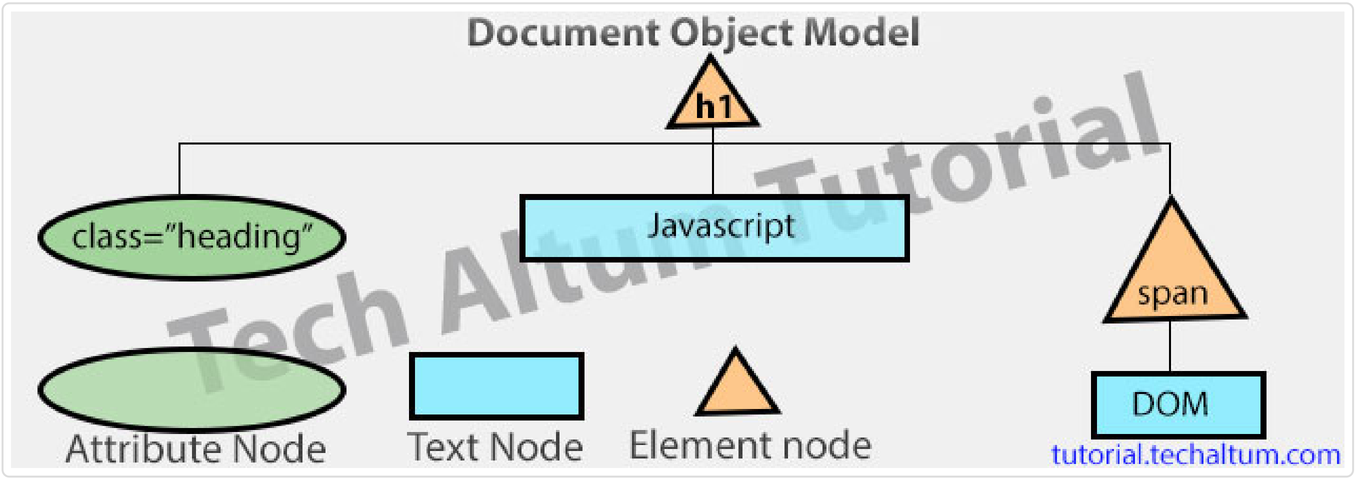
JavaScript DOM

Document Object Model or **DOM** is a **Browser API** to get, change and update the HTML Elements or attributes on runtime in a browser window. We can see these changes in browser inspect, but not in source code. **DOM** is a *Tree Like Structure* of HTML Document, with connected nodes, as shown below. Everything in a webpage page is treated as a node, i.e. HTML tags, text inside, attributes and child tag. HTML Tag, (<*HTML*>) is the Root Node, and all other elements are descendant node of html tag. Even doctype and comments are also nodes.

The webpage is document here and all elements and attributes inside are document nodes. These nodes can be element node, text node, attribute node, etc. Using **document**, we can manipulated the content in webpage.



JavaScript Document Object Model *(DOM)*

To access **DOM Element**, or Elements, `document` or `window.document` keyword is used.

DOM Methods

- document
- document.write
- getting elements
- getElementById
- getElementsByTagName
- getElementsByClassName
- querySelector
- querySelectorAll
- createElement

DOM Element Example

```
<h1 class="heading">Javascript <span>DOM</span></h1>

 h1         = Element Node
 class      = Attribute Node
 Javascript = Text Node
 span       = Child Node
 DOM        = Text Node
```

## document

JS document object is build-in object in client side javascript to access html elements in javascript. **document object** can get anything in webpage loaded, like elements, attributes, doctype, comments, charset, etc.

document object is only available in client side javascript, not in Node JS. The **source code** is the code downloaded from web browser. But the code available in **browser inspect** in elements tab is DOM. DOM can be manipulation, but source code in not.

▶#document

```
    console.log(document);
```

document.write

document.write() method is used to write something in current **document** or **DOM** node.
If `<script>` tag is inside body, **document.write()** will write plain text in body node. But if `<script>` tag is in another element, like p, div etc, **document.write()** will write plain text in that p or div element node.

If webpage is already loaded, **document.write()** method will replace the current document.

Hello JS

```
document.write("Hello JS");
```

Hello JS in h1 tag

```
document.write("<h1>Hello JS in h1 tag</h1>");
```

Hello JS, with *i tag* in p.

```
document.write("<p>Hello JS, with <i>i tag</i> in p.</p>");
```

As **document.write()** method will replace the current document, we usually avoid using **document.write()** method.

Getting Elements from DOM

`document method` or `window.document` is used to access any element in a webpage. These Method returns a node object or a node list(array like structure) or `null` (if id is not found).

document.body

```
var x=document.body;
x.nodeName              // return BODY
```

document.title

Get webpage title

```
var x=document.title;            // return title of webpage
```

Change webpage title

```
document.title="new title";
```

document.images

```
var y=document.images;
y.length       // return no of images
y[0].nodeName  // return IMG
y[0].nodeType  // return 1
```

DOM Shortcut Methoods

Method	**Use**
document.body	To access body element
document.head	To access head element
document.characterSet	return character set of web document
document.images	return list of all images in document.

document.links	return list of all hyperlinks in document.
document.anchors	return list of only anchor tags in document.
document.forms	return list of form tags in document.
document.contact	return form element with name="contact" in document. `<form name="contact"></form>`
document.contact.username	return form control with name="username" on contact form of document. `<form name="contact"> <input type="text" name="username"> </form>`

nodeType

Code	Type
1	element
2	attribute
3	text
8	comment
9	document
10	doctype

`document.body` is node object. But `document.images` is array-like object. To access individual element inside list, use index notation. Exp, `document.images[0]` means first image of webpage.

