**DOM and Events**

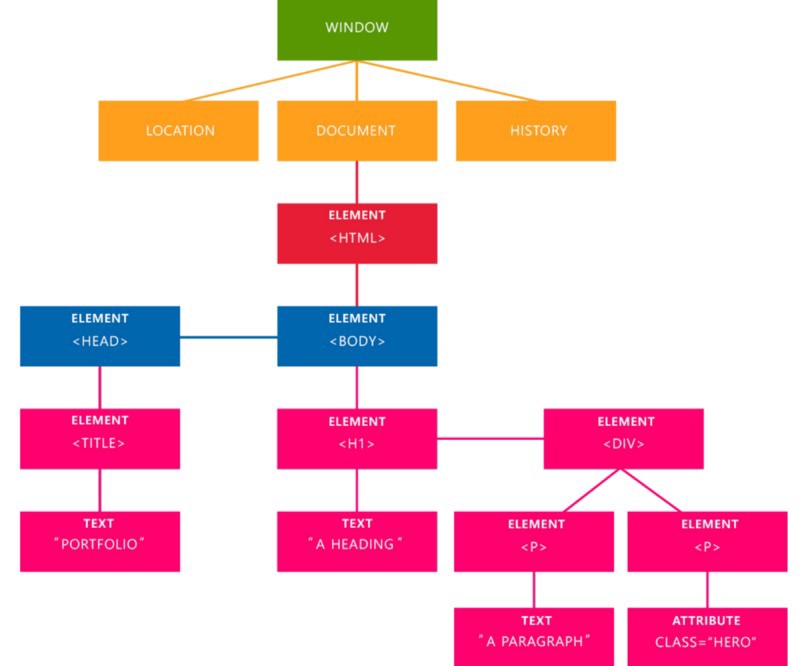
What is DOM?

Shoirt answer: DOM is a programming interface for website.

It present the HTML documents nodes for browser to read them and allow programming language like JavaScript to manipulate the HTML content (such as: change styles, animating, moving HTML elements).

The web browser use DOM to read your html files. After the browser read the HTML documents, it will create a tree to represent the HTML objects (can be refer as node) and defines how it can be access. This tree is called Document Object Model (DOM).

Example of the tree:



The tree image there represent the website HTML documents (DOM) that was created by browser we have 5 important element at this tree:

1. Window is the root of HTML documents. The window represent a window containing a DOM.
2. Document: It treats all the HTML documents.
3. Elements: All html tags that are inside your HTML or XML turn into a DOM element.
4. Text: All the tags’ value. (the element value)
5. Attributes: All the attributes from a specific HTML element. In the image, the attribute class=”hero” is an attribute from the <p> element.

Note: When a web page loads, the browser builds the page's DOM tree.

---------------------------

Why DOM Exists?

Document Object Model (DOM) aims to make it easier for Javascript to access HTML and XML elements.

DOM was created by World Wide Web Consortium (W3C) to allow any programming languages access to the Web document structure.

The first DOM specification (Level 0) was developed at the same time as JavaScript and early browsers. It is supported by Netscape 2 onward.

Accesing DOM  
Now we know about DOM, but now how to access the DOM to use it? To add new document, edit, remove? Every programming language can use the DOM built in method to access the DOM, such as:

document.getElementById(id) //finds element by id

document.getElementsByClassName(name) //finds elements by class name

document.getElementsByTagName(name) //finds elements by tag name

document.querySelector(name) //finds element by any query

getElementById

Usually the most common case to access web document is by selecting the ID, which used the method getElementById.

<p id="demo">

        THis is a paragraph

    </p>

    <script>

        let obj = document.getElementById("demo");

        obj.innerHTML= "This was edited by DOM";

    </script>

This code will changes the paragraph “This is a paragraph” into “This was edited by DOM.

innerHTML is an Element property that used to gets or sets the HTML or XML markup value.

We have used it to set the paragraph with demo ID.

