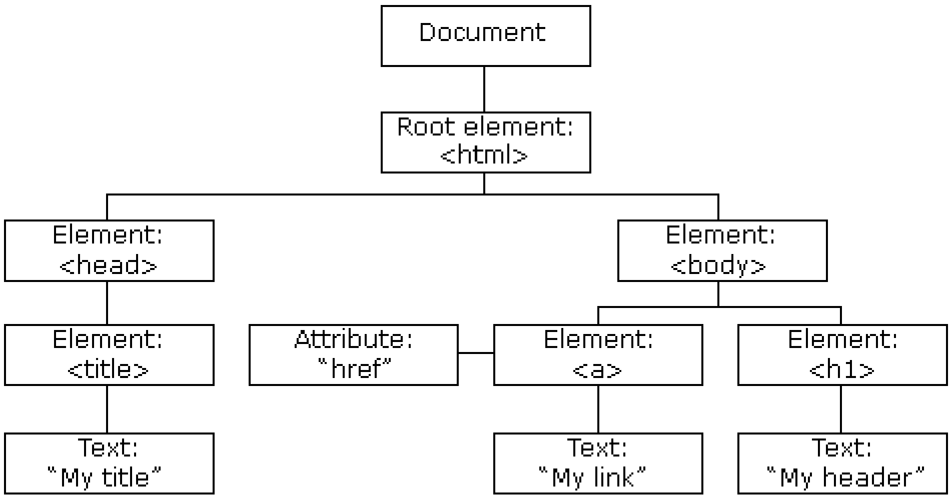
JavaScript HTML DOM

Def

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

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

The HTML DOM Tree of Objects



What is DOM

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

What is Html DOM

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

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

Uses

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

Programming Interface

Def

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

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

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

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

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

Example

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

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

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

DOM Document

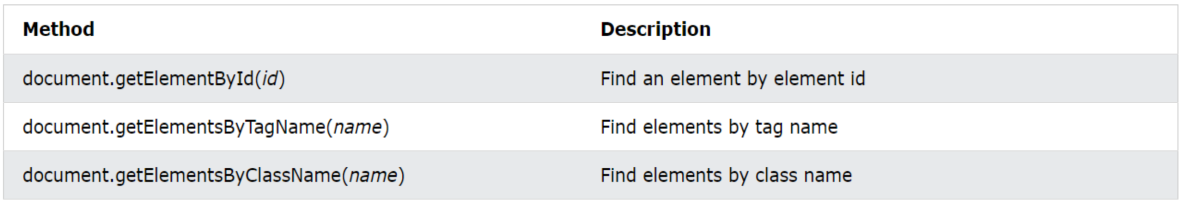
Def

The HTML DOM document object is the owner of all other objects in your web page.

The document object represents your web page.

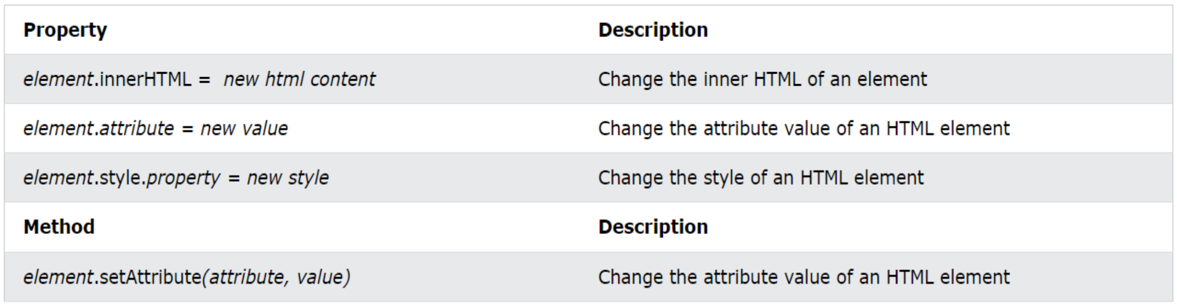
If you want to access any element in an HTML page, you always start with accessing the document object.