Get Element By Id

`document.getElementById()` return the element with unique id that is given as an argument. It is supported in all major browsers. As per w3c specifications, id attribute and id's value is unique. **Get Element By Id** will call an element with his unique id.

```
<h1 id="heading">Javascript DOM</h1>
<script>
 var x=document.getElementById("heading");
</script>
```

Get Elements By Tag Name

`document.getElementsByTagName()` return list of all elements with tag name given as an argument. It is supported in all major browsers.

```
<p> Para 1</p>
<p> Para 2</p>
<script>
 var x=document.getElementsByTagName("p");
 x.length;                      // return 2,
 var p1=x[0];        //return first p element;
 var p2=x[1];        //return second p element;
</script>
```

Get Elements By Class Name

`document.getElementsByClassName()` return list of all elements with class name given as an argument. It is supported in all major browsers, except IE 8 and below doesn't support.

```
<p class="para"> Para 1</p>
<p class="para"> Para 2</p>
<script>
 var x=document.getElementsByClassName("para");
 var p1=x[0];        //return first p element;
 var p2=x[1];        //return second p element;
</script>
```

The datatype of document.getElementsByTagName and document.getElementsByClassName is an **array-like objects**. Array like means they have length property and indexed elements. But if we check their datatype, it is not array.

Query Selector

`document.querySelector()` return first element in document with **tag, id, class** or any css selector given as an argument. **querySelector** is pure CSS selector based. To call element by id, use "#id", to call an element with class name, use ".classname" and to call element by tagname, use "tagname". It is supported in all major browsers, except IE 7 and below. IE 8 and above support CSS 2.1, but I.E 7 doesn't support querySelector.

```
<p class="para" id="para1"> Para 1</p>
<p class="para" id="para2"> Para 2</p>

<script>
document.querySelector('p');                                  first p tag;
document.querySelector('#para1');                                 id para1
document.querySelector('.para');                           first para class
document.querySelector('ul li');                        first li of first ul
document.querySelector('ol > li');                      first child li of ol
document.querySelector('[disabled]');                 first disabled element

document.querySelector('p:last-child');                 last child p element
document.querySelector('p:nth-child(2)');                   second p element
</script>
```

Query Selector All

`document.querySelectorAll()` return list of elements in document with tag, id, or class name given as an argument. `querySelectorAll()` returns data in array-like structure, but not array. It is pure CSS selector based. **document.querySelectorAll** method is supported in all major browsers, except IE 7 and below. IE 8 and above support CSS 2.1, but 7 doesn't.

```
<p class="para"> Para 1</p>
<p class="para"> Para 2</p>
<p> Para 3</p>

<script>
 var x=document.querySelectorAll('p');                            // all p elements

 var y=document.querySelectorAll('.para');                      // all p with class para

 x.length         // return 3
 y.length         // return 2

</script>
```

Iterate each element in querySelectorAll

To **iterate each element in querySelectorAll** , we should use **forEach** method. forEach is a loop used to iterate Arrays or Objects. **querySelectorAll** returns a **Array Like** list, which supports **forEach** Method.

forEach methods first parameter is Callback Function with parameter in Callback.

Para 1

Para 2

Para 3

```

    <p> Para 1</p>
    <p> Para 2</p>
    <p> Para 3</p>

<script>
    document.querySelectorAll('p').forEach(function(i){
        console.log(i);
    });
</script>
```

`querySelector`, `querySelectorAll` are supported in IE 8 and above browsers. `getElementsByClassName` is supported in IE 9 and above browsers only.

Navigating DOM TREE

Javascript can **Navigate DOM**. All Node Objects have various properties and methods. All these nodes are interconnected. An element node can have *children nodes*, *sibling nodes*, *parent nodes* etc.

Node Property	Use
childNodes	return list of all children nodes connected, including text nodes.
children	return only element nodes of children connected, excluding text nodes
firstElementChild	return first child element of node
lastElementChild	return last element child of a node
parentNode	return parent node of a node
nextElementSibling	return next adjacent element sibling of a node, will return null if node is first child.
previousElementSibling	return previous adjacent element sibling of a node, will return null if node is first child.

DOM Node Properties

Node Property	Use
innerHTML	get text content and children nodes of element as string
textContent	get text content of node as string
value	get value of inputs, select, checkbox, radio buttons, textarea as `string`
checked	check checked state of radio and checkbox as `true` or `false`
id	get value of id attribute
classList	get values of class attribute in array
className	get values of class attribute as string
src	get source of img, audio, video, iframe.

width	width of img, video, iframe.
height	height of img, video, iframe.
alt	alt of img tag.
size	size of input and select.
maxLength	max-length input and textarea.
dataset	object with all custom data attributes

DOM Attributes

To get value of an attribute, change attribute value and **remove attribute**, **JavaScript** is having following methods.

Node Property	**Use**
getAttribute()	get attribute value, parameter should be attributes name. `document.body.getAttribute("title");`
setAttribute()	set attributes value, two parameter required `document.body.setAttribute("title","this is body");`
removeAttribute()	remove attribute and attributes value. one parameter required `document.body.removeAttribute("title");`

Always use two parameters in setAttribute. For boolean attributes, use blank as second parameter. See exp `document.querySelector("#username").setAttribute("disabled","");`

createElement

document.createElement is used to **create a new element in dom**. Any html element or custom element can be created using **createElement method**.

appendChild method is used to append the element created.

```
var ptag=document.createElement("p");    // create element
document.body.appendChild(ptag);         // append element
```

createElement Example

This is marquee

```
<div class="marquee"></div>
<script>
    var m=document.createElement("marquee");
    m.innerHTML="This is marquee";
    document.querySelector(".marquee").appendChild(m);
</script>
```