

Learning outcome:

In this Lab, you will learn how to manipulate the DOM with vanilla JavaScript.

**1. First things first**

Before we dive into coding, let’s learn what the Document Object Model really is:

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects. That way, programming languages can connect to the page.

<https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction>

Basically, when a browser loads a page it creates an object model of that page and prints it on the screen. The object model is represented in a tree data structure, each node is an object with properties and methods, and the topmost node is the document object.

A programming language can be used to access and modify this object model, and this action is called DOM manipulation. And we will do that with JavaScript because JS is awesome.

**2. The actual tutorial**

For the tutorial, we are going to need two files, one index.html, and the other manipulation.js.

<!-- Index.html file --><html> <head> <title>DOM Manipulation</title> </head> <body> <div id="division"><h1 id="head"><em>DOM<em> manipulation</h1> <p class="text" id="middle">Tutorial</p><p>Sibling</p> <p class="text">Medium Tutorial</p> </div> <p class="text">Out of the div</p> <!-- Then we call the javascript file --> <script src="manipulation.js"></script> </body></html>

So there we have our HTML file, and as you can see we have a div with the id of division. Inside of that we have an h1 element, and in the same line, you will understand why later, we have two p elements and the div closing tag. Finally we have a p element with a class of text.

**2.1. Accessing the elements**

We can either access a single element or multiple elements.

**2.1.1. Accessing a single element**

For accessing a single element, we will look at two methods: getElementByID and querySelector.

// the method below selects the element with the id ofheadlet id = document.getElementById('head');

// the code below selects the first p element inside the first divlet q = document.querySelector('div p');

/\* Extra code \*/// this changes the color to redid.style.color = 'red';// give a font size of 30pxq.style.fontSize = '30px';

Now we have accessed two elements, the h1 element with the id of **head**and the first **p** element inside the div.

**getElementById** takes as argument an id, and **querySelector** takes as argument a CSS selector and returns the first element that matches the selector. As you can see I assigned the methods outcome into variables, and then I added some styling at the end.

**2.1.2. Accessing multiple elements**

When accessing multiple elements, a node list is returned. It is not an array but it works like one. So you can loop through it and use the length property to get the size of the node list. If you want to get a specific element you can either use the array notation or the item method. You will see them in the code.

For accessing multiple elements we are going to use three methods:

getElementsByClassName,

getElementsByTagName, and

querySelectorAll.

// gets every element with the class of textlet className = document.getElementsByClassName('text');

// prints the node listconsole.log(className);

/\* prints the third element from the node list using array notation \*/console.log(className[2]);

/\* prints the third element from the node list using the item function \*/console.log(className.item(2));

let tagName = document.getElementsByTagName('p');let selector = document.querySelectorAll('div p');

First, we use the **getElementsByClassName** that takes a class name as an argument. It returns a node list with every element that has **text**as a class. Then we print the node list on the console. We also print the third element from the list using the **array notation** and the **item method**.

Second, we select every **p**element using the getElementsByTagName method that takes a tag name as an argument and returns a node list of that element.

Finally, we use the **querySelectorAll** method, that takes as an argument a CSS selector. In this case, it takes **div p**so it will return a node list of **p** elements inside a div.

As a practice exercise, print all the elements from **tagName** and **selector** node list and find out their size.

**2.2. Traversing the DOM**

So far we have found a way of accessing specific elements. What if we want to access an element next to an element that we have already accessed, or access the parent node of a previously accessed element? The properties **firstChild, lastChild, parentNode, nextSibling,** and**previousSibling**can get this job done for us.

**firstChild**is used to get the first child element of a node. **lastChild**, as you guessed, it gets the last child element of a node. **parentNode**isused to access a parent node of an element. **nextSibling** gets for us the element next to the element already accessed, and **previousSibling**gets for us the element previous to the element already accessed.

