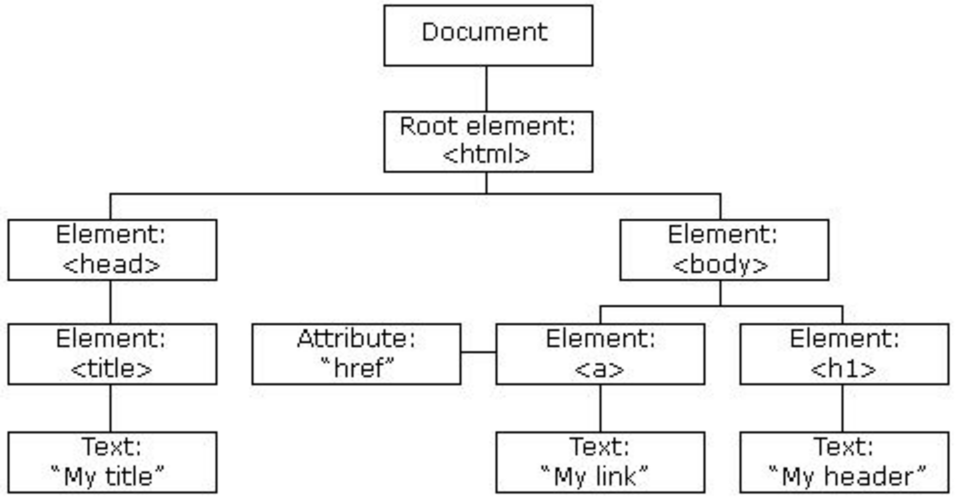
Document Object Model

COEN 161

What is the DOM?



- When the page loads, the browser creates a **Document Object Model** of the page
- The HTML DOM is built as a tree of objects

JS and the DOM

- With this model, JavaScript has all it needs to make a web page dynamic
- JavaScript can...
    - Change all the HTML *elements*, *attributes*, and CSS *styles* in the page
    - Remove existings elements and attributes
    - React to existing HTML events in the page
    - Create new HTML events in the page

What is the HTML DOM?

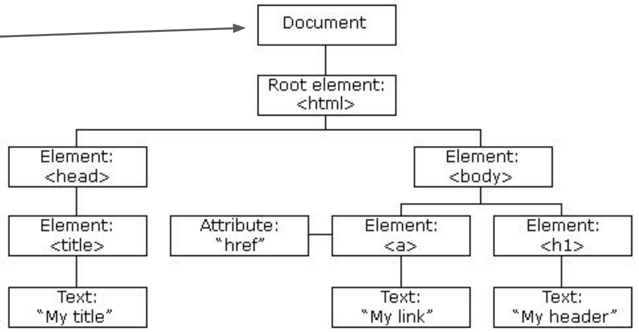
- The DOM is an interface for HTML
- It defines
    - The HTML elements as **objects**
    - The **properties** of all HTML elements
    - The **methods** to access HTML elements
    - The **events** for all HTML elements

HTML DOM

- The HTML DOM can be accessed by JavaScript
  - It can also be accessed by other languages, but JavaScript is natively built to work with DOM
- All elements are defined as JavaScript objects
  - That means that all HTML elements have **properties** and **methods**
- In the DOM, a property is a value that you can set, such as the content of an HTML element
- HTML DOM methods are actions that can be performed on the DOM, such as adding or removing elements

The document Object

- The base object for the DOM is `document`
- The document represents your entire HTML document
- To access any HTML elements, you start by accessing the document object



The getElementById Method

- Returns the HTML element that matches the given id

```
var myElement = document.getElementById("intro");
```

- If the element is not found, it returns null

The getElementsByTagName Method

- Returns all the elements with the given tag name

```
var x = document.getElementsByTagName("p");
```

- This method is available to all HTML elements

```
var x = document.getElementById("main");

// y contains all p elements in the element with id "main"

var y = x.getElementsByTagName("p");
```

The getElementsByClassName Method

- Returns all elements with the given class name

```
var x = document.getElementsByClassName("intro");
```

Finding Elements with CSS Selectors

- The `querySelectorAll` method supports CSS selectors for selecting DOM elements

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

- Note: the parameter uses the same syntax as CSS

The HTMLCollection Object

- The methods getElementsByTagName, getElementsByClassName, and querySelectorAll return an array-like list of HTML elements

```
var x = document.getElementsByTagName("p");

var y = x[1];
```

