DOM

Section 1. Understanding the document object model in javascript

INTRODUCTION

What is Document Object Model (DOM)

The Document Object Model (DOM) is an application programming interface (API) for manipulating HTML documents.

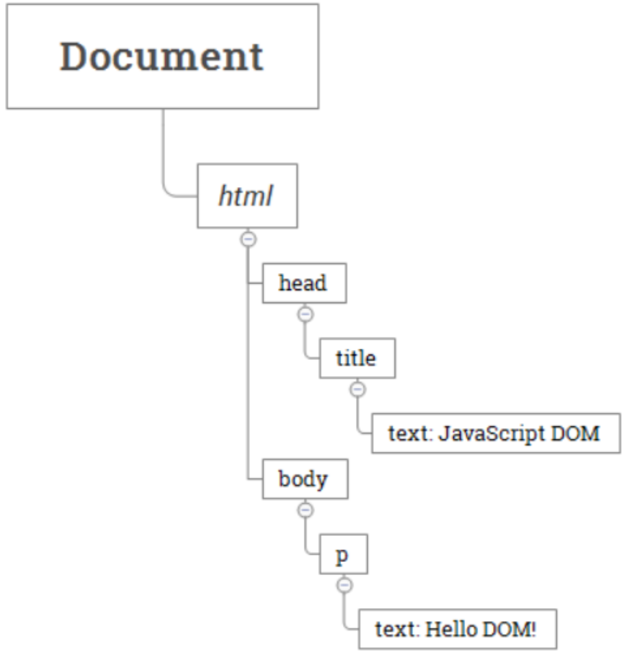
The DOM represents an HTML document as a tree of nodes. The DOM provides functions that allow you to add, remove, and modify parts of the document effectively.

INTRODUCTION

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>
```



NODE RELATIONSHIPS

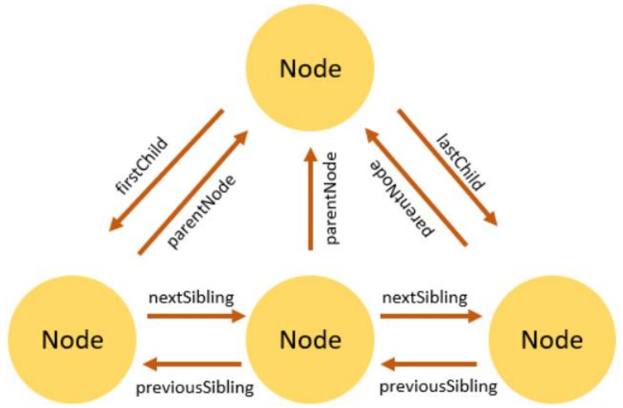
What is Document Object Model (DOM)

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:



Section 2.
Selecting elements

1ST

getElementById() – select an element by id.

2ND

getElementsByName() – select elements by name.

3RD

getElementsByTagName() – select elements by a tag name.

4TH

getElementsByClassName() – select elements by one or more class names.

5TH

querySelector() – select elements by CSS selectors.

SELECTING ELEMENTS

1st javascript 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.

Because the id of an element is unique within an HTML document, the document.getElementById() is a quick way to access an element.

Javascript getElementById() method Example.

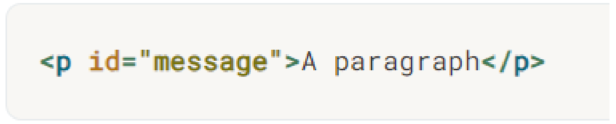
Suppose you have the following HTML document:

```
<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>
```

2nd javascript getElementsByName() method

Every element on an HTML document may have a name attribute:

```
<input type="radio" name="language" value="JavaScript">
```

Unlike the **id** attribute, multiple **HTML** elements can share the same value of the name attribute like this:

```
<input type="radio" name="language" value="JavaScript">
<input type="radio" name="language" value="TypeScript">
```

To get all elements with a specified name, you use the getElementsByName()

```
let elements = document.getElementsByName(name);
```

The getElementsByName() returns a live NodeList of elements with a specified name.

The NodeList is an array-like object, not an array object.

3nd javascript getElementsByTagName() method

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

The following illustrates the syntax of the **getElementsByTagName()**:

```
let elements = document.getElementsByTagName(tagName);
```

The **getElementsByTagName()** is a method of the document or element object.

The **getElementsByTagName()** accepts a tag name and returns a list of elements with the matching tag name.

The **getElementsByTagName()** returns a live **HTMLCollection** of elements. The HTMLCollection is an array-like object.

4nd javascript 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);
```

The method returns the elements which is a live HTMLCollection of the matches elements.

