DOM

**2.3.4** [**DOM ("Document Object Model") Manipulation**](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents)

**2.3.4.1 Background**

The DOM is the interface between JS and HTML/CSS (for making content interactive). When you load an HTML page, the browser turns every HTML tag into a JS object that we can select and manipulate, and all of these are stored inside of a "document" object. To view the HTML elements converted into JS objects (i.e., the DOM), type the following into the console: **console.dir(document);**

**NOTE**: When performing DOM manipulation, be sure that the JS files are loaded after all HTML is written (or else you will be trying to manipulate HTML that does not yet exist to the JS files, and it will not work.)

**2.3.4.2 Important DOM Selector Methods** (the following five methods have been added into the "document" object):

**· document.getElementById()**

o Takes in an ID name and returns the one element that matches that name (because an ID can only be associated with one element in a page). For example, if you have an HTML element with the ID of "highlight", you can return that element by stating:

var tag = document.getElementById('highlight');

* You can then use console.dir(tag) to see all of the properties contained in the "highlight" object.

**· document.getElementsByClassName()**

* Operates in the same way as getElementById(), but applies to all elements sharing the same class. If you want to return a list of all elements with the class "bolded", you would state:

var tags = document.getElementsByClassName('bolded');

* You can then simply state "tags" to see all of the properties for all "bolded" objects, and you can use console.log(tags[0]) to access the first index of the **Node List** (not technically an array, but is array-like) containing the "bolded" objects.

· The node list is "array-like" in the sense that you can access individual indices, but you **CANNOT** apply forEach() to it (because there is no forEach() function defined for these node lists; they are defined for arrays).

* **BUT NOTE**: You can use the "**[Array.from()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/from)**" method to convert node lists and HTML collections into arrays.
* **NOTE**: You can also access a specific index (e.g., the first index) more directly by simply stating:

var tags = document.getElementsByClassName('bolded')[0];

**· document.getElementsByTagName()**

* Operates in the same way as the two above, but it refers to a general HTML tag (such as <li>, <ul>, <h1>, <head>, <body>) rather than a specific class/ID. If you want to return a list of all elements with the <li> tag, state:

var tags = document.getElementsByTagName('li');

**· document.querySelector()**

* Operates in the same fashion as the methods above, but does so by using a **CSS**-Style Selector (i.e., "#" for ID, and "." for class). **BUT**, it differs significantly by only returning the **FIRST MATCH**. For example, to if you want to return an HTML element with the ID of "highlight", you would state:

var tag = document.querySelector('#highlight');

* If "highlight" were a class, you would use a period instead of an octothorp, but note that you will only return the first match.
* This works for any syntax that would be valid in CSS, e.g.:

var tags = document.querySelector('li a.special');

* query.Selector can also take in basic **HTML** tag names like <h1>:

var tag = document.querySelector('h2');

…but, again, only for the first match.

* If you want to select a **Specific Type** of element (e.g., a numerical input), you would use brackets and state:

var tag = document.querySelector('input[type="number"]');

**· document.querySelectorAll()**

* Operates in the same manner as query.Selector, except it returns a node list of **ALL** elements (as objects) that match a given CSS-style selector.

**2.3.4.3 Manipulating Style**

· A DOM object's **Style** property is one way to manipulate an HTML element's style, e.g.:

// select your object:

var tag = document.getElementById('highlight');

// manipulate your object:

tag.style.color = 'blue';

tag.style.border = '10px solid red';

tag.style.fontSize = '70px';

tag.style.background = 'yellow';

tag.style.marginTop = '200px';

* **NOTE**: This is not actually the best way to manipulate an object, as it is very repetitious (not "dry" code), but more significant is the "**Separation of Concerns**" (the principle that HTML, CSS, and JS should each be responsible for their own separate domain, and crossover between the three should be avoided – HTML should be pure structure, CSS should be pure presentation, and JS should be pure behavior).
* Rather than changing multiple style properties in JS, you should turn them on/off by using the CSS file. For example, you can define a CSS class, select an element in JS, and add the class to its **classList**:

