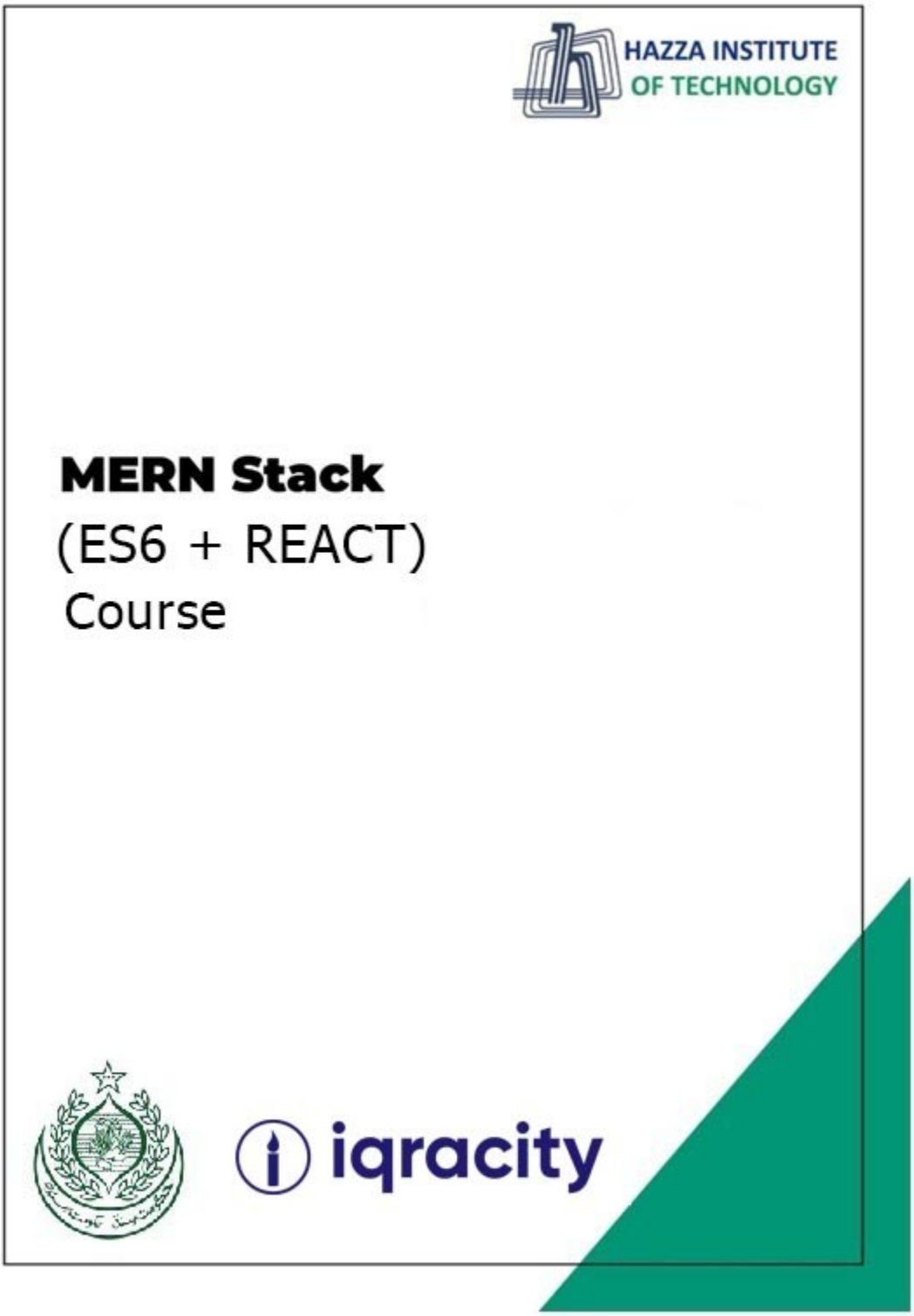
HAZZA INSTITUTE OF TECHNOLOGY

iqracity



Lab 12

Total Time:
3 hours

Pre-Lab Activities:

- No Pre-Lab Activity

Learning Outcomes:

- Perform the execution, debugging, testing, and profiling of web apps in modern IDEs.

Lab Tasks:

- DOM CSS
- DOM Animations
- DOM Events
- DOM Event Listener
- DOM Navigation
- DOM Nodes
- DOM Collections
- DOM Node Lists

Student Activities:

- Explore DOM CSS
- Explore DOM Animations
- Explore DOM Events
- Explore DOM Event Listener
- Explore DOM Navigation
- Explore DOM Nodes
- Explore DOM Collections
- Explore DOM Node Lists

Lab Solution

JavaScript HTML DOM - Changing CSS

The HTML DOM allows JavaScript to change the style of HTML elements.