Finding HTML Elements



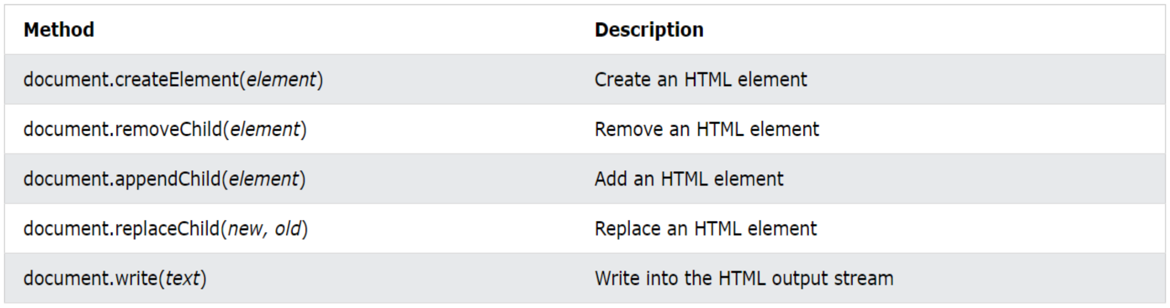
Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements



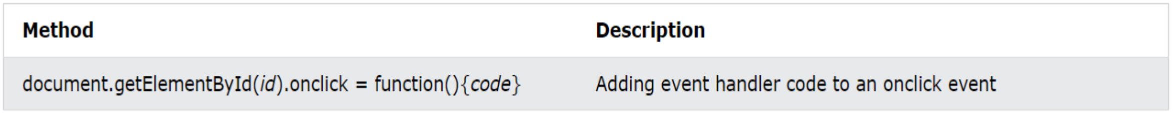
Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, 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**
document.anchors	Returns all <a> elements that have a name attribute
document.applets	Deprecated
document.baseURI	Returns the absolute base URI of the document
document.body	Returns the <body> element
document.cookie	Returns the document's cookie
document.doctype	Returns the document's doctype
document.documentElement	Returns the <html> element
document.documentMode	Returns the mode used by the browser
document.documentURI	Returns the URI of the document
document.domain	Returns the domain name of the document server
document.domConfig	Obsolete.
document.embeds	Returns all <embed> elements
document.forms	Returns all <form> elements
document.head	Returns the <head> element
document.images	Returns all <img> elements
document.implementation	Returns the DOM implementation
document.inputEncoding	Returns the document's encoding (character set)
document.lastModified	Returns the date and time the document was updated
document.links	Returns all <area> and <a> elements that have a href attribute
document.readyState	Returns the (loading) status of the document
document.referrer	Returns the URI of the referrer (the linking document)
document.scripts	Returns all <script> elements
document.strictErrorChecking	Returns if error checking is enforced
document.title	Returns the <title> element
document.URL	Returns the complete URL of the document

DOM Elements

Def

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 Elements

by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with id="intro":

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

by Tag Name

This example finds all <p> elements:

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

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

```
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");
```

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");
```

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".

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

by HTML Object Collections

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

```
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;
```

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

- document.anchors
- document.body
- document.documentElement
- document.embeds
- document.forms
- document.head
- document.images
- document.links
- document.scripts
- document.title

Changing HTML Elements

Content of Element

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*

This example changes the content of a <p> element:

```
<p id="p1">Hello World!</p>
<script>
document.getElementById("p1").innerHTML = "New text!";
</script>
```

Value of Attribute

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

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

This example changes the value of the src attribute of an <img> element:

```
<img id="myImage" src="smiley.gif">
<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>
```

DOM Forms Validations

Def

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>
```

