Agenda

1. What is DOM?
2. What DOM Covers?
3. DOM Basics
4. HTML DOM Elements
5. Accessing DOM Elements

What is DOM?

The Document Object Model (DOM) is a programming interface that lets us add, edit, and delete components from a document.

The DOM views an HTML document as a tree of nodes. A node represents an HTML element.

Imagine this: you have the TV on. You don't like the show that's being streamed, and you want to change it. You also want to increase its volume.

To do that, there has to be a way for you to interact with your television. And what do you use to do that? **A remote**.

The remote serves as the **bridge** which allows you interact with your television.

You make the TV **active** and **dynamic** via the remote. And in the same way, JavaScript makes the HTML page active and dynamic via the **DOM**.

Just like how the television can't do much for itself, JavaScript doesn't do much more than allow you to perform some calculations or work with bas strings.

So to make an HTML document more interactive and dynamic, the script needs to be able to access the contents of the document and it also needs know when the user is interacting with it.

It does this by communicating with the browser using the properties, methods, and events in the interface called the Document Object Model, or DOM.

Why DOM Is Required?

The DOM is what allows you to change the content and design on structure via JavaScript. This is because it gives you the ability to add new or modi existing elements in or on it in your scripts, which can speed up specific tasks (like dynamically adding an iframe) and decrease load times for page using it. The DOM is how both browsers and JavaScript read HTML files to render them onto the screen. The Document Object Model is an integral pa of any website. Now we'll look at what all things DOM Covers.

The DOM represents an HTML document as a tree of nodes. The DOM provides functions that allow you to add, remove, and modify parts of tl document effectively.

What DOM Covers?

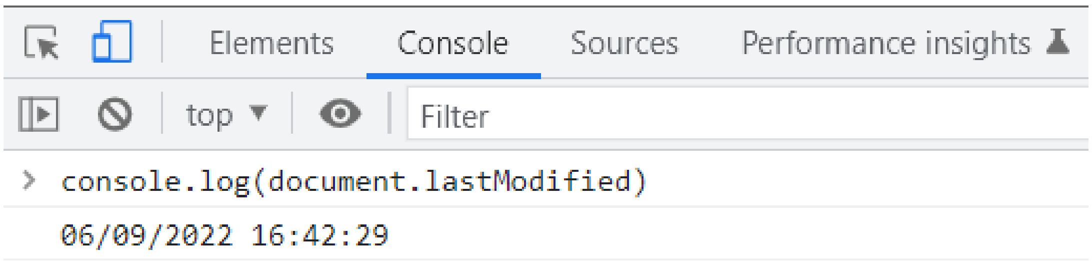
The DOM API allows full control, both querying and altering the DOM, and you have practically full support in the API to leverage all features provided l the HTML markup.

The best way to understand what the DOM API covers is to look at examples, but before doing this, let's have a short overview of what this API offers developers:

- First of all, it **allows access to a certain part of the document (the HTML markup)**. You can access a certain markup element using its identifier, or a collection of elements by their name, class type (according to the values of the element's class attribute), their tag name (for example all `<h1>` elements), and several other aspects.

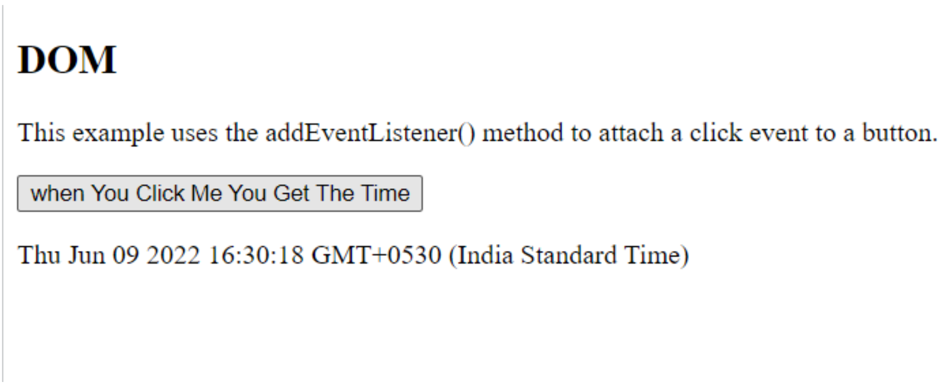
```
const element = document.getElementsByTagName("h1");
```

- You can **query the metadata of a document** including its URL, character set, last modification date, etc.



```
> console.log(document.lastModified)
  06/09/2022 16:42:29
```

Almost every interaction between a user and a web page is built on DOM. When you click a component of a page, a node of the DOM hierarchy receiv the event and responds by executing some code.


DOM

This example uses the addEventListener() method to attach a click event to a button.

Thu Jun 09 2022 16:30:18 GMT+0530 (India Standard Time)

- Web pages that keep the UI responsive while running queries against the server in the background use the DOM to update the page when results arrive, too. Example:

> If the viewport is less than, or equal to, 700 pixels wide, change the background color to yellow. If it is greater than 700, change it to pink

```
function myFunction(x) {
  if (x.matches) { // If media query matches
    document.body.style.backgroundColor = "yellow";
  } else {
    document.body.style.backgroundColor = "pink";
  }
}

var x = window.matchMedia("(max-width: 700px)")
myFunction(x) // Call listener function at run time
x.addListener(myFunction) // Attach listener function on state changes
```

- Many pages generate additional UI components at the client-side, right after the page is loaded, such as thumbnails, quick links, table of contents, or other elements that help with page navigation. The scripts behind these activities also utilize the DOM.

