setAttribute(attribute, value) Change the attribute value of an HTML element

**Adding and Deleting Elements**

**Method Description**

document.createElement(element) Create an HTML element

document.removeChild(element) Remove an HTML element

document.appendChild(element) Add an HTML element

document.replaceChild(new, old) Replace an HTML element

document.write(text) Write into the HTML output stream

**Adding Events Handlers**

**Method Description**

document.getElementById(id).onclick = function(){code} Adding event handler code to an onclick event

**Finding HTML Objects**

Property Description DOM

document.anchors Returns all <a> elements that have a name attribute 1

document.applets Deprecated 1

document.baseURI Returns the absolute base URI of the document 3

document.body Returns the <body> element 1

document.cookie Returns the document's cookie 1

document.doctype Returns the document's doctype 3

document.documentElement Returns the <html> element 3

document.documentMode Returns the mode used by the browser 3

document.documentURI Returns the URI of the document 3

document.domain Returns the domain name of the document server 1

document.domConfig Obsolete. 3

document.embeds Returns all <embed> elements 3

document.forms Returns all <form> elements 1

document.head Returns the <head> element 3

document.images Returns all <img> elements 1

document.implementation Returns the DOM implementation 3

document.inputEncoding Returns the document's encoding (character set) 3

document.lastModified Returns the date and time the document was updated 3

document.links Returns all <area> and <a> elements that have a href attribute 1

document.readyState Returns the (loading) status of the document 3

document.referrer Returns the URI of the referrer (the linking document) 1

document.scripts Returns all <script> elements 3

document.strictErrorChecking Returns if error checking is enforced 3

document.title Returns the <title> element 1

document.URL Returns the complete URL of the document 1

**JavaScript HTML DOM Elements**

**Finding HTML Elements**

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are several ways to do this:

Finding HTML elements by id

Finding HTML elements by tag name

Finding HTML elements by class name

Finding HTML elements by CSS selectors

Finding HTML elements by HTML object collections

**Finding HTML Element by Id**

const element = document.getElementById("intro");

**Finding HTML Elements by Tag Name**

This example finds all <p> elements:

const element = document.getElementsByTagName("p");

**Finding HTML Elements by Class Name**

If you want to find all HTML elements with the same class name, use getElementsByClassName().

This example returns a list of all elements with class="intro".

const x = document.getElementsByClassName("intro");

**Finding HTML Elements by CSS Selectors**

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

This example returns a list of all <p> elements with class="intro".

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Query Selector</p>

<p class="intro">Hello World!.</p>

<p class="intro">This example demonstrates the <b>querySelectorAll</b> method.</p>

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

<script>

const x = document.querySelectorAll("p.intro");

document.getElementById("demo").innerHTML =

'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;

</script>

**Finding HTML Elements by HTML Object Collections**

This example finds the form element with id="frm1", in the forms collection, and displays all element values:

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements Using <b>document.forms</b>.</p>

<form id="frm1" action="/action\_page.php">

First name: <input type="text" name="fname" value="Donald"><br>

Last name: <input type="text" name="lname" value="Duck"><br><br>

<input type="submit" value="Submit">

</form>

<p>These are the values of each element in the form:</p>

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

<script>

const x = document.forms["frm1"];

let text = "";

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

text += x.elements[i].value + "<br>";

}

document.getElementById("demo").innerHTML = text;

</script>

**The following HTML objects (and object collections) are also accessible:**

**document.anchors**

<h2>Finding HTML Elements Using document.anchors</h2>

<a name="html">HTML Tutorial</a><br>

<a name="css">CSS Tutorial</a><br>

<a name="xml">XML Tutorial</a><br>

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

<script>

document.getElementById("demo").innerHTML =

"Number of anchors are: " + document.anchors.length;

</script>

**document.body**

<h2>JavaScript HTMLDOM</h2>

<p>Displaying document.body</p>

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

<script>

document.getElementById("demo").innerHTML = document.body.innerHTML;

</script>

**document.documentElement**

<h2>JavaScript HTMLDOM</h2>

<p>Displaying document.documentElement</p>

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

<script>

document.getElementById("demo").innerHTML = document.documentElement.innerHTML;