Automatic 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>
```

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

Server-side validation is performed by a web server, after input has been sent to the server.

Client-side validation is performed by a web browser, before input is sent to a web server.

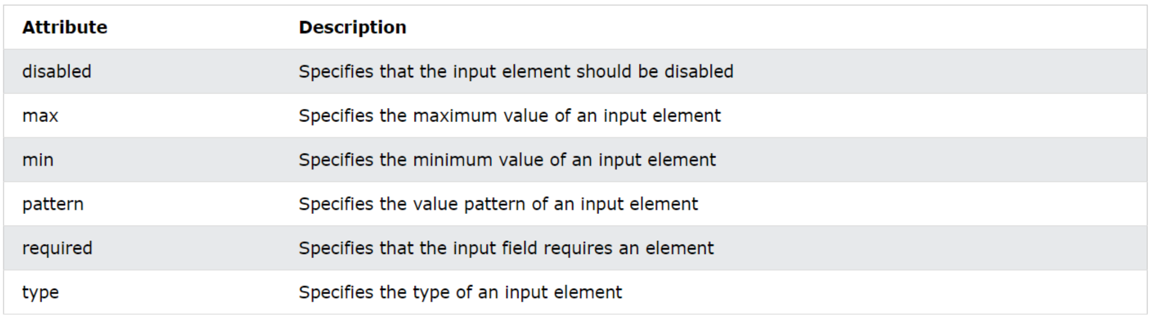
Constraints Validation

HTML5 introduced a new HTML validation concept called **constraint validation**.

HTML constraint validation is based on:

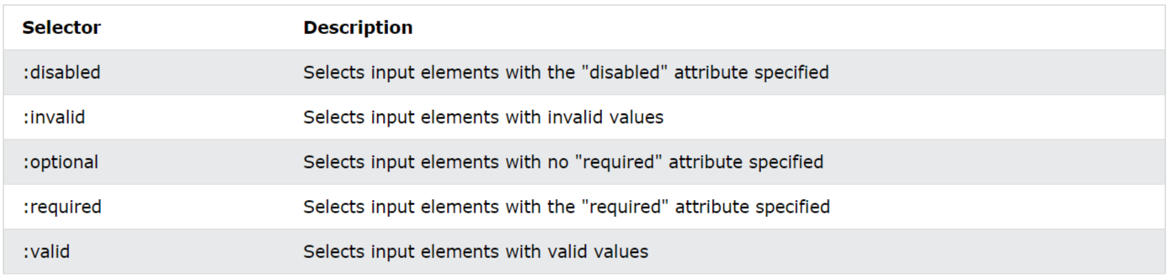
- Constraint validation **HTML Input Attributes**
- Constraint validation **CSS Pseudo Selectors**
- Constraint validation **DOM Properties and Methods**

By HTML Input Attributes



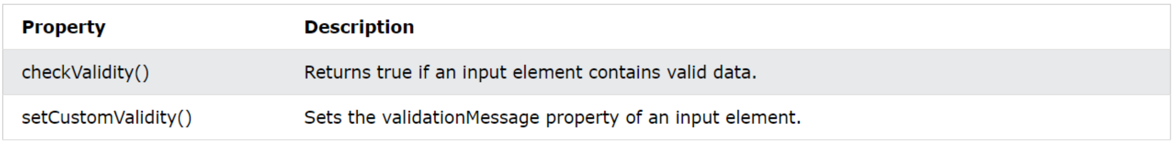
Attribute	Description
disabled	Specifies that the input element should be disabled
max	Specifies the maximum value of an input element
min	Specifies the minimum value of an input element
pattern	Specifies the value pattern of an input element
required	Specifies that the input field requires an element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

By CSS Pseudo Selectors



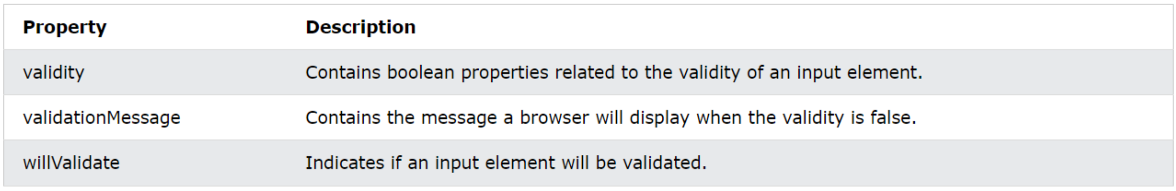
Selector	Description
:disabled	Selects input elements with the "disabled" attribute specified
:invalid	Selects input elements with invalid values
:optional	Selects input elements with no "required" attribute specified
:required	Selects input elements with the "required" attribute specified
:valid	Selects input elements with valid values

By DOM Methods



Property	Description