> • `DOMContentLoaded` – the browser fully loaded HTML, and the DOM tree is built, but external resources like pictures `<img>` and stylesheets may not yet have loaded.

```
function ready() {
    alert('DOM is ready');

    // image is not yet loaded (unless it was cached), so the size is 0x0
    alert(`Image size: ${img.offsetWidth}x${img.offsetHeight}`);
  }

document.addEventListener("DOMContentLoaded", ready);
```

- Moving charts, animated figures, interactive footers and banners all leverage the DOM.

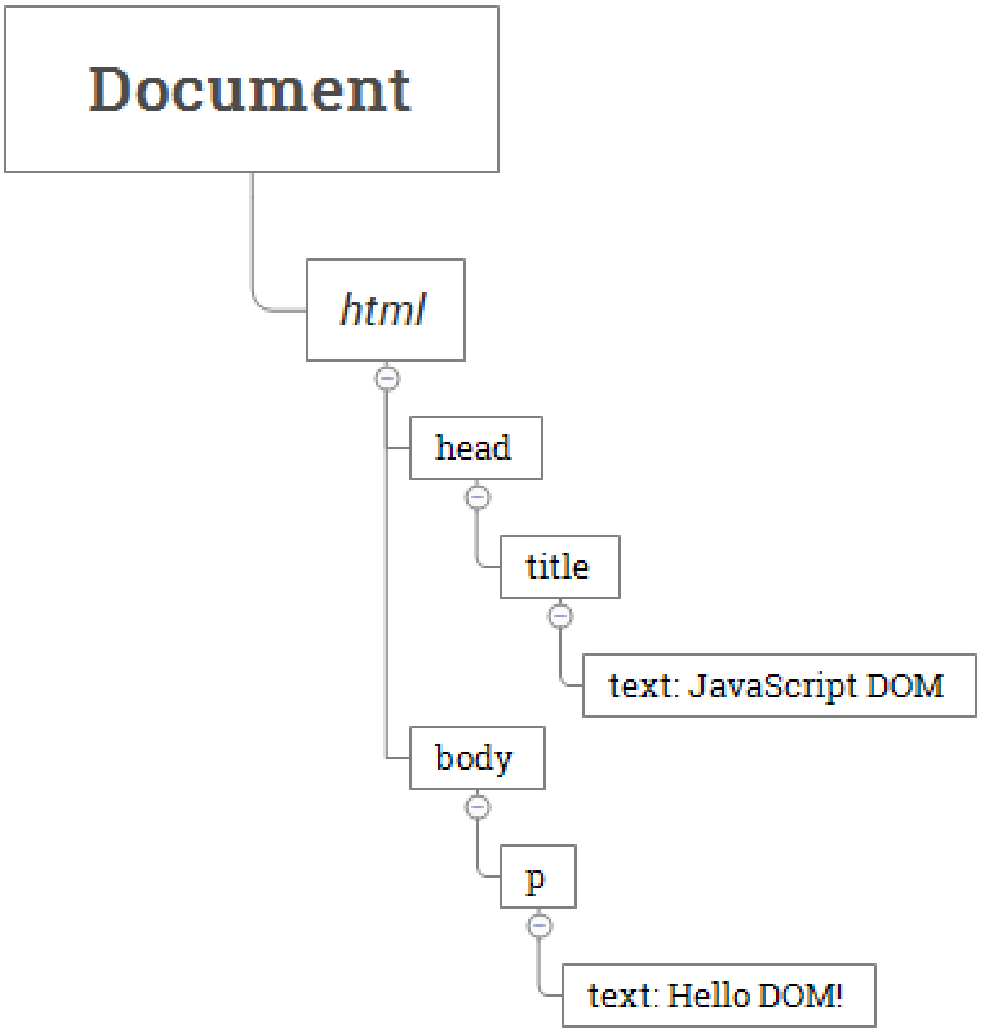
DOM Basics

A document as a hierarchy of nodes

The DOM represents an HTML document as a hierarchy of nodes. Consider the following HTML document:

```
<html>
    <head>
        <title>JavaScript DOM</title>
    </head>
    <body>
        <p>Hello DOM!</p>
    </body>
</html>
```

The following tree represents the above HTML document:



In this DOM tree, the document is the root node. The root node has one child node which is the element. The element is called the *document element*.

Each document can have only one document element. In an HTML document, the document element is the element. Each markup can be represente
by a node in the tree.

Node and Element

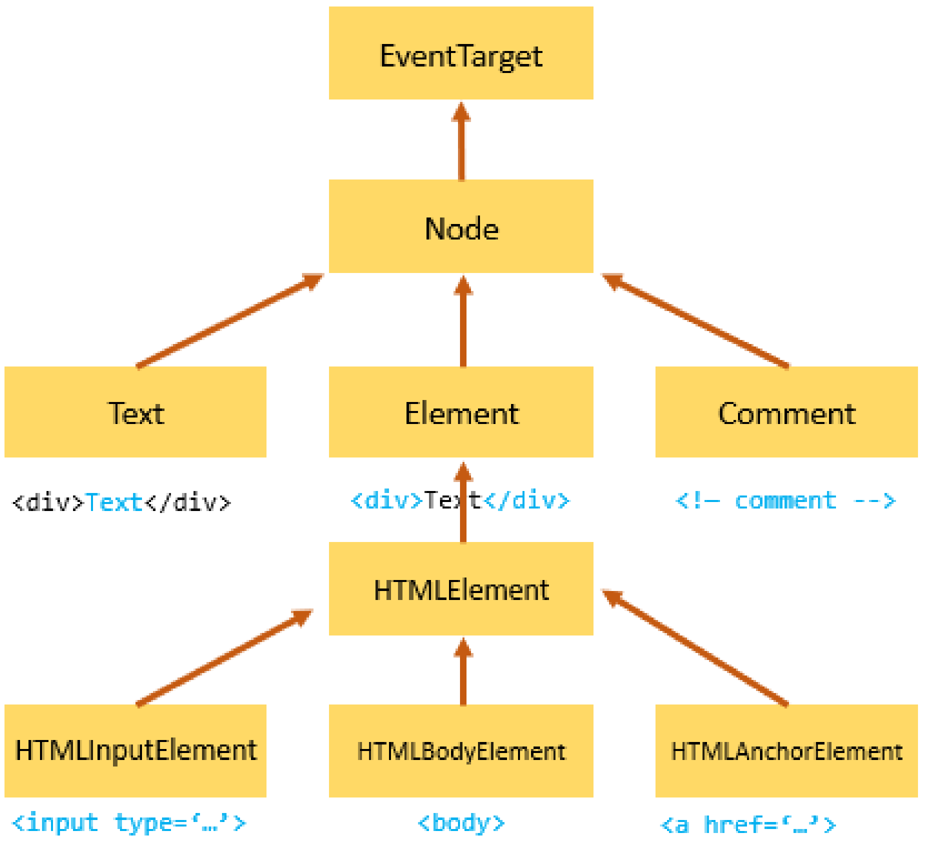
Sometimes it's easy to confuse between the `Node` and the `Element` .