</script>

**document.embeds**

<h2>JavaScript HTMLDOM</h2>

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

<script>

document.getElementById("demo").innerHTML =

"Number of embeds: " + document.embeds.length;

</script>

**document.forms**

<h2>Finding HTML Elements Using document.forms</h2>

<form action="">

First name: <input type="text" name="fname" value="Donald">

<input type="submit" value="Submit">

</form>

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

<script>

document.getElementById("demo").innerHTML =

"Number of forms: " + document.forms.length;

</script>

**document.head**

<head>

<title>W3Schools Demo</title>

</head>

<body>

<h2>JavaScript HTMLDOM</h2>

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

<script>

document.getElementById("demo").innerHTML =

document.head;

</script>

**document.images**

<h2>Finding HTML Elements Using document.images</h2>

<img src="pic\_htmltree.gif" width="486" height="266">

<img src="pic\_navigate.gif" width="362" height="255">

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

<script>

document.getElementById("demo").innerHTML =

"Number of images: " + document.images.length;

</script>

**document.links**

<h2>Finding HTML Elements Using document.links</h2>

<p>

<a href="/html/default.asp">HTML</a>

<br>

<a href="/css/default.asp">CSS</a>

</p>

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

<script>

document.getElementById("demo").innerHTML =

"Number of links: " + document.links.length;

</script>

**document.scripts**

<h2>Finding HTML Elements Using document.scripts</h2>

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

<script>

document.getElementById("demo").innerHTML =

"Number of scripts: " + document.scripts.length;

</script>

**document.title**

<head>

<title>W3Schools Demo</title>

</head>

<body>

<h2>Finding HTML Elements Using document.title</h2>

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

<script>

document.getElementById("demo").innerHTML =

"The title of this document is: " + document.title;

</script>

**Changing HTML Content**

The easiest way to modify the content of an HTML element is by using the innerHTML property.

To change the content of an HTML element, use this syntax:

document.getElementById(*id*).innerHTML =*new HTML*

**Changing the Value of an Attribute**

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

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

**Dynamic HTML content**

JavaScript can create dynamic HTML content:

Date : Wed Apr 27 2022 08:34:27 GMT-0300 (hora estándar de Argentina)

document.getElementById("demo").innerHTML = "Date : " + Date();

**document.write()**

In JavaScript, document.write() can be used to write directly to the HTML output stream:

Never use document.write() after the document is loaded. It will overwrite the document.

**JavaScript Forms**

**JavaScript Form Validation**

HTML form validation can be done by JavaScript.

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

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

<form name="myForm" action="/action\_page.php" **onsubmit="return validateForm()"** method="post">  
Name: <input type="text" name="fname">  
<input type="submit" value="Submit">  
</form>

**JavaScript Can Validate Numeric Input**

JavaScript is often used to validate numeric input:

<body>

<h2>JavaScript Validation</h2>

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

<input id="numb">

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

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

<script>

function myFunction() {

// Get the value of the input field with id="numb"

let x = document.getElementById("numb").value;

// If x is Not a Number or less than one or greater than 10

let text;

if (isNaN(x) || x < 1 || x > 10) {

text = "Input not valid";

} else {

text = "Input OK";

}

document.getElementById("demo").innerHTML = text;

}

</script>

</body>

**Automatic HTML Form Validation**

HTML form validation can be performed automatically by the browser:

If a form field (fname) is empty, the required attribute prevents this form from being submitted:

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

**Constraint Validation HTML Input Attributes**

**Attribute Description**

disabled Specifies that the input element should be disabled

max Specifies the maximum value of an input element

min Specifies the minimum value of an input element

pattern Specifies the value pattern of an input element

required Specifies that the input field requires an element

type Specifies the type of an input element

Example

Set a max date, a min date, and a range of legal values:

<form>  
  <label for="datemax">Enter a date before 1980-01-01:</label>  
  <input type="date" id="datemax" name="datemax" max="1979-12-31"><br><br>  
  
  <label for="datemin">Enter a date after 2000-01-01:</label>  
  <input type="date" id="datemin" name="datemin" min="2000-01-02"><br><br>  
  
  <label for="quantity">Quantity (between 1 and 5):</label>  
  <input type="number" id="quantity" name="quantity" min="1" max="5">  
</form>

**The size Attribute**

The input size attribute specifies the visible width, in characters, of an input field.