Changing HTML Style

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

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

The following example changes the style of a <p> element:

Example

```
<html>
<body>

<p id="p2">Hello World!</p>

<script>
document.getElementById("p2").style.color = "blue";
</script>

</body>
</html>
```

Try it Yourself »

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

You will learn more about events in the next chapter of this tutorial.

This example changes the style of the HTML element with id="id1", when the user clicks a button:

Example

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

<h1 id="id1">My Heading 1</h1>

<button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
Click Me!</button>

</body>
</html>
```

Try it Yourself »

More Examples

Visibility How to make an element invisible. Do you want to show the element or not?

HTML DOM Style Object Reference

For all HTML DOM style properties, look at our complete HTML DOM Style Object Reference.

Test Yourself With Exercises

Exercise:

Change the text color of the <p> element to "red".

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

<script>
document.getElementById("demo")[ ] = "red";
</script>
```

JavaScript HTML DOM Animation

Learn to create HTML animations using JavaScript.

A Basic Web Page

To demonstrate how to create HTML animations with JavaScript, we will use a simple web page:

Example

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

<h1>My First JavaScript Animation</h1>

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

</body>
</html>
```

Create an Animation Container

All animations should be relative to a container element.

Example

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

Style the Elements

The container element should be created with style = "position: relative".

The animation element should be created with style = "position: absolute".

Example

```
#container {
  width: 400px;
```

```
  height: 400px;
  position: relative;
  background: yellow;
}
#animate {
  width: 50px;
  height: 50px;
  position: absolute;
  background: red;
}
```

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.

The basic code is:

Example

```
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

Example

```
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';
  }
 }
}
```

JavaScript HTML DOM Events

HTML DOM allows JavaScript to react to HTML events:

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

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

Example

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

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

</body>
</html>
```

Try it Yourself »

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

Example

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

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

<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

HTML Event Attributes

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

Example

Assign an onclick event to a button element:

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

Try it Yourself »

In the example above, a function named displayDate will be executed when the button is clicked.

Assign Events Using the HTML DOM

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

Example

Assign an onclick event to a button element:

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

Try it Yourself »

In the example above, a function named displayDate is assigned to an HTML element with the id="myBtn".

The function will be executed when the button is clicked.

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.

Example

```
<body onload="checkCookies()">
```

Try it Yourself »

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.

Example

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

Try it Yourself »

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:

Mouse Over Me

Try it Yourself »

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.

Click Me

Try it Yourself »

More Examples

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

onload
Display an alert box when the page has finished loading.

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

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

JavaScript HTML DOM EventListener

The addEventListener() method

Example

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

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

Try it Yourself »

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.)

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

Example

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

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

Try it Yourself »

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

Example

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

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

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

Try it Yourself »

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:

Example

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

Try it Yourself »

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

Example

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

Try it Yourself »

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.

Example

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

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

Try it Yourself »

Passing Parameters

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

Example

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

Try it Yourself »

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 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 element's click event.

In *capturing* the outer most element's event is handled first and then the inner: the 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.

Example

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

Try it Yourself »

The removeEventListener() method

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

Example

element.removeEventListener("mousemove", myFunction);

Try it Yourself »

HTML DOM Event Object Reference

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

Test Yourself With Exercises

Exercise:

Use the eventListener to assign an onclick event to the <button> element.

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

<script>
document.getElementById("demo").[          ]("[          ]", myFunction);
</script>
```

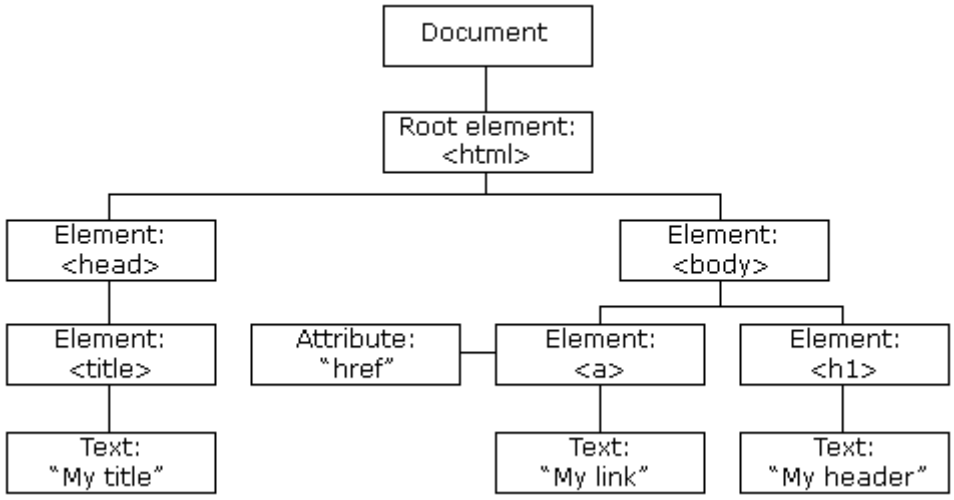
JavaScript HTML DOM Navigation

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.