- This is NOT an Array, it is an object that behaves like an array
    - You can get the length of the list and loop through it, but you can’t use array methods like push and pop

HTML Collections

- Example

```
var myCollection = document.getElementsByTagName("p");

var i;

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

    myCollection[i].style.backgroundColor = "red";

}
```

HTML Object Collections

- The document has built in object collections
  - document.forms
  - document.images
  - document.links
  - document.scripts
- These properties are HTML collections and can be used as shorthand for selecting specific types of elements

Changing HTML Content

- The innerHTML property let's you set the content of an HTML element

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

- Example

```
<!DOCTYPE html>
<html>
     <body>
          <h1 id="id01">Old Heading</h1>
          <script>
               var element = document.getElementById("id01");
               element.innerHTML = "New Heading";
          </script>
`    </body>
</html>
```

Changing HTML Attributes

- HTML element attributes can be accessed as properties

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

- Example

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

Changing CSS Styles

- HTML elements also have a style attribute

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

- The style property has its own set of CSS properties you can set

```
<html>
     <body>
          <p id="p2">Hello World!</p>
          <script>
               document.getElementById("p2").style.color = "blue";
          </script>
          <p>The paragraph above was changed by a script.</p>
     </body>
</html>
```

Changing CSS Styles

- You can’t set the style property directly, you must access one its properties

```
document.getElementById("p2").style = "color:blue"; // does not work
```

- You can, however use the `setAttribute` method to set the style using CSS rule syntax

```
document.getElementById("p2").setAttribute("style", "color:blue"); // works!
```

- setAttribute can be used for any HTML attribute

```
document.getElementById("myImage").setAttribute("src", "landscape.jpg");
```

HTML DOM Events

- We’ve seen how to invoke JavaScript in scripts by calling functions from our scripts
- However, JavaScript can also be executed when an *event* occurs
- HTML events include
    - 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

HTML DOM Events

- HTML has attributes that let you define what happens when these events occur
- To set a click event we use the onclick attribute

  `onclick=`*`JavaScript`*

- Example

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

HTML DOM Events

- The value of the event attribute is just a string that contains JavaScript code
- We can even invoke functions from within the event attribute

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

HTML DOM Events

- We know that HTML attributes can be accessed from the DOM elements in JavaScript
- This includes HTML event attributes

```
<!DOCTYPE html>
<html>
    <body>
        <button id="myBtn">Try it</button>
        <script>
            document.getElementById("myBtn").onclick = displayDate;
            function displayDate() {
                alert(Date());
            }
        </script>
    </body>
</html>
```

Other HTML Events

- The *onload* and *onunload* events are triggered when a user enters and leaves the page, respectively
- The *onchange* event is used in combination with input elements

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

- The mouse events, such as onmouseover and onmouseout all trigger when some action is performed with the mouse
- The keyboard events, such as onkeydown, trigger when a key is pressed

Adding Event Listeners

- The addEventListener method lets you add an event handler to the specified event

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

- This lets you add more than, since it doesn’t override the event attribute

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

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

Event Bubbling and Capturing

- There are two ways events get triggered, bubbling and capturing
- These are two different orders in which events get handled, especially when multiple event handlers apply to the same element
- When events are bubbling, the innermost element gets handled first and the outermost element gets handled last
- When events are captured, the outermost element gets handled first and the innermost element gets handled last

Event Bubbling and Capturing

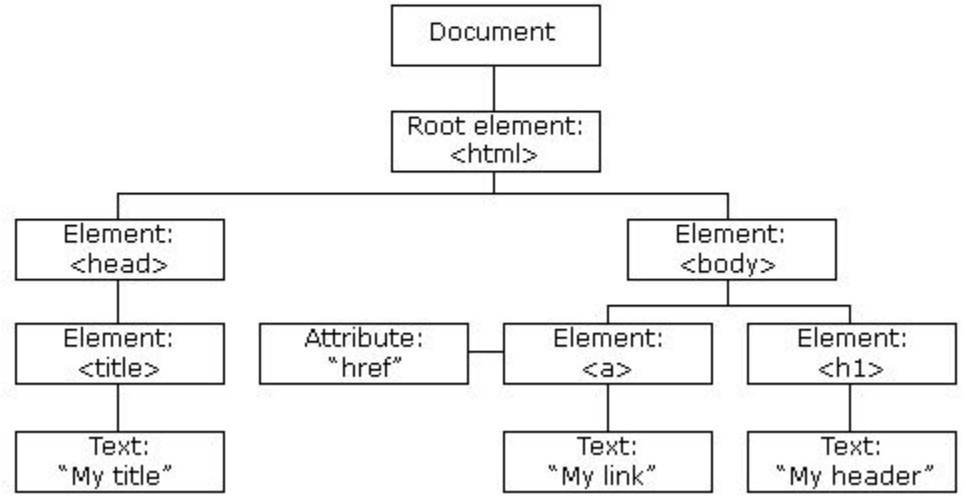
- The addEventListener methods takes a parameter that lets you define how to handle events

  ```
  addEventListener(event, function, useCapture); // boolean
  ```

