JavaScript

HTML DOM: -

HTML DOM Animation: -

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.

An example below shows a small box will move diagonally.

```
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';
    }
```

HTML DOM Events: -

HTML DOM allows JavaScript to react to HTML events. A JavaScript can be executed when an event occurs, like when a 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.

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

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

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

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

HTML Event Attributes: -

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

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

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

Assign Events Using the HTML DOM: -

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

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.

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

The oninput Event: -

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

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

When you write in the input field, a function is triggered to transform the input to upper case.

The onchange Event: -

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

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

When you leave the input field, a function transforms the input to upper case.

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.

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.

The onfocus event: -

Example is to change the background-color of an input field when it gets focus.

HTML DOM EventListener: -

The addEventListener() method: -

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

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

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

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: -

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

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

below example shows external "named" function.

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

function myFunction() {

  alert ("Hello World!");

}
```

Add Many Event Handlers to the Same Element: -

element.addEventListener("click", myFunction);

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

Both will execute one after another.

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.

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

The removeEventListener() method: -

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

element.removeEventListener("mousemove", myFunction);

HTML DOM Navigation: -

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

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

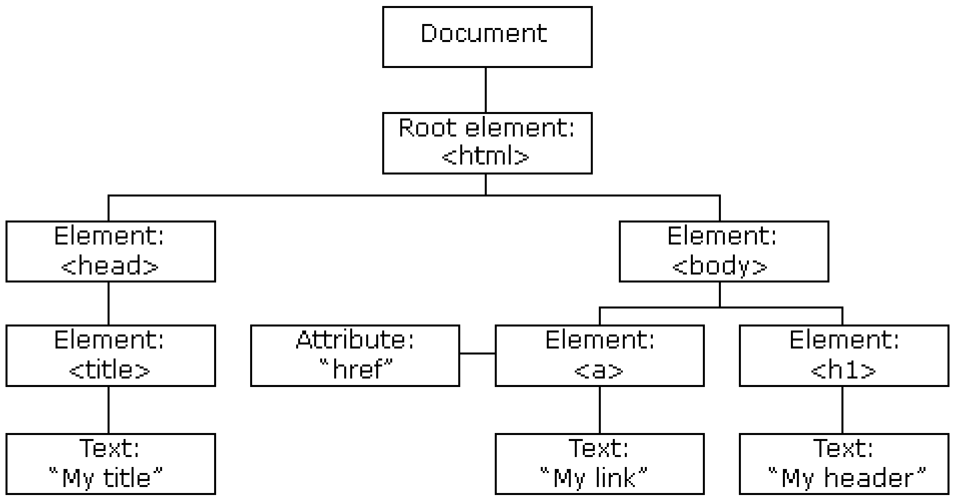
The entire document is a document node.

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 any 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>
```

<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 siblings

Navigating Between Nodes: -

There are node properties to navigate between nodes with JavaScript. These are parentNode, childNodes[nodenumber], firstChild, lastChild, nextSibling, previousSibling

Child Nodes and Node Values: -

```
<title id="demo">DOM Tutorial</title>
```

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

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

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

The whole text of body is copied to element having id – demo.

Other also does same way.

The nodeName Property: -

The nodeName property specifies the name of a node.

nodeName is read-only.

nodeName of an element node and attribute node is the same as the tag name and attribute node.

nodeName of a text node is always #text

nodeName of the document node is always #document

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

<p id="id02"></p>

<script>

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

</script> // H1
```

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 text and attribute nodes is attribute value.

The nodeType Property: -

The nodeType property is read only. It returns the type of a node. For element node it returns 1.

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 child = document.getElementById("p1");
element.insertBefore(para, child);
```

Removing Existing HTML Elements: -

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

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

Replacing HTML Elements: -

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

```
const parent = document.getElementById("div1");
const child = document.getElementById("p1");
parent.replaceChild(para, child);
```

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

The index starts at 0.

To access the second <p> element you can write myCollection[1]

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

myCollection.length

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.

HTML DOM Node List 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.

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

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

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

myNodelist.length

The Difference Between an HTMLCollection and a NodeList: -

An HTMLCollection is a collection of document elements.

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

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.