The default value for size is 20.

Note: The size attribute works with the following input types: text, search, tel, url, email, and

password.

<form>  
  <label for="fname">First name:</label><br>  
  <input type="text" id="fname" name="fname" size="50"><br>  
  <label for="pin">PIN:</label><br>  
  <input type="text" id="pin" name="pin" size="4">  
</form>

**The maxlength Attribute**

The input maxlength attribute specifies the maximum number of characters allowed in an input field.

<form>  
  <label for="fname">First name:</label><br>  
  <input type="text" id="fname" name="fname" size="50"><br>  
  <label for="pin">PIN:</label><br>  
  <input type="text" id="pin" name="pin" maxlength="4" size="4">  
</form>

**The min and max Attributes**

The input min and max attributes specify the minimum and maximum values for an input field.

The min and max attributes work with the following input types: number, range, date, datetime-local, month, time and week.

Example

Set a max date, a min date, and a range of legal values:

<form>  
  <label for="datemax">Enter a date before 1980-01-01:</label>  
  <input type="date" id="datemax" name="datemax" max="1979-12-31"><br><br>  <label for="datemin">Enter a date after 2000-01-01:</label>  
  <input type="date" id="datemin" name="datemin" min="2000-01-02"><br><br>  <label for="quantity">Quantity (between 1 and 5):</label>  
  <input type="number" id="quantity" name="quantity" min="1" max="5">  
</form>

**The multiple Attribute**

The input multiple attribute specifies that the user is allowed to enter more than one value in an input field.

The multiple attribute works with the following input types: email, and file.

<form>  
  <label for="files">Select files:</label>  
  <input type="file" id="files" name="files" multiple>  
</form>

**The pattern Attribute**

The input pattern attribute specifies a regular expression that the input field's value is checked against, when the form is submitted.

The pattern attribute works with the following input types: text, date, search, url, tel, email, and password.

Tip: Use the global title attribute to describe the pattern to help the user.

<form>  
  <label for="country\_code">Country code:</label>  
  <input type="text" id="country\_code" name="country\_code"  
  pattern="[A-Za-z]{3}" title="Three letter country code">  
</form>

**The placeholder Attribute**

The input placeholder attribute specifies a short hint that describes the expected value of an input field (a sample value or a short description of the expected format).

The short hint is displayed in the input field before the user enters a value.

The placeholder attribute works with the following input types: text, search, url, tel, email, and password.

<form>  
  <label for="phone">Enter a phone number:</label>  
  <input type="tel" id="phone" name="phone"  
  placeholder="123-45-678"  
  pattern="[0-9]{3}-[0-9]{2}-[0-9]{3}">  
</form>

**The required Attribute**

The input required attribute specifies that an input field must be filled out before submitting the form.

The required attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

<form>  
  <label for="username">Username:</label>  
  <input type="text" id="username" name="username" required>  
</form>

**The step Attribute**

The input step attribute specifies the legal number intervals for an input field.

Example: if step="3", legal numbers could be -3, 0, 3, 6, etc.

Tip: This attribute can be used together with the max and min attributes to create a range of legal values.

The step attribute works with the following input types: number, range, date, datetime-local, month, time and week.

<form>  
  <label for="points">Points:</label>  
  <input type="number" id="points" name="points" step="3">  
</form>

**The autofocus Attribute**

The input autofocus attribute specifies that an input field should automatically get focus when the page loads.

<form>  
  <label for="fname">First name:</label><br>  
  <input type="text" id="fname" name="fname" autofocus><br>  
  <label for="lname">Last name:</label><br>  
  <input type="text" id="lname" name="lname">  
</form>

**The height and width Attributes**

The input height and width attributes specify the height and width of an <input type="image"> element.