// gets first child of the element with the id of divisionlet fChild = document.getElementById('division').firstChild;console.log(fChild);

// gets the last element from element with the id of divisionlet lChild = document.querySelector('#division').lastChild;

// gets the parent node of the element with the id divisionlet parent = document.querySElector('#division').parentNode;console.log(parent);

// selects the element with the id of middlelet middle = document.getElementById('middle');// prints ond the console the next sibling of middleconsole.log(middle.nextSibling);

The code above first gets the **firstChild** element of the element with the division id and then prints it on the console. Then it gets the **lastChild** element from the same element with the division id. Then it gets the **parentNode**of the element with the id of division and prints it on the console. Finally, it selects the element with the id of middle and prints its **nextSibling**node.

Most browsers treat white spaces between elements as text nodes, which makes these properties work differently in different browsers.

**2.3. Get and Updating element content**

**2.3.1. Setting and getting text Content**

We can get or set the text content of elements. To achieve this task we are going to use two properties: **nodeValue** and **textContent**.

**nodeValue** is used to set or get the text content of a text node. **textContent**is used to set or get the text of a containing element.

// get text with nodeValuelet nodeValue = document.getElementById('middle').firstChild.nodeValue;console.log(nodeValue);

// set text with nodeValuedocument.getElementById('middle').firstChild.nodeValue = "nodeValue text";

// get text with textContentlet textContent = document.querySelectorAll('.text')[1].textContent;console.log(textContent);

// set text with textContentdocument.querySelectorAll('.text')[1].textContent = 'new textContent set';

Did you notice the difference between **nodeValue** and **textContent**?

If you look carefully at the code above, you will see that for us to get or set the text with **nodeValue**, we first had to select the text node. First, we got the element with the **middle** id, then we got its **firstChild** which is the text node, then we got the **nodeValue** which returned the word Tutorial.

Now with **textContent**, there is no need to select the text node, we just got the element and then we got its **textContent**, either to set and get the text.

**2.3.2. Adding and Removing HTML content**

**You can add and remove HTML content in the DOM.**

For that, we are going to look at three methods and one property.

Let’s start with the **innerHTML** property because it is the easiest way of adding and removing HTML content. **innerHTML** can either be used to get or set HTML content. But be careful when using innerHTML to set HTML content, because it removes the HTML content that is inside the element and adds the new one.

document.getElementById('division').innerHTML =`<ul> <li>Angular</li> <li>Vue</li> <li>React</li></ul>`;

If you run the code, you will notice that everything else in the div with the id of division will disappear, and the list will be added.

We can use the methods: **createElement()**, c**reateTextNode()**, and **appendChild()** to solve this problem.

createElement is used to create a new HTML element. creatTextNode used to create a text node, and appendChild is used to append a new element into a parent element.

//first we create a new p element using the creatElementlet newElement = document.createElement('p');/\* then we create a new text node and append the text node to the element created\*/let text = document.createTextNode('Text Added!');newElement.appendChild(text);

/\* then we append the new element with the text node into the div with the id division.\*/document.getElementById('division').appendChild(newElement);

There is also a method called **removeChild** used to remove HTML elements.

// first we get the element we want to removelet toBeRemoved = document.getElementById('head');// then we get the parent node, using the parentNOde propertylet parent = toBeRemoved.parentNode;/\* then we use the removeChild method, with the element to be removed as a parameter.\*/parent.removeChild(toBeRemoved);

So first we get the element that we want to remove, and then we get its parent node. Then we called the method removeChild to remove the element.

**2.4. Attribute node**

Now we know how to handle elements, so let’s learn how to handle the attributes of these elements. There are some methods like **GetAttribute**, **setAttribute**, **hasAttribute**, **removeAttribute**, and some properties like **className** and **id**.

**getAttribute** as its name may suggest, it is used to get an attribute. Like the class name, the id name, the href of a link or any other HTML attribute.

**setAttribute** is used to set a new attribute to an element. It takes two arguments, first the attribute and second the name of the attribute.

**hasAttribute**used to check if an attribute exists, takes an attribute as an argument.

**removeAttribute**used to remove an attribute, it takes an attribute as an argument.

**Id** this property is used to set or get the id of an element.

**ClassName** is used to set or get the class of an element.

// selects the first divlet d = document.querySelector('div');//