// define a class in CSS:

.toggleWarning {

color: red;

border: 10px solid red;

}

// add the .toggleWarning class to a select object in JS:

var tag = document.getElementsByTagName('h1');

tag.classList.add('toggleWarning');

* If you so choose, you can later remove the class by stating:

tag.classList.remove('toggleWarning');

* A similar useful method is **toggle**, which operates by (1) adding the specified class if it is not present in the class list, and (2) removing the specified class if it is present in the class list:

tag.classList.toggle('toggleWarning');

* **NOTE**: A class list is technically **NOT AN ARRAY**. This is why you must use add() and remove() rather than push() and pop().

**2.3.4.4 Manipulating Text and Content**

The **textContent** property retrieves the text inside of an HTML element. "Text" is defined as anything between the HTML tags but not including any tags contained within (it extracts only **Plain Text**). The method can be used to alter the text by the following syntax:

tag.textContent = 'blah blah blah';

* **NOTE**: Because this method only works in plain text, no modifiers like <strong> or <em> will be preserved.
* The **innerHTML** property is used to manipulate both **HTML Text and Inner Element**s.
* **IMPORTANT SYNTAX NOTE**: When manipulating text and content (or style), it is not necessary to set a variable first. Instead, you can simply add the textContent or innerHTML property after the selector method, e.g.:

document.querySelector('h1').textContent = 'Heading 1';

* **Manipulating Attributes**

· The attributes of an HTML element (e.g., href, src, or anything else following the name="text" syntax, such as id, class, etc.) can be modified by using **getAttribute()** and **setAttribute()** to read and write attributes. For example, to return the URL for the following:

<a href="https://www.google.com/">Search</a>

…you would state:

var link = document.querySelector('a');

link.getAttribute('href');

…and to modify the link, you would state the attribute as the first argument in the method, and the new URL as the second argument:

link.setAttribute('href', 'https://www.yahoo.com/');

* **NOTE**: When manipulating **Images**, the "src" attribute may be ignored if the "srcset" attribute has also been set. In such cases, srcset must be modified.

**2.3.4.5 DOM Events**

· The [**MDN Event Reference**](https://developer.mozilla.org/en-US/docs/Web/Events)contains all of the different events that are recognized by the DOM application programming interface (**API**). Some common events to be most familiar with are "click", "change", "mouseover", and "mouseout".

· Events trigger the running of code when a certain action has been taken, as opposed to automatically executing when the page loads (e.g., clicking a button, hovering over a link, pressing a key, etc.). The event must be selected and then attached to a specific element by means of an **Event Listener** (e.g., "Listen for a click on this <button>." To add a listener, use the **addEventListener** method through this syntax:

element.addEventListener('type', functionToCall);

* For example, to have a message print to console when the first button in a document is clicked, you would state:

var button = document.querySelector('button'); // select element

button.addEventListener('click', function() { // add event listener

console.log('The button has been clicked.'); // run code

});

* It is not necessary to use an anonymous function. The same code could be executed as follows:

var button = document.querySelector('button');

button.addEventListener('click', printConsole);

function printConsole() {

console.log('The button has been clicked.');

}

However, unless there is a need to use a named function again somewhere else outside of the click listener, it is better to simply use the anonymous function.

* **NOTE**: You can have more than one listener on a given element. They will execute in the order that they were added.
* **THIS NOTE**: Inside of a listener, the "this" keyword refers to the item that was clicked on (or hovered upon, or which a keypress referred to, etc.). So whatever element is attached to the addEventListener is the event to which "this" refers. This is useful in situations where you want a large number of similar but individual (sibling) elements to run specific code only upon itself when activated. For example, if you have a <ul> with a dozen <li> tags within, and you want each individual <li> to change its color when clicked, you can use a for loop and "this" as follows:

var lis = document.querySelectorAll('li');

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

lis[i].addEventListener('click', function() {

this.style.color = 'pink';

});

}