<form>  
  <label for="fname">First name:</label>  
  <input type="text" id="fname" name="fname"><br><br>  
  <label for="lname">Last name:</label>  
  <input type="text" id="lname" name="lname"><br><br>  
  <input type="image" src="img\_submit.gif" alt="Submit" width="48" height="48">  
</form>

**The list Attribute**

The input list attribute refers to a <datalist> element that contains pre-defined options for an <input> element.

<form>  
  <input list="browsers">  
  <datalist id="browsers">  
    <option value="Internet Explorer">  
    <option value="Firefox">  
    <option value="Chrome">  
    <option value="Opera">  
    <option value="Safari">  
  </datalist>  
</form>

**The autocomplete Attribute**

The input autocomplete attribute specifies whether a form or an input field should have autocomplete on or off.

Autocomplete allows the browser to predict the value. When a user starts to type in a field, the browser should display options to fill in the field, based on earlier typed values.

The autocomplete attribute works with <form> and the following <input> types: text, search, url, tel, email, password, datepickers, range, and color.

<form action="/action\_page.php" autocomplete="on">  
  <label for="fname">First name:</label>  
  <input type="text" id="fname" name="fname"><br><br>  
  <label for="lname">Last name:</label>  
  <input type="text" id="lname" name="lname"><br><br>  
  <label for="email">Email:</label>  
  <input type="email" id="email" name="email" autocomplete="off"><br><br>  
  <input type="submit" value="Submit">  
</form>

**Constraint Validation CSS Pseudo Selectors**

**Selector Description**

:disabled Selects input elements with the "disabled" attribute specified

:invalid Selects input elements with invalid values

:optional Selects input elements with no "required" attribute specified

:required Selects input elements with the "required" attribute specified

:valid Selects input elements with valid values

**JavaScript HTML DOM - Changing CSS**

Changing HTML Style

To change the style of an HTML element, use this syntax:

document.getElementById(*id*).style.*property*=*new style*

**JavaScript HTML DOM Animation**

<!DOCTYPE html>

<html>

<style>

#container {

width: 400px;

height: 400px;

position: relative;

background: yellow;

}

#animate {

width: 50px;

height: 50px;

position: absolute;

background-color: red;

}

</style>

<body>

<p><button onclick="myMove()">Click Me</button></p>

<div id ="container">

<div id ="animate"></div>

</div>

<script>

function myMove() {

let id = null;

const elem = document.getElementById("animate");

let pos = 0;

clearInterval(id);

id = setInterval(frame, 5);

function frame() {

if (pos == 350) {

clearInterval(id);

} else {

pos++;

elem.style.top = pos + "px";

elem.style.left = pos + "px";

}

}

}

</script>

</body>

</html>

**JavaScript HTML DOM Events**

**Using Events**

The HTML DOM allows you to execute code when an event occurs.

Events are generated by the browser when "things happen" to HTML elements:

* An element is clicked on
* The page has loaded
* Input fields are changed

<body>  
<h1 id="id1">My Heading 1</h1>  
<button type="button"  
onclick="document.getElementById('id1').style.color = 'red'">  
Click Me!</button>  
</body>

**Reacting to Events**

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

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

onclick=*JavaScript*

Examples of HTML events:

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

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>

function changeText(id) {

id.innerHTML = "Ooops!";

}

</script>

**HTML Event Attributes**

To assign events to HTML elements you can use event attributes.

<body>

<h2>JavaScript HTML Events</h2>

<p>Click the button to display the date.</p>

<button onclick="displayDate()">The time is?</button>

<script>

function displayDate() {

document.getElementById("demo").innerHTML = Date();

}

</script>

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

</body>

**Assign Events Using the HTML DOM**

The HTML DOM allows you to assign events to HTML elements using JavaScript:

<body>

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

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

<script>

document.getElementById("myBtn").onclick = displayDate;

function displayDate() {

document.getElementById("demo").innerHTML = Date();

}

</script>

</body>

**The onload and onunload Events**

The onload and onunload events are triggered when the user enters or leaves the page.

The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

The onload and onunload events can be used to deal with cookies.