Use the JavaScript getElementsByClassName() method to select the child elements of an element that has one or more give class names.

5nd javascript querySelector() and querySelectorAll() methods

The querySelector() method allows you to select the first element that matches one or more CSS selectors.

The following illustrates the syntax of the querySelector() method:

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

In this syntax, the selector is a CSS selector or a group of CSS selectors to match the descendant elements of the parentNode
If the selector is not valid CSS syntax, the method will raise a SyntaxError exception.

If no element matches the CSS selectors, the querySelector() returns null.

querySelectorAll() methods

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

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

The querySelectorAll() method returns a static NodeList of elements that match the CSS selector. If no element matches, it returns an empty NodeList.

Note that the NodeList is an array-like object, not an array object.

```
let nodeList = document.querySelectorAll(selector);
let elements = Array.from(nodeList);
```

querySelectorAll() methods

1) Universal selector:- The universal selector is denoted by * that matches all elements of any type:

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

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');
```

And the following example finds all `h2` elements:

```
let heading2 = document.querySelectorAll('h2');
```

4) ID Selector:- The following example finds the first element with the id `#logo`:

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

Since the `id` should be unique in the document, the `querySelectorAll()` is not relevant.

Section 3.
Traversing elements

1ST

Get the parent element –

get the parent node of an element.

2ND

Get child elements –

get children of an element.

3RD

Get siblings of an element –

get siblings of an element.

SECTION 3.

TRAVERSING ELEMENTS

1st JavaScript Get the Parent Element parentNode

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

```
let parent = node.parentNode;
```

```
<!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>
```

2st Getting Child Elements of a Node in JavaScript

Get the first child element

To get the first child element of a specified element, you use the **firstChild** property of the element:

```
let firstChild = parentElement.firstChild;
```

Or to get the first child with the Element node only, you can use the firstElementChild property:

```
let firstElementChild = parentElement.firstElementChild;
```

Get the last child element:- To get the last child element of a node, you use the lastChild property:

2st Getting Child Elements of a Node in JavaScript

Get all child elements

To get a live **NodeList** of child elements of a specified element, you use the **childNodes** property:

```
let children = parentElement.childNodes;
```

The childNodes property returns all child elements with any node type.

To get the child element with only the element node type, you use the children property:

```
let children = parentElement.children;
```

SUMMARY

- The **firstChild** and **lastChild** return the first and last child of a node, which can be any node type including text node, comment node, and element node.
- The **firstElementChild** and **lastElementChild** return the first and last child Element node.
- The **childNodes** returns a live **NodeList** of all child nodes of any node type of a specified node. The **children** return all child Element nodes of a specified node.

3st JavaScript Siblings

Let's say we have the following list of items:

:
```
<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>
```

Get the next siblings:

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

```
let nextSibling = currentNode.nextElementSibling;
```

JavaScript Siblings

Get all siblings of an element

To get all siblings of an element, we'll use the logic:

- First, select the parent of the element whose siblings you want
- define a function name that takes the selected element as an argument.
- initialize a variable name with the first child element of the parent.
- enter a while loop . within the loop, check if the current sibling is an element node (nodeType === 1) and not equal to the original element.
  if the condition is met, push the sibling into the array.
  move to the next sibling using sibling.nextSibling.
  continue looping until there are no more siblings.
- return the array of siblings from the function.
- call the function with the selected element as a parameter..

Section 4.
Manipulating elements

1ST

createElement() –

create a new element.

2ND

appendChild() –
append a node to a list of child nodes of a specified parent node.

3RD

textContent –
get and set the text content of a node.

4TH

innerHTML –
get and set the HTML content of an element.
innerHTML vs. createElement – explain the differences between innerHTML and createElement when it comes to creating new elements.

5TH

querySelector() – select elements by CSS selectors.

SECTION 4. MANIPULATING ELEMENTS

JavaScript CreateElement

To create an HTML element, you use the document.createElement() method:

```
let element = document.createElement(htmlTag);
```

```
let li = document.createElement('li');
li.textContent = 'Products';
```

JavaScript appendChild

To create an HTML element, you use the document.createElement() method:
The element.appendChild() appends an HTML element to an existing element.