A node is a generic name of any object in the DOM tree. It can be any built-in DOM element such as the document. Or it can be any HTML tag specifie
in the HTML document like `<div>` or `<p>` .

An element is a node with a specific node type `Node.ELEMENT_NODE` .

In other words, the node is the generic type of element. The element is a specific type of the node with the node type `Node.ELEMENT_NODE` .

The following picture illustrates the relationship between the `Node` and `Element` types:



> Note that the getElementById()and querySelector()returns an object with the Element type while getElementsByTagName()or querySelectorAll()returns NodeList which is a collection of nodes.

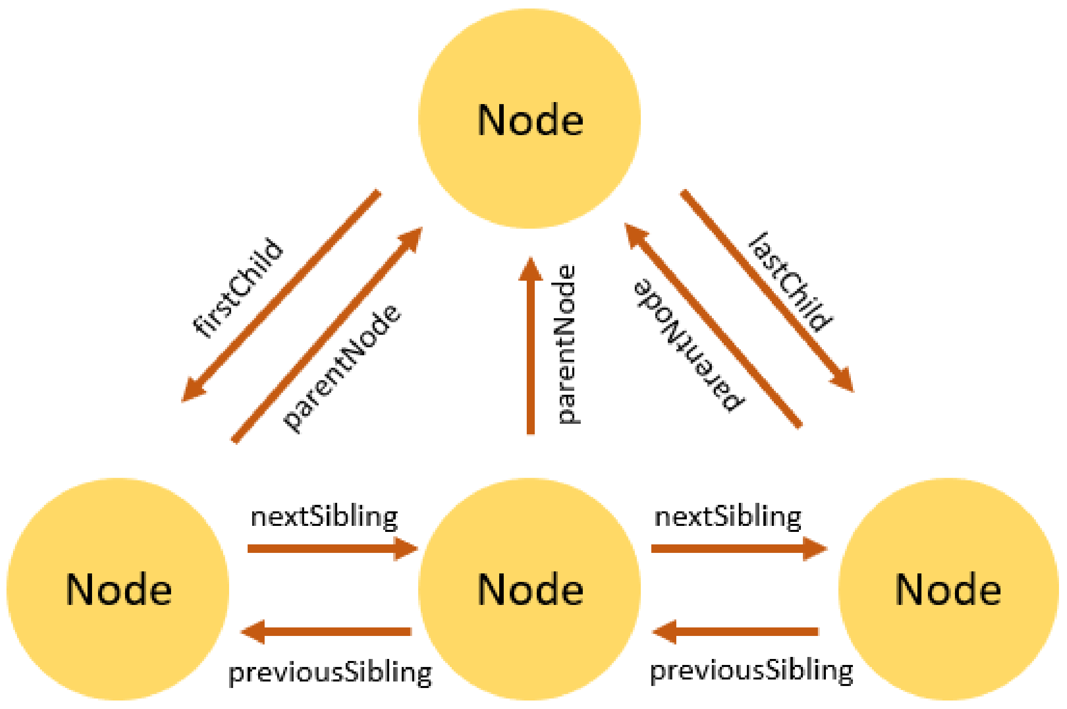
Node Relationships

Any node has relationships to other nodes in the DOM tree. The relationships are the same as the ones described in a traditional family tree.

For example, `<body>` is a child node of the `<html>` node, and `<html>` is the parent of the `<body>` node.

The `<body>` node is the sibling of the `<head>` node because they share the same immediate parent, which is the `<html>` element.

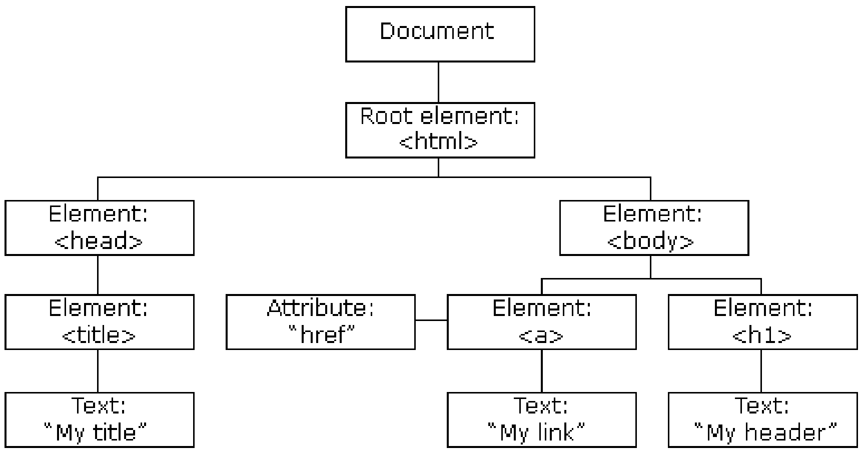
The following picture illustrates the relationships between nodes:



HTML DOM Elements

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:



The Element Object

In the HTML DOM, the **Element object** represents an HTML element, like P, DIV, A, TABLE, or any other HTML element. Each element object ha
certain properties and methods associated with it.