Try to experience to use the innerHTML with getElementById

<p id="get-it">Paragraph</p>

    <p>Another paragraph</p>

    <p id="set-it"></p>

    <script>

    let getId= document.getElementById("get-it");

    console.log(getId.innerHTML);

    let setId = document.getElementById("set-it");

    setId.innerHTML = getId.innerHTML;

    </script>

There is the getId and setId, getId is used to get the html element value, the setId is used to set it up.

getElementsByClassName and getElementsByTagName

Before we learn the getClassName and TagName, Carefully notice between these documents.get**Element**ById with documents.get**Elements**ByClassName and documents.get**Elements**ByTagName!

These classes and tagName has Elements (which is plurar means more than one) while the id only Element which is single mean only one. getElementById will only pick one element that has the following ID.

The getElementsByClassName() method will return a collection of element that used the class name. For example, our html file contain 4 elements that used the class name “demo” the following code would return all those elements as an array:

<p class="demo">This is a paragraph</p>

    <p class="demo">This is a paragraph</p>

    <p class="demo">This is a paragraph</p>

    <p class="demo">This is a paragraph</p>

    <script>

        var arr = document.getElementsByClassName("demo");

        //accessing the second element

        arr[1].innerHTML = "Hi";

    </script>

This will change the second paragraph only.

Try to use arr.innerHTML = "Hi";

It will do nothing. Why? Because it returns an array of child elements, which is like this arr[0], arr[1], etc. Even if there is only one element that use “demo” class, it won’t output anything without define the array child number([0]). For example:

<p class="demo">This is a paragraph</p>

<script>

     var arr = document.getElementsByClassName("demo");

     arr.innerHTML = "Hi"; //this won’t work!!

</script>

<p class="demo">This is a paragraph</p>

 <script>

    var arr = document.getElementsByClassName("demo");

    arr[0].innerHTML = "Hi";//this will work, because it has the child array

 </script>

getElementsByTagName, will access document based on the tag name. Such as (“p”, “div”, “h1”, etc) Similar to getElementsByClassName, getElementsByTagName also return an array of child. Because there will be multiple element has the same tag name, for example three paragraph tags in html file.

<p>hi</p>

<p>hello</p>

<p>hi</p>

<script>

var arr = document.getElementsByTagName("p");

for(let x=0; x< arr.length; x++) {

    arr[x].innerHTML = "This was edited by DOM";

}

</script>

This will override all the <p> in the html file.

Now we understand about accesing elements with their tag name, class, and id. Now we proceed to Query Selector

<p id="demo">This is an ID paragraph</p>

<p class="demo">This is a Class paragraph</p>

let objId = document.querySelector("#demo");

let objClass = document.querySelector(".demo");

objId.innerHTML= "This ID was edited by DOM";

objClass.innerHTML= "This Class was edited by DOM";

The query Selector able to access any queries like ID, class, and even tag name. And in auto they will able to access the first element if there’s more than one element use the same class or tags.

<p class="demo">This is a Class paragraph</p>

    <p class="demo">This is a Class paragraph</p>

    <script>

        let objClass = document.querySelector(".demo");

        objClass.innerHTML= "This Class was edited by DOM";

    </script>

At above example, only the first one change while the second stay. That’s because its querySelector. It will only access the first element. Now we proceed to another method to access all element which called querySelectorAll.

querySelectorAlll will access all of the element it select, like more than one element which of course will return array like the ByClassName and ByTagName.

    <h1 class="demo">Heading</h1>

    <p class="demo">Paragraph</p>

    <script>

        var obj = document.querySelectorAll(".demo");

        obj[0].innerHTML = "This is Heading has been edit by DOM!";

        obj[1].innerHTML = "This is Paragraph has been edit by DOM!";

    </script>

**This querySelectorAll will select any elements that has class “demo” the [0] is the first index which is the h1 while the second is the paragraph.**

**Others property to mention:**

* **element.childNodes returns an array of an element's child nodes.**
* **element.firstChild returns the first child node of an element.**
* **element.lastChild returns the last child node of an element.**
* **element.hasChildNodes returns true if an element has any child nodes, otherwise false.**
* **element.nextSibling returns the next node at the same tree level.**
* **element.previousSibling returns the previous node at the same tree level.**
* **element.parentNode returns the parent node of an element.**

**Hint: Try to use setTimeout for better learning**

//calling the function with setTimeout to make sure the HTML is loaded

setTimeout(functionName, 500);