<body onload="checkCookies()">

<h2>JavaScript HTML Events</h2>

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

<script>

function checkCookies() {

var text = "";

if (navigator.cookieEnabled == true) {

text = "Cookies are enabled.";

} else {

text = "Cookies are not enabled.";

}

document.getElementById("demo").innerHTML = text;

}

</script>

</body>

**The onchange Event**

The onchange event is often used in combination with validation of input fields.

Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

<body>

<h2>JavaScript HTML Events</h2>

Enter your name: <input type="text" id="fname" onchange="upperCase()">

<p>When you leave the input field, a function is triggered which transforms the input text to upper case.</p>

<script>

function upperCase() {

const x = document.getElementById("fname");

x.value = x.value.toUpperCase();

}

</script>

**The onmouseover and onmouseout Events**

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

<div onmouseover="mOver(this)" onmouseout="mOut(this)"

style="background-color:#D94A38;width:120px;height:20px;padding:40px;">

Mouse Over Me</div>

<script>

function mOver(obj) {

obj.innerHTML = "Thank You"

}

function mOut(obj) {

obj.innerHTML = "Mouse Over Me"

}

</script>

**The onmousedown, onmouseup and onclick Events**

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

<div onmousedown="mDown(this)" onmouseup="mUp(this)"

style="background-color:#D94A38;width:90px;height:20px;padding:40px;">

Click Me</div>

<script>

function mDown(obj) {

obj.style.backgroundColor = "#1ec5e5";

obj.innerHTML = "Release Me";

}

function mUp(obj) {

obj.style.backgroundColor="#D94A38";

obj.innerHTML="Thank You";

}

</script>

**onmousedown and onmouseup**

Change an image when a user holds down the mouse button.

<script>

function lighton() {

document.getElementById('myimage').src = "bulbon.gif";

}

function lightoff() {

document.getElementById('myimage').src = "bulboff.gif";

}

</script>

</head>

<body>

<img id="myimage" onmousedown="lighton()" onmouseup="lightoff()" src="bulboff.gif" width="100" height="180" />

<p>Click mouse and hold down!</p>

**onload**

Display an alert box when the page has finished loading.

<script>

function mymessage() {

alert("This message was triggered from the onload event");

}

</script>

</head>

<body onload="mymessage()">

</body>

**onfocus**

Change the background-color of an input field when it gets focus.

<script>

function myFunction(x) {

x.style.background = "yellow";

}

</script>

</head>

<body>

Enter your name: <input type="text" onfocus="myFunction(this)">

<p>When the input field gets focus, a function is triggered which changes the background-color.</p>

</body>

Mouse Events

Change the color of an element when the cursor moves over it.

<h1 onmouseover="style.color='red'"

onmouseout="style.color='black'">

Mouse over this text</h1>

**HTML DOM Event Object Reference -** <https://www.w3schools.com/jsref/dom_obj_event.asp>

**JavaScript HTML DOM EventListener**

The addEventListener() method

document.getElementById("myBtn").addEventListener("click", displayDate);

<h2>JavaScript addEventListener()</h2>

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

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

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

<script>

document.getElementById("myBtn").addEventListener("click", displayDate);

function displayDate() {

document.getElementById("demo").innerHTML = Date();

}

</script>

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

The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The addEventListener() method makes it easier to control how the event reacts to bubbling.

When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the removeEventListener() method.

Syntax

element.addEventListener(event, function, useCapture);

The first parameter is the type of the event (like "click" or "mousedown" or any other HTML DOM Event.) <https://www.w3schools.com/jsref/dom_obj_event.asp>

The second parameter is the function we want to call when the event occurs.

The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

Note that you don't use the "on" prefix for the event; use "click" instead of "onclick".