DOM Attributes/Properties

The values you can set or change the HTML elements are called HTML DOM Properties. The HTML properties are defined by DOM, which can be easi
modified by using the pre-defined HTML DOM Properties.

Now, check out the below-given table; we have compiled some of the HTML DOM Properties that can be utilized on all HTML elements:

HTML DOM Properties	Description
accessKey	The "accessKey" sets or returns the accesskey HTML DOM property.
attributes	The "attributes" HTML DOM property outputs a NamedNodeMap of the attribute of an HTML element.
activeElement	The "activeElement" property gets the element that is currently active.
childElementCount	The "childElementCount" HTML DOM property is utilized for displaying the total number of an HTML element's child elements.
childNodes	The "childNodes" HTML DOM property outputs the child nodes of an HTML element.
classList	The "classList" HTML DOM property returns an element's class name.
className	The "className" HTML DOM property is utilized for changing or displaying the HTML class name value.
clientHeight	The "clientHeight" HTML DOM property outputs the HTML element's height.
clientLeft	The "clientLeft" HTML DOM property is utilized for displaying the left border's width of an HTML element.
clientTop	The "clientTop" HTML DOM property is utilized for displaying the top border's width of an HTML element.
clientWidth	The "clientWidth" property returns or outputs the width of an element, which includes padding.
contentEditable	The "contentEditable" HTML DOM property sets or returns whether an element's content is editable or not.
dir	The "dir" HTML DOM property is utilized for changing or displaying an element's dir value.
firstChild	The "firstChild" HTML DOM property is utilized for displaying the first child node of an HTML element.
firstElementChild	The "firstElementChild" HTML DOM property is utilized for displaying the first child element of an HTML

HTML DOM Properties	Description
	element.
id	The “id” HTML DOM property is utilized for changing and displaying an element’s id attribute value.
innerHTML	The “innerHTML” HTML DOM property is utilized for changing and displaying the element’s content.
innerText	The “innerText” HTML DOM property is utilized for changing and displaying the node’s text content and its descendants.
isContentEditable	The “isContentEditable” HTML DOM property is utilized for checking if an element’s content is editable or not.
lang	The “lang” HTML DOM property is utilized for setting the lang attribute’s value of an HTML element.
lastChild	The “lastChild” property returns an elements’ last child node.
lastElementChild	The “lastElementChild” HTML DOM property is utilized to display an element’s last child element.
namespaceURI	The “namespaceURL” HTML DOM property is utilized to output the namespace URI of an HTML element.
nextSibling	The “nextSibling” HTML DOM property is utilized for displaying the next node in a tree level.
nextElementSibling	The “nextElementSibling” HTML DOM property is utilized for displaying the next HTML element in a tree level.
nodeName	The “nodeName”HTML DOM property outputs the node name.
nodeType	The “nodeType” HTML DOM property is utilized for displaying the node type.
nodeValue	The “nodeValue” HTML DOM property is utilized for displaying or changing the node value.
ownerDocument	The “ownerDocument” property sets or returns the root element or document object for an element.
parentNode	The “parentNode” HTML DOM property is utilized for displaying the parent node of an element.
parentElement	The “parentElement” HTML DOM property is utilized for displaying the HTML element’s parent element.
previousSibling	The “previousSibling” HTML DOM property is utilized for displaying the HTML element’s previous sibling.
previousElementSibling	The “previousElementSibling” HTML DOM property is utilized for displaying the sibling of the previous HTML element.
scrollHeight	The “scrollHeight” HTML DOM property is utilized for displaying the height of an HTML element.
style	The “style” HTML DOM property is utilized for changing or displaying the attribute’s value of an HTML element.
tabIndex	The “tabIndex” HTML DOM property is utilized for changing or displaying the attribute’s tab index value of an HTML element.
tagName	The “tagName” HTML DOM property is utilized for displaying an element’s tag name.
textContent	The “textContent” HTML DOM property is utilized for changing or displaying a node’s textual content.
title	The “title” is utilized for changing or displaying the title attribute’s value of the HTML element.
scripts	The “scripts” property is used to get all the script elements.
strictErrorChecking	The “strictErrorChecking” property is utilized for setting the error checking to strict mode.
URL	The “URL” HTML DOM property is utilized for getting the full URL of an HTML document.

DOM Methods

The "**actions**" you perform on HTML elements are called HTML DOM Methods. You can use the HTML DOM methods as a JavaScript beginner f generating the required results without putting in too much effort and writing a bunch of code.