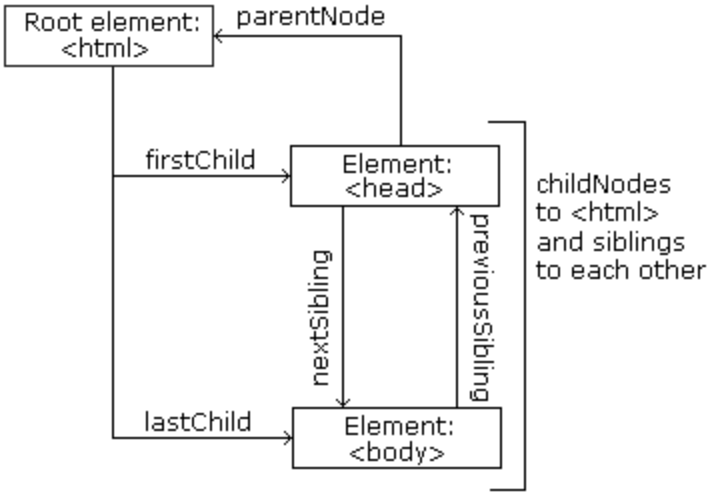
- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings (brothers or sisters) are nodes with the same parent

```
<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>

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

A common error in DOM processing is to expect an element node to contain text.

Example:

```
<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;
```

All the (3) following examples retrieves the text of an <h1> element and copies it into a <p> element:

Example

```
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").innerHTML;
</script>

</body>
</html>
```

Try it Yourself »

Example

```
<html>
<body>
```

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").firstChild.nodeValue;
</script>

</body>
</html>
```

Try it Yourself »

Example

```
<html>
<body>

<h1 id="id01">My First Page</h1>
<p id="id02">Hello!</p>

<script>
document.getElementById("id02").innerHTML =
document.getElementById("id01").childNodes[0].nodeValue;
</script>

</body>
</html>
```

Try it Yourself »

InnerHTML

In this tutorial we use the innerHTML property to retrieve the content of an HTML element.

However, learning the other methods above is useful for understanding the tree structure and the navigation of the DOM.

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

Example

```
<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>
```

Try it Yourself »

Example

```
<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>
```

Try it Yourself »

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

Example

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeName;
</script>
```

Try it Yourself »

Note: nodeName always contains the uppercase tag name of an HTML element.

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.

Example

```
<h1 id="id01">My First Page</h1>
<p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").nodeType;
</script>
```

Try it Yourself »

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)

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.

Example

```
<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>
```

Try it Yourself »

Example Explained

This code creates a new <p> element:

```
const para = document.createElement("p");
```

To add text to the <p> element, you must create a text node first. This code creates a text node:

```
const node = document.createTextNode("This is a new paragraph.");
```

Then you must append the text node to the <p> element:

```
para.appendChild(node);
```

Finally you must append the new element to an existing element.

This code finds an existing element:

```
const element = document.getElementById("div1");
```

This code appends the new element to the existing element:

```
element.appendChild(para);
```

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:

Example

```
<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>
```

Try it Yourself »

Removing Existing HTML Elements

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

Example

```
<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>
```

Try it Yourself »

Example Explained

The HTML document contains a element with two child nodes (two <p> elements):

```
<div>
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>
```

Find the element you want to remove:

```
const elmnt = document.getElementById("p1");
```

Then execute the remove() method on that element:

```
elmnt.remove();
```

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:

Example

```
<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>
```

Try it Yourself »

Example Explained

This HTML document contains a element with two child nodes (two <p> elements):

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>
```

Find the element with id="div1":

const parent = document.getElementById("div1");

Find the <p> element with id="p1":

const child = document.getElementById("p1");

Remove the child from the parent:

parent.removeChild(child);

Here is a common workaround: Find the child you want to remove, and use its parentNode property to find the parent:

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

Replacing HTML Elements

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

Example

```
<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>
```

Try it Yourself »

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:

Example

```
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]
```

Try it Yourself »

Note: The index starts at 0.

HTML HTMLCollection Length

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

Example

```
myCollection.length
```

Try it Yourself »

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

Example

Change the text color of all <p> elements:

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

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:

Example

```
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]
```

Try it Yourself »

Note: The index starts at 0.

HTML DOM Node List Length

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

Example

```
myNodelist.length
```

Try it Yourself »

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

Example

Change the color of all <p> elements in a node list:

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

Try it Yourself »

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.