**Add an Event Handler to an Element**

element.addEventListener("click", function(){ alert("Hello World!"); });

<h2>JavaScript addEventListener()</h2>

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

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

<script>

document.getElementById("myBtn").addEventListener("click", function() {

alert("Hello World!");

});

</script>

**Add Many Event Handlers to the Same Element**

The addEventListener() method allows you to add many events to the same element, without overwriting existing events:

**<body>**

<h2>JavaScript addEventListener()</h2>

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

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

<script>

var x = document.getElementById("myBtn");

x.addEventListener("click", myFunction);

x.addEventListener("click", someOtherFunction);

function myFunction() {

alert ("Hello World!");

}

function someOtherFunction() {

alert ("This function was also executed!");

}

</script>

**</body>**

**<body>**

<h2>JavaScript addEventListener()</h2>

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

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

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

<script>

var x = document.getElementById("myBtn");

x.addEventListener("mouseover", myFunction);

x.addEventListener("click", mySecondFunction);

x.addEventListener("mouseout", myThirdFunction);

function myFunction() {

document.getElementById("demo").innerHTML += "Moused over!<br>";

}

function mySecondFunction() {

document.getElementById("demo").innerHTML += "Clicked!<br>";

}

function myThirdFunction() {

document.getElementById("demo").innerHTML += "Moused out!<br>";

}

</script>

**</body>**

**Add an Event Handler to the window Object**

The addEventListener() method allows you to add event listeners on any HTML DOM object such as HTML elements, the HTML document, the window object, or other objects that support events, like the xmlHttpRequest object.

**<body>**

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method on the window object.</p>

<p>Try resizing this browser window to trigger the "resize" event handler.</p>

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

<script>

window.addEventListener("resize", function(){

document.getElementById("demo").innerHTML = Math.random();

});

</script>

**</body>**

**Passing Parameters**

**<body>**

<h2>JavaScript addEventListener()</h2>

<p>This example demonstrates how to pass parameter values when using the addEventListener() method.</p>

<p>Click the button to perform a calculation.</p>

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

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

<script>

let p1 = 5;

let p2 = 7;

document.getElementById("myBtn").addEventListener("click", function() {

myFunction(p1, p2);

});

function myFunction(a, b) {

document.getElementById("demo").innerHTML = a \* b;

}

</script>

**</body>**

**Event Bubbling or Event Capturing?**

There are two ways of event propagation in the HTML DOM, bubbling and capturing.

Event propagation is a way of defining the element order when an event occurs. If you have a <p> element inside a <div> element, and the user clicks on the <p> element, which element's "click" event should be handled first?

In bubbling the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.

In capturing the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

With the addEventListener() method you can specify the propagation type by using the "useCapture" parameter:

addEventListener(event, function, useCapture);

**<!DOCTYPE html>**

<html>

<head>

<style>

#myDiv1, #myDiv2 {

background-color: coral;

padding: 50px;

}

#myP1, #myP2 {

background-color: white;

font-size: 20px;

border: 1px solid;

padding: 20px;

}

</style>

<meta content="text/html; charset=utf-8" http-equiv="Content-Type">

</head>

<body>

<h2>JavaScript addEventListener()</h2>

<div id="myDiv1">

<h2>Bubbling:</h2>

<p id="myP1">Click me!</p>

</div><br>

<div id="myDiv2">

<h2>Capturing:</h2>

<p id="myP2">Click me!</p>

</div>

<script>

document.getElementById("myP1").addEventListener("click", function() {

alert("You clicked the white element!");

}, false);

document.getElementById("myDiv1").addEventListener("click", function() {

alert("You clicked the orange element!");

}, false);

document.getElementById("myP2").addEventListener("click", function() {

alert("You clicked the white element!");

}, true);

document.getElementById("myDiv2").addEventListener("click", function() {

alert("You clicked the orange element!");

}, true);

</script>

</body>

**</html>**

**The removeEventListener() method**

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

**<!DOCTYPE html>**

<html>

<head>

<style>

#myDIV {

background-color: coral;

border: 1px solid;

padding: 50px;

color: white;

font-size: 20px;

}

</style>

</head>

<body>

<h2>JavaScript removeEventListener()</h2>

<div id="myDIV">

<p>This div element has an onmousemove event handler that displays a random number every time you move your mouse inside this orange field.</p>

<p>Click the button to remove the div's event handler.</p>

<button onclick="removeHandler()" id="myBtn">Remove</button>

</div>

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

<script>

document.getElementById("myDIV").addEventListener("mousemove", myFunction);

function myFunction() {

document.getElementById("demo").innerHTML = Math.random();