- The default value is false, which means we bubble events, in to out
- If useCapture is true we capture events, out to in
- Example

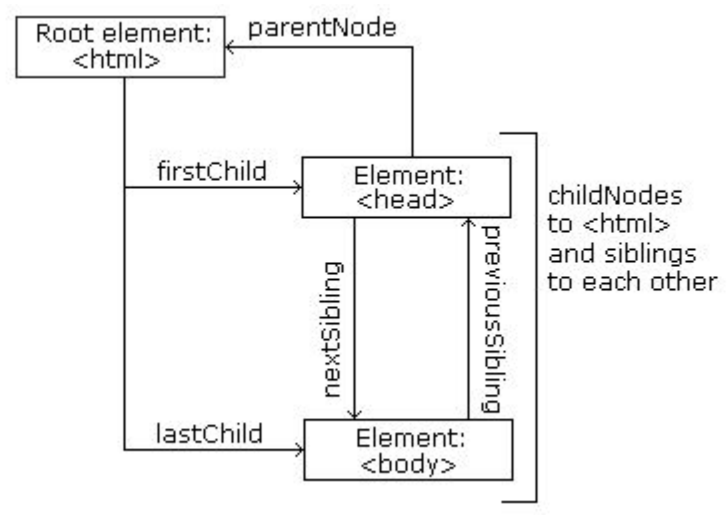
DOM Nodes

- Everything in the DOM is defined as a node



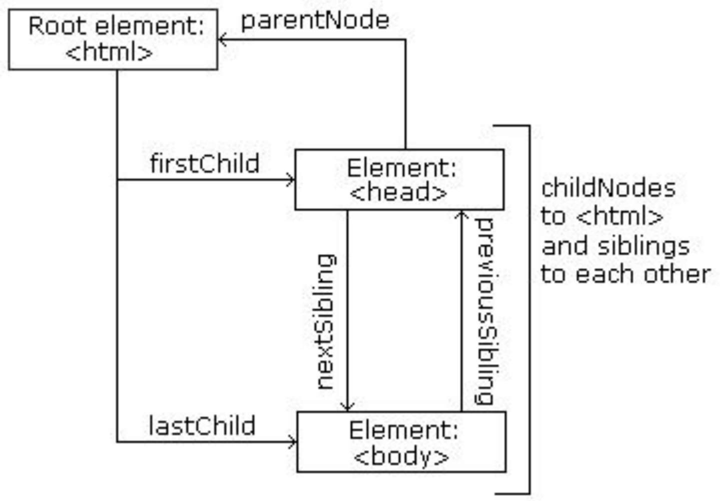
DOM Nodes

- JavaScript can access and modify these nodes, remove nodes, and add new nodes
- All nodes are a tree nodes, they have a parent, children, and siblings



Navigating DOM Nodes

- Each node has a set of properties to access other nodes
    - parentNode
    - childNodes[nodenumber]
    - firstChild
    - lastChild
    - nextSibling
    - previousSibling



Text Nodes

- Contrary to popular beliefs, nodes do not contain text, they actually contain a **text node**
- This text node has a value that can be accessed using the *nodeValue* property
- This is the same as accessing an elements *innerHTML* property

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

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

Other DOM Node Properties

- The *nodeName* property specifies the name of a node
  - nodeName cannot be modified
  - 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

- The *nodeValue* property specifies the value of a node
  - nodeValue for element nodes is undefined
  - nodeValue for text nodes is the text itself
  - nodeValue for attribute nodes is the attribute value