checkValidity()	Returns true if an input element contains valid data.
setCustomValidity()	Sets the validationMessage property of an input element.

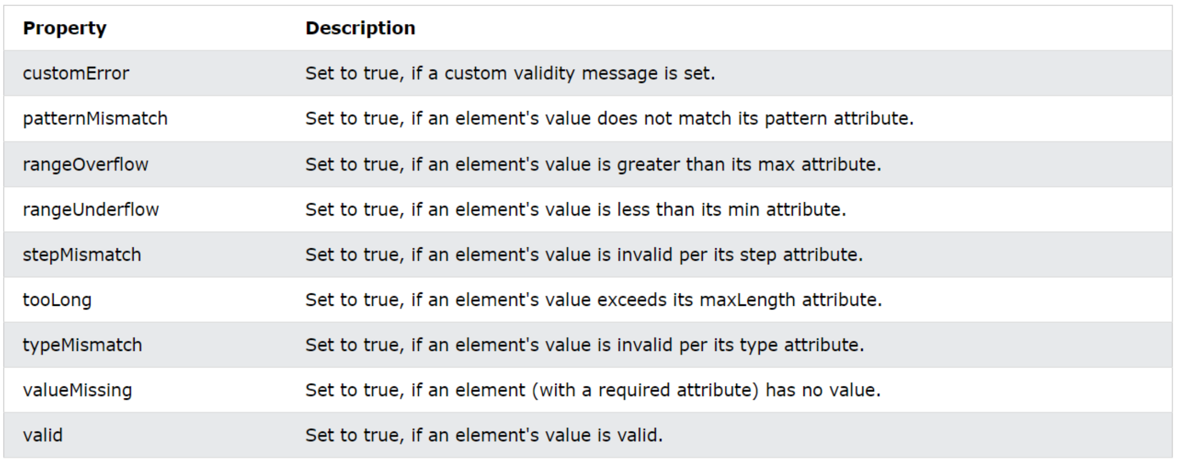
By DOM Properties



Property	Description
validity	Contains boolean properties related to the validity of an input element.
validationMessage	Contains the message a browser will display when the validity is false.
willValidate	Indicates if an input element will be validated.

Validity Properties

The **validity property** of an input element contains a number of properties related to the validity of data:



Property	Description
customError	Set to true, if a custom validity message is set.
patternMismatch	Set to true, if an element's value does not match its pattern attribute.
rangeOverflow	Set to true, if an element's value is greater than its max attribute.
rangeUnderflow	Set to true, if an element's value is less than its min attribute.
stepMismatch	Set to true, if an element's value is invalid per its step attribute.
tooLong	Set to true, if an element's value exceeds its maxLength attribute.
typeMismatch	Set to true, if an element's value is invalid per its type attribute.
valueMissing	Set to true, if an element (with a required attribute) has no value.
valid	Set to true, if an element's value is valid.

DOM Animation

Def

For Creating Animation in Javascript following steps need to be followed:

1. A Basic Web Page
2. Create an Animation Container
3. Style the Elements
4. Animation Code
5. Create the Full Animation Using JavaScript

A Basic Web Page