}

function removeHandler() {

document.getElementById("myDIV").removeEventListener("mousemove", myFunction);

}

</script>

</body>

**</html>**

**JavaScript HTML DOM Navigation**

DOM Nodes

* The entire document is a document node
* Every HTML element is an element node
* The text inside HTML elements are text nodes
* Every HTML attribute is an attribute node (deprecated)
* All comments are comment nodes



**Node Relationships**

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

* In a node tree, the top node is called the root (or root node)
* Every node has exactly one parent, except the root (which has no parent)
* A node can have a number of children
* Siblings (brothers or sisters) are nodes with the same parent

|  |  |
| --- | --- |
| <html>    <head>     <title>DOM Tutorial</title>   </head>    <body>     <h1>DOM Lesson one</h1>     <p>Hello world!</p>   </body>  </html> | Node tree |

From the HTML above you can read:

* <html> is the root node
* <html> has no parents
* <html> is the parent of <head> and <body>
* <head> is the first child of <html>
* <body> is the last child of <html>

and:

* <head> has one child: <title>
* <title> has one child (a text node): "DOM Tutorial"
* <body> has two children: <h1> and <p>
* <h1> has one child: "DOM Lesson one"
* <p> has one child: "Hello world!"
* <h1> and <p> are siblings

**Navigating Between Nodes**

You can use the following node properties to navigate between nodes with JavaScript:

* parentNode
* childNodes[nodenumber]
* firstChild
* lastChild
* nextSibling
* previousSibling

**Child Nodes and Node Values**

<title id="demo">DOM Tutorial</title>

The element node <title> (in the example above) does not contain text.

It contains a text node with the value "DOM Tutorial".

The value of the text node can be accessed by the node's innerHTML property:

myTitle = document.getElementById("demo").innerHTML;

Accessing the innerHTML property is the same as accessing the nodeValue of the first child:

myTitle = document.getElementById("demo").firstChild.nodeValue;

Accessing the first child can also be done like this:

myTitle = document.getElementById("demo").childNodes[0].nodeValue;

**DOM Root Nodes**

There are two special properties that allow access to the full document:

* document.body - The body of the document
* document.documentElement - The full document

<html>  
<body>  
<h2>JavaScript HTMLDOM</h2>  
<p>Displaying document.body</p>  
<p id="demo"></p>  
<script>  
document.getElementById("demo").innerHTML = document.body.innerHTML;  
</script>  
</body>  
</html>

<html>  
<body>  
  
<h2>JavaScript HTMLDOM</h2>  
<p>Displaying document.documentElement</p>  
  
<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = document.documentElement.innerHTML;  
</script>  
  
</body>  
</html>

**The nodeName Property**

The nodeName property specifies the name of a node.

* nodeName is read-only
* nodeName of an element node is the same as the tag name
* nodeName of an attribute node is the attribute name
* nodeName of a text node is always #text
* nodeName of the document node is always #document

document.getElementById("id02").innerHTML = document.getElementById("id01").nodeName;

**The nodeValue Property**

The nodeValue property specifies the value of a node.

* nodeValue for element nodes is null
* nodeValue for text nodes is the text itself
* nodeValue for attribute nodes is the attribute value

document.getElementById("id02").innerHTML = document.getElementById("id01").nodeType;

**The most important nodeType properties are:**

Node Type Example

ELEMENT\_NODE 1 <h1 class="heading">W3Schools</h1>

ATTRIBUTE\_NODE 2 class = "heading" (deprecated)

TEXT\_NODE 3 W3Schools

COMMENT\_NODE 8 <!-- This is a comment -->

DOCUMENT\_NODE 9 The HTML document itself (the parent of <html>)

DOCUMENT\_TYPE\_NODE 10 <!Doctype html>

Type 2 is deprecated in the HTML DOM (but works). It is not deprecated in the XML DOM.

**JavaScript HTML DOM Elements (Nodes)**

**Creating New HTML Elements (Nodes)**

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

<div id="div1">  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
const para = document.createElement("p");  
const node = document.createTextNode("This is new.");  
para.appendChild(node);  
  
const element = document.getElementById("div1");  
element.appendChild(para);  
</script>

**Creating new HTML Elements - insertBefore()**

The appendChild() method in the previous example, appended the new element as the last child of the parent.