Other DOM Properties

- The *nodeType* property returns the type of that node

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>

Creating New Nodes

- Creating new elements involves two steps
  1. Creating the node
  2. Appending 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>
     var para = document.createElement("p");
     var node = document.createTextNode("This is new.");
     para.appendChild(node);
     var element = document.getElementById("div1");
     element.appendChild(para);
</script>
```

Creating New Nodes

- The *appendChild* method always adds a new node as the last child
- The *insertBefore* method lets you insert a node before another node, as its sibling

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

Removing Nodes

- To remove an existing element, you must know the parent of that node
- You must call *removeChild* from the parent node, passing the child node

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

<script>
     var parent = document.getElementById("div1");
     var child = document.getElementById("p1");
     parent.removeChild(child);
</script>
```

NodeLists

- Node lists are very similar to HTMLCollections
- Both are array-like, but NOT Arrays
- Both have a length property and let you access elements by index number
- However…
- HTML Collections can also be accessed by name or id, NodeLists can only be accessed by index
- NodeLists have additional nodes, like text or attribute

The Browser Object Model

- The BOM refers to the properties and methods that are used to interact with the browser
- The `window` object is used to access the BOM
- The `window` object is also considered the *global scope*
    - All global JavaScript objects, functions, and variables automatically become members of the window object
    - Even the document object is part of the window object

```
window.document.getElementById("header"); // is the same as...

document.getElementById("header");
```

Window Properties

- The window object has properties such that tell you information about the size of the browser window
  - **window.innerHeight** - the inner height of the browser window (in pixels)
  - **window.innerWidth** - the inner width of the browser window (in pixels)
- The window object also has properties that let you manipulate the browser
  - **window.open()** - open a new window
  - **window.close()** - close the current window
  - **window.moveTo()** -move the current window
  - **window.resizeTo()** -resize the current window

Window Location

- The *location* property can be used to get the current URL
  - **window.location.href** returns the href (URL) of the current page
  - **window.location.hostname** returns the domain name of the web host
  - **window.location.pathname** returns the path and filename of the current page
  - **window.location.protocol** returns the web protocol used (http: or https:)
  - **window.location.assign** loads a new document

```
window.location.assign("https://www.w3schools.com");
```

Window History

- The *history* object contains the browser’s history
  - However, to protect the privacy of the user, there are limitations as to what you can do with JavaScript
- The *back* method loads the previous URL in the history

```
history.back()
```

- The forward method loads the next URL in the history

```
history.forward()
```

Window Popups

- There are three kinds of window popups
- Alert - displays a message and goes away when you click “ok”
- Confirm - displays a message and the user can either click “ok” or “cancel”
  - Ok returns true
  - Cancel returns false
- Prompt - displays a message and allows the user to enter text input, also has “ok” and “cancel”
  - Ok returns the input value
  - Cancel returns null

Timing Events

- The window object also allows specific code to be run at specific time intervals
- Two events are used to control this, setTimeout and setInterval

The setTimeout Method

- It takes two parameters, a function to be executed, and the number of milliseconds before execution

```
window.setTimeout(function, milliseconds);
```

- Example

```
<button onclick="setTimeout(myFunction, 3000)">Try it</button>
<script>
    function myFunction() {
        alert('Hello');
    }
</script>
```

Stopping a Timeout

- setTimeout returns a variable that identifies that specific timeout event
- The clearTimeout method takes this variable as a parameter and cancels the event from executing, if it hasn't already
- Example

```
<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button>
<button onclick="clearTimeout(myVar)">Stop it</button>
```

The setInterval Method

- Also takes a function, but the second parameter is an interval at which to repeat the function execution

```
window.setInterval(function, milliseconds);
```

- Example

```
var myVar = setInterval(myTimer, 1000);
function myTimer() {
    var d = new Date();
    document.getElementById("demo")
        .innerHTML = d.toLocaleTimeString();
}
```

Stopping an Interval

- The method *setInterval* also returns a variable that identifies the interval event
- The method *clearInterval* takes that variable as a parameter and stops execution of that interval
- Example

```
<p id="demo"></p>
<button onclick="clearInterval(myVar)">Stop time</button>
<script>
     var myVar = setInterval(myTimer, 1000);
     function myTimer() {
         var d = new Date();
         document.getElementById("demo").innerHTML = d.toLocaleTimeString();
     }
</script>
```

Resources

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

https://www.w3schools.com/js/js_htmldom_document.asp

https://www.w3schools.com/js/js_htmldom_elements.asp

https://www.w3schools.com/js/js_htmldom_events.asp

https://www.w3schools.com/jsref/dom_obj_event.asp

https://www.w3schools.com/js/js_htmldom_eventlistener.asp

https://www.w3schools.com/js/js_window.asp

https://www.w3schools.com/js/js_timing.asp