In the below-given table, we have compiled some of the HTML DOM Methods that can be utilized on all HTML elements:

HTML DOM Methods	Description
addEventListener()	The "addEventListener()" HTML DOM method is utilized for attaching an event handler to a particular element.
appendChild()	The "appendChild()" method is used to add a new child node for an element.
blur()	The "blur()" method is used to eliminate focus from an HTML element.
click()	The "click()" method puts on a mouse-click on an HTML element.
cloneNode()	The "cloneNode()" method is utilized to clone an element.
closest()	The "closest()" HTML DOM method is utilized for searching the DOM tree for the closest element, which matches with a specified CSS selector.
compareDocumentPosition()	The "compareDocumentPosition()" HTML DOM method is utilized for comparing the document position of any two HTMLelements.
exitFullscreen()	The "existFullscreen()" property cancels an element in a fullscreen mode.
focus()	The "focus()" HTML DOM method for assigning focus to an HTML element.
getAttribute()	The "getAttribute()" HTML DOM method displays an element's attribute value.
getAttributeNode()	The "getAttributeNode()" HTML DOM method displays the specified attribute node.
getBoundingClientRect()	The "getBoundingClientRect()" HTML DOM method outputs the position and size of an element related to the viewport.
getElementsByClassName()	The "getElementByClassName()" HTML DOM method displays the list of child elements of a particular class.
getElementsByTagName()	The "getElementsByTagName()" HTML DOM method displays the list of child elements of a particular tag.
hasAttribute()	The "hasAttribute()" HTML DOM method outputs true if an element has the added attribute, otherwise, this method returns false.
hasChildNodes()	The "hasChildNodes()" HTML DOM method outputs true if a particular element has any child nodes, otherwise, this method returns false.
insertAdjacentElement()	The "insertAdjacentElement()" HTML DOM method is utilized for inserting a HTML element at a particular position related to the active element.
insertAdjacentHTML()	The "insertAdjacentHTML()" HTML DOM method is utilized for inserting an HTML formatted text at a particular position related to the active element.
insertAdjacentText()	The "insertAdjacentText()" HTML DOM method is utilized for inserting text at a particular position related to the active element.
insertBefore()	The "insertBefore()" HTML method is used for inserting a new child node before an existing child node.
isDefaultNamespace()	The "isDefaultNamespace()" HTML DOM method outputs true if a particular namespaceURI is set as default, otherwise, this method outputs false.
isEqualNode()	The "isEqualNode()" method is used for checking if two HTML elements are equal or not.
isSameNode()	The "isSameNode()" HTML DOM method is utilized for verifying if two HTML elements are the same node or not.

HTML DOM Methods	Description
isSupported()	The “isSupported()” HTML DOM method outputs true if a particular feature is supported for the HTML element.
matches()	The “matches()” method outputs a Boolean value indicating whether a specific CSS selector matches an element or not.
normalize()	The “normalize()” HTML DOM method is utilized for joining adjacent text nodes and for eliminating empty text nodes from a HTML element.
querySelector()	The “querySelector()” HTML DOM method is utilized to output the first child element that gets matched with a particular CSS selector of an HTML element.
querySelectorAll()	The “querySelectorAll()” method returns all child elements that match a specified CSS selector of an element.
remove()	The “remove()” HTML DOM method is utilized for removing a particular HTML element from DOM.
removeAttribute()	The “removeAttribute()” HTML DOM method is utilized for removing a particular attribute from a HTML element.
removeAttributeNode()	The “removeAttributeNode()” HTML DOM method eliminates the added attribute node and shows it as output.
removeChild()	The “removeChild()” HTML DOM method is utilized for removing a particular a child node from an HTML element.
removeEventListener()	The “removeEventListener()” HTML DOM method is utilized for removing an attached event handler.
replaceChild()	The “replaceChild()” HTML DOM method is used for replacing a child node from an HTML element.
requestFullscreen()	The “requestFullscreen()” HTML DOM method is utilized for showing the HTML element in fullscreen mode.
scrollIntoView()	The “scrollIntoView()” HTML DOM method is utilized for scrolling a particular HTML element into the browser’s visible area.
setAttribute()	The “setAttribute()” HTML DOM method is utilized for setting the attributes to the added value.
setAttributeNode()	The “setAttributeNode()” HTML DOM method is utilized for changing or displaying the specified attribute node.
toString()	The “toString()” HTML DOM method is utilized for converting an element to a string.

Accessing DOM Elements

The DOM API provides us with some methods through which we can access the elements.

Lets see a few of them.

`getElementById()` method:

The `document.getElementById()` method returns an `Element` object that represents an HTML element with an id that matches a specified string.

If the document has no element with the specified id the `document.getElementById()` returns `null`

If multiple elements have the same `id` , even though it is invalid, the `getElementById()` returns the first element it encounters.

Suppose you have the following HTML document:

```
<html>
    <head>
        <title>JavaScript getElementById() Method</title>
    </head>
    <body>
        <p id="message">A paragraph</p>
    </body>
</html>
```

The document contains a `<p>` element that has the `id` attribute with the value `message` :

```
const p = document.getElementById('message');
console.log(p);
```

OUTPUT:

```
<p id="message">A paragraph</p>
```

`getElementsByName()` method:

The `getElementsByName()` accepts a `name` which is the value of the `name` attribute of elements and returns a live `NodeList` of elements.

The return collection of elements is live. It means that the return elements are automatically updated when elements with the same nam are inserted and/or removed from the document.

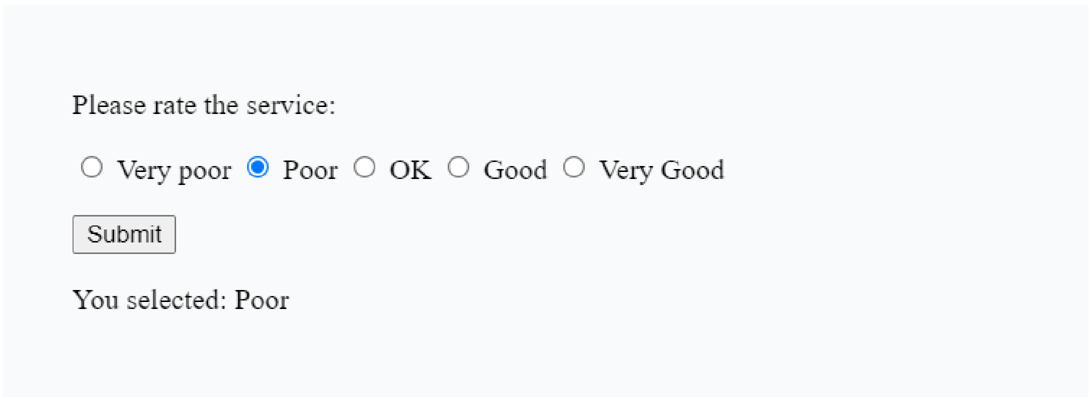
The following example shows a radio group that consists of radio buttons that have the same name (`rate`).