If you don't want that you can use the insertBefore() method:

<div id="div1">  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
const para = document.createElement("p");  
const node = document.createTextNode("This is new.");  
para.appendChild(node);  
  
const element = document.getElementById("div1");  
const child = document.getElementById("p1");  
element.insertBefore(para, child);  
</script>

**Removing Existing HTML Elements**

To remove an HTML element, use the remove() method:

<div>  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
const elmnt = document.getElementById("p1"); elmnt.remove();  
</script>

The remove() method does not work in older browsers, see the example below on how to use removeChild() instead.

**Removing a Child Node**

For browsers that does not support the remove() method, you have to find the parent node to remove an element:

<div id="div1">  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
const parent = document.getElementById("div1");  
const child = document.getElementById("p1");  
parent.removeChild(child);  
</script>

**Replacing HTML Elements**

To replace an element to the HTML DOM, use the replaceChild() method:

<div id="div1">  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
</div>  
<script>  
const para = document.createElement("p");  
const node = document.createTextNode("This is new.");  
para.appendChild(node);  
  
const parent = document.getElementById("div1");  
const child = document.getElementById("p1");  
parent.replaceChild(para, child);  
</script>

**JavaScript HTML DOM Collections**

**The HTMLCollection Object**

The getElementsByTagName() method returns an HTMLCollection object.

An HTMLCollection object is an array-like list (collection) of HTML elements.

The following code selects all <p> elements in a document:

const myCollection = document.getElementsByTagName("p");

The elements in the collection can be accessed by an index number.

To access the second <p> element you can write:

myCollection[1]

**HTML HTMLCollection Length**

The length property defines the number of elements in an HTMLCollection:

myCollection.length

The length property is useful when you want to loop through the elements in a collection:

const myCollection = document.getElementsByTagName("p");  
for (let i = 0; i < myCollection.length; i++) {  
  myCollection[i].style.color = "red";  
}

An HTMLCollection is NOT an array!

An HTMLCollection may look like an array, but it is not.

You can loop through the list and refer to the elements with a number (just like an array).

However, you cannot use array methods like valueOf(), pop(), push(), or join() on an HTMLCollection.

**JavaScript HTML DOM Node Lists**

**The HTML DOM NodeList Object**

A NodeList object is a list (collection) of nodes extracted from a document.

A NodeList object is almost the same as an HTMLCollection object.

Some (older) browsers return a NodeList object instead of an HTMLCollection for methods like getElementsByClassName().

All browsers return a NodeList object for the property childNodes.

Most browsers return a NodeList object for the method querySelectorAll().

The following code selects all <p> nodes in a document:

const myNodeList = document.querySelectorAll("p");

The elements in the NodeList can be accessed by an index number.

To access the second <p> node you can write:

myNodeList[1]

**HTML DOM Node List Length**

The length property defines the number of nodes in a node list:

myNodelist.length

const myNodelist = document.querySelectorAll("p");  
for (let i = 0; i < myNodelist.length; i++) {  
  myNodelist[i].style.color = "red";  
}

**JavaScript Window - The Browser Object Model**

The Window Object

The window object is supported by all browsers. It represents the browser's window.

All global JavaScript objects, functions, and variables automatically become members of the window object.

Global variables are properties of the window object.

Global functions are methods of the window object.

Even the document object (of the HTML DOM) is a property of the window object:

window.document.getElementById("header");

is the same as:

document.getElementById("header");

**Window Size**

Two properties can be used to determine the size of the browser window.

Both properties return the sizes in pixels:

window.innerHeight - the inner height of the browser window (in pixels)

window.innerWidth - the inner width of the browser window (in pixels)

The browser window (the browser viewport) is NOT including toolbars and scrollbars.

let w = window.innerWidth;  
let h = window.innerHeight;

Other Window Methods

window.open() - open a new window

window.close() - close the current window

window.moveTo() - move the current window

window.resizeTo() - resize the current window