```
<h1>My First JavaScript Animation</h1>
<div id="animation">My animation will go here</div>
```

Create an Animation Container

All animations should be relative to a container element.

```
<div id ="container">
  <div id ="animate">My animation will go here</div>
</div>
```

Animation Code

JavaScript animations are done by programming gradual changes in an element's style.

The changes are called by a timer. When the timer interval is small, the animation looks continuous.

```
id = setInterval(frame, 5);
function frame() {
  if (/* test for finished */) {
    clearInterval(id);
  } else {
    /* code to change the element style */
  }}
```

Create the Full Animation Using JavaScript

```
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';
    } } }
```

DOM Events

Def

HTML DOM allows JavaScript to react to HTML events

Reacting to Events

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

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

In this example, the content of the <h1> element is changed when a user clicks on it:

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

In this example, a function is called from the event handler:

```
<h1 onclick="changeText(this)">Click on this text!</h1>
<script>
function changeText(id) {
  id.innerHTML = "Ooops!"; }
</script>
```

Assign Events Using the HTML DOM

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

Assign an onclick event to a button element:

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

HTML Event Attributes

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

Assign an onclick event to a button element:

```
<button onclick="displayDate()">Try it</button>
```

onload and onunload

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()">
```

oninput

The oninput event is often to some action while the user input data.

Below is an example of how to use the oninput to change the content of an input field.

```
<input type="text" id="fname" oninput="upperCase()">
```

onchange

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.

```
<input type="text" id="fname" onchange="upperCase()">
```

onmouseover and onmouseout

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

```
<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>
```

onmousedown, onmouseup and onclick

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.

DOM EventListener

Def

DOM EventListeners are used to trigger a function to handle the particular events.

The addEventListener() method

Add an event listener that fires when a user clicks a button:

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

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

The first parameter is the type of the event (like "click" or "mousedown" or any other HTML DOM Event.)

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

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

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

Add an Event Handler to an Element

Alert "Hello World!" when the user clicks on an element:

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

You can also refer to an external "named" function:

Alert "Hello World!" when the user clicks on an element:

```
element.addEventListener("click", myFunction);
```

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:

```
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
```

You can add events of different types to the same element:

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

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.

Add an event listener that fires when a user resizes the window:

```
window.addEventListener("resize", function(){
  document.getElementById("demo").innerHTML = sometext;
});
```

Passing Parameters

When passing parameter values, use an "anonymous function" that calls the specified function with the parameters:

```
element.addEventListener("click", function(){ myFunction(p1, p2); });
```

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);
```

The default value is false, which will use the bubbling propagation, when the value is set to true, the event uses the capturing propagation.

```
document.getElementById("myP").addEventListener("click", myFunction, true);
document.getElementById("myDiv").addEventListener("click", myFunction, true);
```

The removeEventListener() method

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

```
element.removeEventListener("mousemove", myFunction);
```

HTML DOM Event Object Reference

For a list of all HTML DOM events, look at our complete HTML DOM Event Object Reference.

DOM Navigation

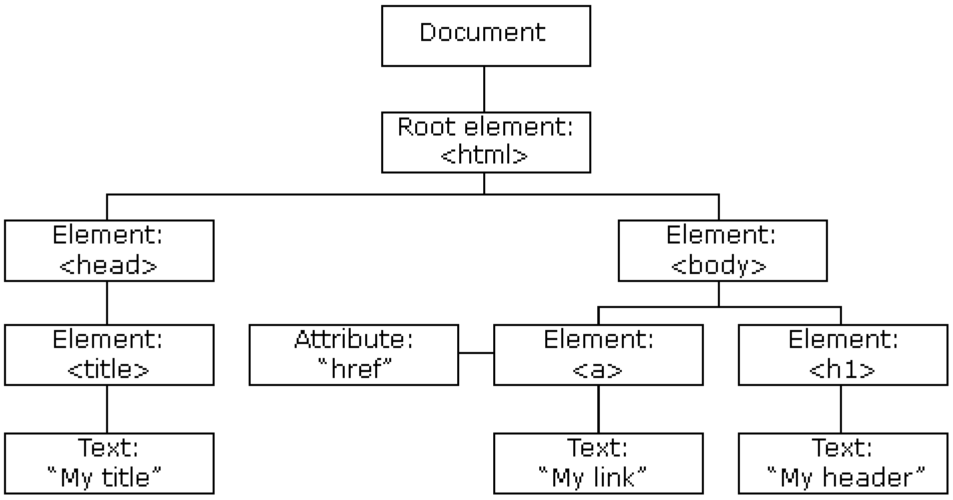
Def

With the HTML DOM, you can navigate the node tree using node relationships.

DOM Nodes

According to the W3C HTML DOM standard, everything in an HTML document is a node:

- 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



With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.

New nodes can be created, and all nodes can be modified or deleted.

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.

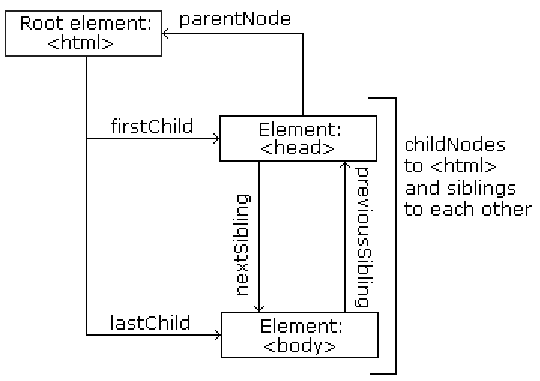
- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