When you select a radio button and click the submit button, the page will show the selected value such as `Very Poor` , `Poor` , `OK` , `Good` , or `Ver Good` :

> Notice that you will learn about events like `click` later. For now, you just need to focus on the `getElementsByName()` method.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>JavaScript getElementsByName Demo</title>
</head>

<body>
    <p>Please rate the service:</p>
    <p>
        <label for="very-poor">
            <input type="radio" name="rate" value="Very poor" id="very-poor"> Very poor
        </label>
        <label for="poor">
            <input type="radio" name="rate" value="Poor" id="poor"> Poor
        </label>
        <label for="ok">
            <input type="radio" name="rate" value="OK" id="ok"> OK
        </label>
        <label for="good">
            <input type="radio" name="rate" value="Good"> Good
        </label>
        <label for="very-good">
            <input type="radio" name="rate" value="Very Good" id="very-good"> Very Good
        </label>
    </p>
    <p>
        <button id="btnRate">Submit</button>
    </p>
    <p id="output"></p>
    <script>
        let btn = document.getElementById('btnRate');
        let output = document.getElementById('output');

        btn.addEventListener('click', () => {
            let rates = document.getElementsByName('rate');
            rates.forEach((rate) => {
                if (rate.checked) {
                    output.innerText = `You selected: ${rate.value}`;
                }
            });

        });
    </script>
</body>

</html>
```

OUTPUT:



Please rate the service:

○ Very poor ● Poor ○ OK ○ Good ○ Very Good

Submit

You selected: Poor

`getElementsByTagName()` method:

The `getElementsByTagName()` is a method of the `document` object or a specific DOM element.

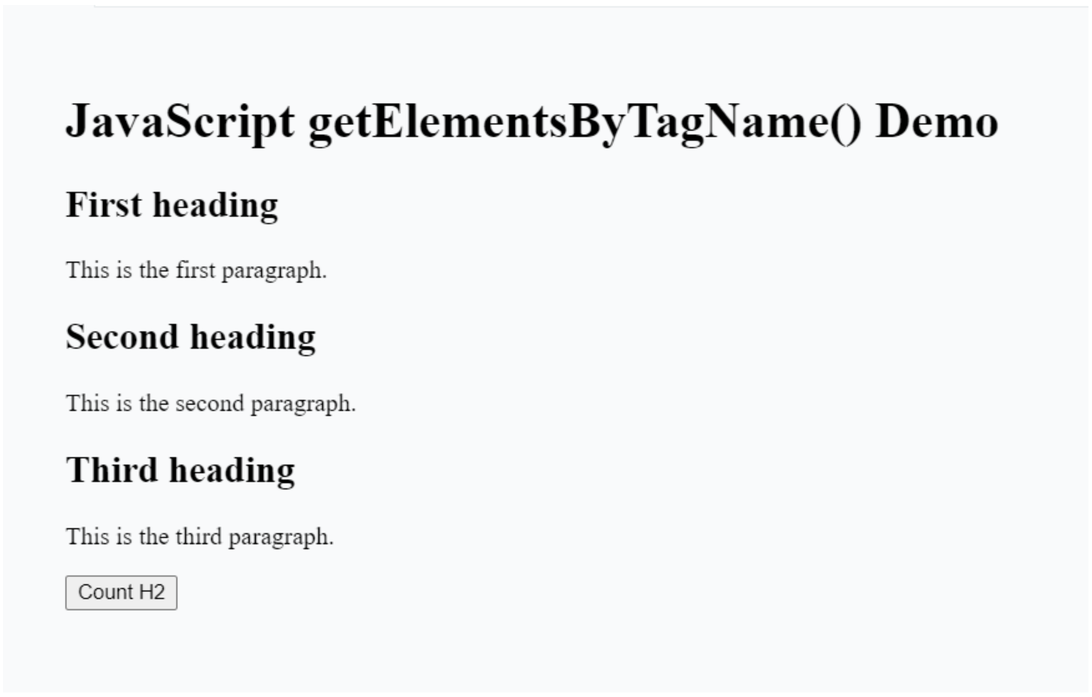
The `getElementsByTagName()` method accepts a tag name and returns a live `HTMLCollection` of elements with the matching tag name in th order which they appear in the document.

The following example illustrates how to use the `getElementsByTagName()` method to get the number of H2 tags in the document.

When you click the **Count H2** button, the page shows the number of H2 tags:

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript getElementsByTagName() Demo</title>
</head>
<body>
    <h1>JavaScript getElementsByTagName() Demo</h1>
    <h2>First heading</h2>
    <p>This is the first paragraph.</p>
    <h2>Second heading</h2>
    <p>This is the second paragraph.</p>
    <h2>Third heading</h2>
    <p>This is the third paragraph.</p>

    <button id="btnCount">Count H2</button>

    <script>
        let btn = document.getElementById('btnCount');
        btn.addEventListener('click', () => {
            let headings = document.getElementsByTagName('h2');
            alert(`The number of H2 tags: ${headings.length}`);
        });
    </script>
</body>

</html>
```

OUTPUT:



JavaScript getElementsByTagName() Demo

First heading

This is the first paragraph.