Changing DOM Nodes

Previously, we have learn how to change HTML elements by accesing the DOM with inner HTML. But now, there’s more properties and method we can use to change HTML Elements with DOM! Such as:

* elementName.innerHTML = renew html element content
* elementName.attribute = change the attribute value of HTML element
* elementName.style.propertyName = change HTML element styles (the element CSS)
* elementName.setAttribute(attribute, value) = change HTML element attribute value

Change HTML DOM Attribute

With this property you can change the html element attribute for example:

* <img> src attribute
* <a> href attribute
* <p> class attribute

And many more. For example, you can change the href attribute link for <a> tag:

    <a href="http://www.example.com">Some link</a>

    <script>

        var el = document.getElementsByTagName("a");

        el[0].href = "http://www.sololearn.com";

    </script>

There’s another way called elementName.setAttribute(attribute, value), to change HTML element attribute through DOM. Give it a try:

    <a href="http://www.example.com">Some link</a>

    <script>

        var el = document.getElementsByTagName("a");

        el[0].setAttribute("href", "http://www.sololearn.com");

    </script>

Change HTML DOM Styles

With elementName.styles.propertyName, we can change the HTML through DOM. For example a red color div, will changed into have blue color, with yellow bg and padding.

Give it a try:

<div id="demo" style="color:red;">some text</div>

var x = document.getElementById("demo");

    x.style.color = "6600FF";

    x.style.padding = "25px";

    x.style.backgroundColor = "yellow"

Adding Elements

There are following methods to create HTML Documents by DOM:

* document.createElement(“name”) : will create a HTML Element
* document.createTextNode(“text”): create a new text node
* document.appendChild(“element”): Add an HTML element child node
* document.write: Write into HTML output

You may confuse between the createElement and the appendChild method, like are they both adding and create new element? Answer is No,

Both doc.createElement() and createTextNode() is only to create the HTML tags and value . BUT it will not be add to the DOCUMENT until you use the appendChild() method to append two of them to add the nodes.

Here’s for example:

    <div id ="demo">some content</div>

    <script>

    //creating a new paragraph

    var p = document.createElement("p");

    var node = document.createTextNode("Some new text");

    //adding the text to the paragraph

    p.appendChild(node);

    var div = document.getElementById("demo");

    //adding the paragraph to the div

    div.appendChild(p);

    </script>

The createElement() used to create the HTML tag and then createTextNode is to create the HTML value. And then use the appendChild() to append between the tag and the value.

The *p.appendChild(node);* only used to join the text to the <p> tag to become one, but not creating the document yet. Until you append with a parent child like <div id=”demo”>.

Removing Elements

The following methods that used to remove HTML documents:

* document.removeChild(element): To remove a HTML element
* document.replaceChild(new child, old child): To replace a HTML element

To remove an element you must select the parent and the child node first, then use the removeChild() to the parent to remove the child. For example:

    <div id="demo">

    <p id="p1">This is a paragraph.</p>

    <p id="p2">This is another paragraph.</p>

    </div>

    <script>

    var parent = document.getElementById("demo");

    var child1 = document.getElementById("p1");

    parent.removeChild(child1);

    </script>

An alternative way of achieving the same result would be the use of the parentNode property to get the parent of the element we want to remove:

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

    child.parentNode.removeChild(child);

You can also replace a HTML content with replaceChild() like *parentDoc.replaceChild(new-child, old-child);* its not about remove or add but replace, for example:

<script>

var p = document.createElement("p");

var node = document.createTextNode("This paragraph has been replaced by DOM");

p.appendChild(node);

var demo = document.getElementById("demo");

var p1 = document.getElementById("p1");

demo.replaceChild(p, p1);

</script>

The formula for replace child is: that you append the new child, and then append the parent and its old child. Then use the replaceChild() with its parents while the new child as first parameter and old child as second parameter.

[How the Document Object Model Works in JavaScript – DOM Tutorial for Beginners (freecodecamp.org)](https://www.freecodecamp.org/news/javascript-dom/)

[What’s the Document Object Model, and why you should know how to use it. (freecodecamp.org)](https://www.freecodecamp.org/news/whats-the-document-object-model-and-why-you-should-know-how-to-use-it-1a2d0bc5429d/)