```
<html>

 <head>
  <title>DOM Tutorial</title>
 </head>

 <body>
  <h1>DOM Lesson one</h1>
  <p>Hello world!</p>
 </body>

</html>
```



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>
- <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 sibling

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

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

and

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

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

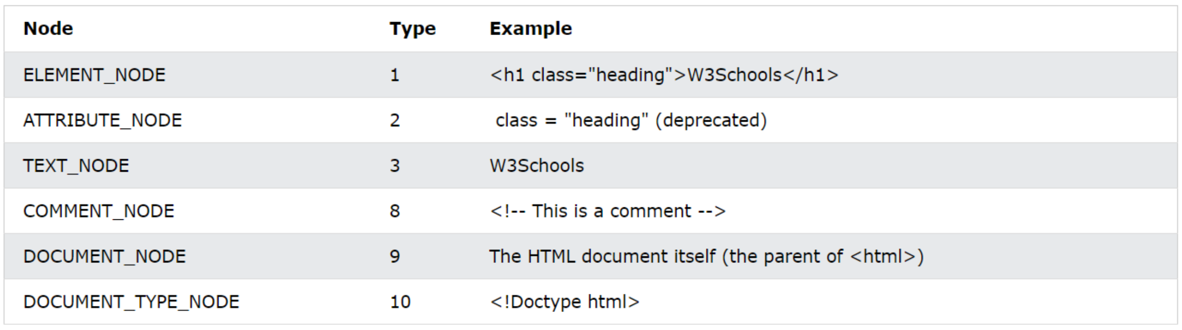
The nodeValue property specifies the value of a node.

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

The nodeType Property

The nodeType property is read only. It returns the type of a node.

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



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>

DOM Nodes

Def

Adding and Removing Nodes (HTML Elements)

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:

```
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);
```

Removing Existing HTML Elements

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

```
document.getElementById("p1"); elmnt.remove();
```

or

```
const parent = document.getElementById("div1");
const child = document.getElementById("p1");
parent.removeChild(child);
```

Replacing HTML Elements

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

```
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);
```

DOM Collections

Def

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]
```

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.

DOM Node Lists

Def

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]
```

Node List Length

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

```
myNodelist.length
```

The length property is useful when you want to loop through the nodes in a node list:

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

The Difference Between an HTMLCollection and a NodeList

A **NodeList** and an **HTMLcollection** is very much the same thing.

Both are array-like collections (lists) of nodes (elements) extracted from a document. The nodes can be accessed by index numbers. The index starts at 0.

Both have a **length** property that returns the number of elements in the list (collection).

An HTMLCollection is a collection of **document elements**.

A NodeList is a collection of **document nodes** (element nodes, attribute nodes, and text nodes).

HTMLCollection items can be accessed by their name, id, or index number.

NodeList items can only be accessed by their index number.

An HTMLCollection is always a **live** collection. Example: If you add a <li> element to a list in the DOM, the list in the HTMLCollection will also change.

A NodeList is most often a **static** collection. Example: If you add a <li> element to a list in the DOM, the list in NodeList will not change.

The getElementsByClassName() and getElementsByTagName() methods return a live HTMLCollection.

The querySelectorAll() method returns a static NodeList.

The childNodes property returns a live NodeList.

Not an Array!

A NodeList may look like an array, but it is not.

You can loop through a NodeList and refer to its nodes by index.

But you cannot use Array methods like push(), pop(), or join() on a NodeList.