Second heading

This is the second paragraph.

Third heading

This is the third paragraph.

`getElementsByClassName()` method:

The `getElementsByClassName()` method returns an array-like of objects of the child elements with a specified class name.

```
let elements = document.getElementsByClassName(names);
```

Let’s take some examples of using the `getElementsByClassName()` method.

Suppose that you have the following HTML document:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JavaScript getElementsByClassName</title>
</head>
<body>
    <header>
        <nav>
            <ul id="menu">
                <li class="item">HTML</li>
                <li class="item">CSS</li>
                <li class="item highlight">JavaScript</li>
                <li class="item">TypeScript</li>
            </ul>
        </nav>
        <h1>getElementsByClassName Demo</h1>
    </header>
    <section>
        <article>
            <h2 class="secondary">Example 1</h2>
        </article>
        <article>
            <h2 class="secondary">Example 2</h2>
        </article>
    </section>
</body>
</html>
```

The following example illustrates how to use the `getElementsByClassName()` method to select the `<li>` items which are the descendants the `<ul>` element:

```
let menu = document.getElementById('menu');
let items = menu.getElementsByClassName('item');

let data = [].map.call(items, item => item.textContent);

console.log(data);
```

OUTPUT:

```
['HTML', 'CSS', 'JavaScript', 'TypeScript']
```

`querySelector() and querySelectorAll` method:

The `querySelector()` is a method of the `Element` interface. The `querySelector()` method allows you to select the first element that matche one or more CSS selectors.

```
let element = parentNode.querySelector(selector);
```

Besides the `querySelector()`, you can use the `querySelectorAll()` method to select all elements that match a CSS selector or a group of CS selectors:

```
let elementList = parentNode.querySelectorAll(selector);
```

Basic selectors

Suppose that you have the following HTML document:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>querySelector() Demo</title>
</head>
<body>
    <header>
        <div id="logo">
            <img src="img/logo.jpg" alt="Logo" id="logo">
        </div>
        <nav class="primary-nav">
            <ul>
                <li class="menu-item current"><a href="#home">Home</a></li>
                <li class="menu-item"><a href="#services">Services</a></li>
                <li class="menu-item"><a href="#about">About</a></li>
                <li class="menu-item"><a href="#contact">Contact</a></li>
            </ul>
        </nav>
    </header>
    <main>
        <h1>Welcome to the JS Dev Agency</h1>

        <div class="container">
            <section class="section-a">
                <h2>UI/UX</h2>
                <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Autem placeat, atque accusamus voluptas
                    laudantium facilis iure adipisci ab veritatis eos neque culpa id nostrum tempora tempore minima.
                    Adipisci, obcaecati repellat.</p>
                <button>Read More</button>
            </section>
            <section class="section-b">
                <h2>PWA Development</h2>
                <p>Lorem ipsum dolor sit, amet consectetur adipisicing elit. Magni fugiat similique illo nobis quibusdam
                    commodi aspernatur, tempora doloribus quod, consectetur deserunt, facilis natus optio. Iure
                    provident labore nihil in earum.</p>
                <button>Read More</button>
            </section>
            <section class="section-c">
                <h2>Mobile App Dev</h2>
                <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Animi eos culpa laudantium consequatur ea!
                    Quibusdam, iure obcaecati. Adipisci deserunt, alias repellat eligendi odit labore! Fugit iste sit
                    laborum debitis eos?</p>
                <button>Read More</button>
            </section>
        </div>
    </main>
    <script src="js/main.js"></script>
</body>
</html>
```

1) Universal selector

The universal selector is denoted by `*` that matches all elements of any type:

The following example uses the `querySelector()` selects the first element in the document:

```
let element = document.querySelector('*');
```

And this select all elements in the document:

```
let elements = document.querySelectorAll('*');
```

2) Type selector

To select elements by node name, you use the type selector e.g., `a` selects all `<a>` elements:

The following example finds the first `h1` element in the document:

```
let firstHeading = document.querySelector('h1');
```

3) Class selector

To find the element with a given CSS class, you use the class selector syntax `.className`

The following example finds the first element with the `menu-item` class:

```
let note = document.querySelector('.menu-item');
```

4) ID Selector

To select an element based on the value of its id, you use the id selector syntax `#id`

The following example finds the first element with the id `#logo` :

```
let logo = document.querySelector('#logo');
```

5. Attribute Selector

To select all elements that have a given attribute, you use one of the following attribute selector syntaxes:

`[attribute]` `[attribute=value]` `[attribute~=valu`
`[attribute|=value]` `[attribute^=value]` `[attribute$=value]` `[attribute*$*=value]`

The following example finds the first element with the attribute `[autoplay]` with any value:

```
let autoplay = document.querySelector('[autoplay]');
```

Grouping Selectors

To group multiple selectors, you use the following syntax: `selector, selector, …`

The selector list will match any element with one of the selectors in the group.

The following example finds all `<div>` and `<p>` elements:

```
let elements = document.querySelectorAll('div, p');
```

Combinators

1) Descendant Combinator:

To find descendants of a node, you use the space ( ) descendant combinator syntax:

`selector selector`

For example `p a` will match all `<a>` elements inside the `p` element:

```
let links = document.querySelector('p a');
```

2) Child Combinator:

The `>` child combinator finds all elements that are direct children of the first element:

`selector > selector`

The following example finds all `li` elements that are directly inside a `<ul>` element:

```
let listItems = document.querySelectorAll('ul > li');
```

To select all `li` elements that are directly inside a `<ul>` element with the class `nav` :

```
let listItems = document.querySelectorAll('ul.nav > li');
```

3) General Sibling Combinator:

The `~` combinator selects siblings that share the same parent: `selector ~ selector`

For example, `p ~ a` will match all `<a>` elements that follow the `p` element, immediately or not:

```
let links = document.querySelectorAll('p ~ a');
```

4) Adjacent Sibling:

The `+` adjacent sibling combinator selects adjacent siblings: `selector + selector`

For example, `h1 + a` matches all elements that directly follow an `h1` :

```
let links = document.querySelectorAll('h1 + a');
```

Pseudo

1) Pseudo Classes:

The `:` pseudo matches elements based on their states: `element:state`

For example, the `li:nth-child(2)` selects the second `<li>` element in a list:

```
let listItem = document.querySelectorAll('li:nth-child(2)');
```

2) Pseudo Elements:

The `::` represent entities that are not included in the document.

For example, `p::first-line` matches the first-line of all `div` elements:

```
let links = document.querySelector('p::first-line');
```

Get the `parentNode` :

To get the parent node of a specified node in the DOM tree, you use the `parentNode` property:

The `Document` and `DocumentFragment` nodes do not have a parent. Therefore, the `parentNode` will always be `null` .

If you create a new node but haven't attached it to the DOM tree, the `parentNode` of that node will also be `null` .

See the following HTML document:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>JavaScript parentNode</title>
</head>
<body>
    <div id="main">
        <p class="note">This is a note!</p>
    </div>

    <script>
        let note = document.querySelector('.note');
        console.log(note.parentNode);
    </script>
</body>
</html>
```

The following picture shows the output in the Console:

Get the Children

To get the first child element of a specified element, you use the `firstChild` property of the element.Or to get the first child with the Element noc only, you can use the `firstElementChild` property.

To get the last child element of a node, you use the `lastChild` property.If you want to select only the last child element with the element node typ you use the `lastElementChild` property.

The `childNodes` property returns all child elements with any node type. To get the child element with only the element node type, you us the children property.

Suppose that you have the following HTML fragment:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>JS Get Child Elements</title>
</head>
<body>
  <ul id="menu">
    <li class="first">Home</li>
    <li>Products</li>
    <li class="current">Customer Support</li>
    <li>Careers</li>
    <li>Investors</li>
    <li>News</li>
    <li class="last">About Us</li>
  </ul>
</body>
</html>
```

First Child:

The following script shows the first child of the #menu element:

```
let content = document.getElementById('menu');
let firstChild = content.firstChild.nodeName;
console.log(firstChild);
```

Last Child:

The following code returns the list item which is the last child element of the menu:

```
let content = document.getElementById('menu');
let lastChild = content.lastChild.nodeName;
console.log(firstChild);
```

Get All Child Elements:

The following example selects all child elements of the element with the Id main :

```
let menu = document.getElementById('menu');
let children = menu.children;
console.log(children);
```

Get the Siblings

To get the next sibling of an element, you use the nextElementSibling attribute.

The nextElementSibling returns null if the specified element is the first one in the list. The following example use the nextElementSibling property to get the next sibling of the list item that has the current class:

```
<ul id="menu">
    <li>Home</li>
    <li>Products</li>
    <li class="current">Customer Support</li>
    <li>Careers</li>
    <li>Investors</li>
    <li>News</li>
    <li>About Us</li>
</ul>
```

```
let current = document.querySelector('.current');
let nextSibling = current.nextElementSibling;
console.log(nextSibling);
```

OUTPUT:

```
<li>Careers</li>
```

To get the previous siblings of an element, you use the `previousElementSibling` attribute.

The `previousElementSibling` property returns `null` if the current element is the first one in the list.

The following example uses the `previousElementSibling` property to get the previous siblings of the list item that has the `current` class:

```
let current = document.querySelector('.current');
let prevSibling = current.previousElementSibling;
console.log(prevSibling);
```

OUTPUT:

```
<li>Products</li>
```

So, till now we have learned the fundamentals of DOM and hence now we are good enough to create any web page and make it more interactive.

In this module we have learned about:

- Basics of DOM and its different elements
- Structure of HTML DOM.
- Different DOM Attributes and Methods.
- How to access HTML elements using DOM

So, now from the next module we will be learning about DOM Events and Manipulation.

Interview Questions

> What are the HTML DOM methods involved?

The following are HTML DOM methods that are mostly used.

- **getElementById(idName) – this method allows you to access or find an element associated with id name defined in parenthesis().** Example – document.getElementById("demo")
- **getElementsByClassName(className) – this method allows you to access or find elements associated with className defined in parenthesis().** Example – document.getElementsByClassName("main")
- **getElementsByTagName(tagName) – this method allows you to access or find elements associated with tagName defined in parenthesis().** Example – document.getElementsByTagName("p")
- **appendChild(element) – this method allows you to add a new element(node) in DOM(Document Object Model) tree structure as the last child of a node.** Example – document.appendChild(*element*)
- **removeChild(element) – this method allows you to remove a child element(node) in DOM(Document Object Model) tree structure.** Example – document.appendChild(*element*)
- **createChild(element) – this method allows you to create an element(node) in DOM(Document Object Model) tree structure.** Example – document.createElement(*element*)
- **replaceChild(new, old) – this method allows you to replace an old element with a new element in DOM(Document Object Model) tree structure.** Example – document.replaceChild(*new, old*)

> How can I find the number of elements (length) in the HTML collection?

To find the number of elements in HTML collection, we have to use length properties.

```
<body>
```

```
<p>This is paragraph one. </p>

<p>  This is paragraph two. </p>

<p>  This is paragraph three. </p>

<p id=”demo”></p>

<script>

var x = document.getElementsByTagName(“p”);

document.getElementById(“demo”).innerHTML = x.length;

</script>

</body>
```

The output will be –

4

Thank you !