```
parentNode.appendChild(childNode);
```

JavaScript appendChild

Reading textContent from a node:

To get the text content of a node and its descendants, you use the textContent property:

```
let note = document.getElementById('note');
console.log(note.textContent);
```

Setting textContent for a node:

Besides reading textContent, you can also use the textContent property to set the text for a node:

```
node.textContent = newText;
```

JavaScript innerHTML

The innerHTML is a property of the Element that allows you to get or set the HTML markup contained within the element:

```
element.innerHTML = 'new content';
element.innerHTML;
```

Reading the innerHTML property of an element:

To get the HTML markup contained within an element, you use the following syntax:

```
let content = element.innerHTML;
```

Setting the innerHTML property of an element:

To set the value of innerHTML property, you use this syntax:

```
const main = document.getElementById('main');

const externalHTML = `<img src='1' onerror='alert("Error loading image")'>`;
// shows the alert
main.innerHTML = externalHTML;
```

JavaScript replaceChild

To replace an HTML element, you use the **node.replaceChild()** method:

```
parentNode.replaceChild(newChild, oldChild);
```

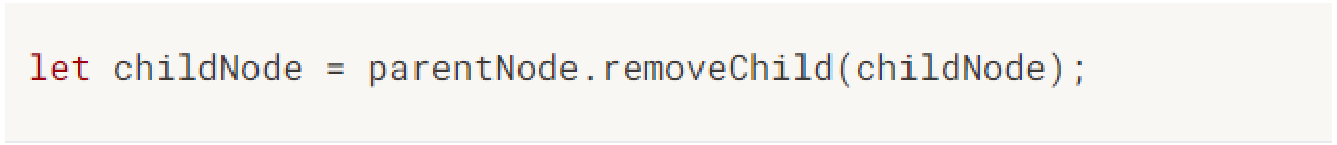
In this method, the newChild is the new node to replace the oldChild node which is the old child node to be replaced.

```
<ul id="menu">
    <li>Homepage</li>
    <li>Services</li>
    <li>About</li>
    <li>Contact</li>
</ul>
```

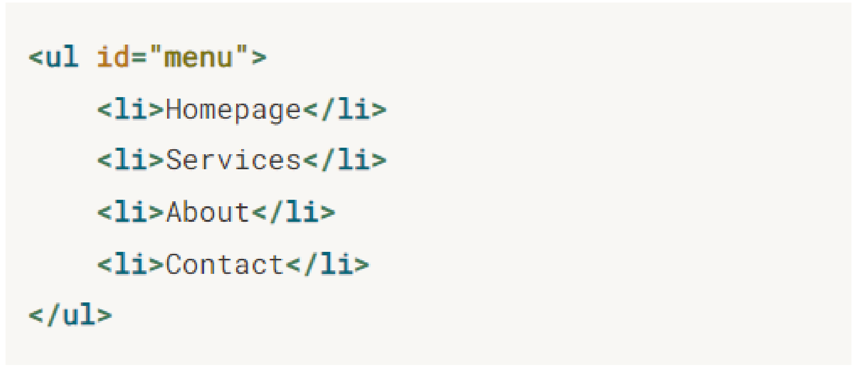
```
let menu = document.getElementById('menu');
// create a new node
let li = document.createElement('li');
li.textContent = 'Home';
// replace the first list item

menu.replaceChild(li, menu.firstElementChild);
```

JavaScript removeChild

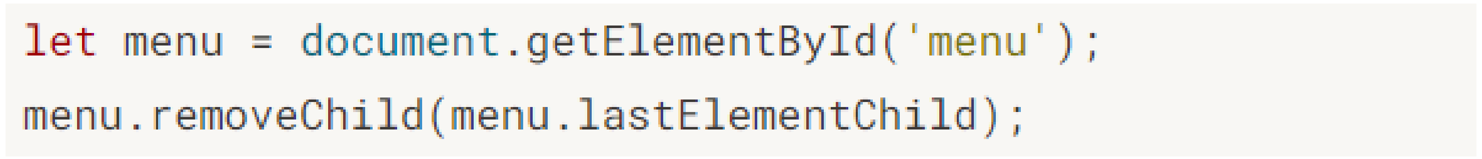
To remove a child element of a node, you use the **removeChild()** method:



```
let childNode = parentNode.removeChild(childNode);
```



```
<ul id="menu">
    <li>Homepage</li>
    <li>Services</li>
    <li>About</li>
    <li>Contact</li>
</ul>
```



```
let menu = document.getElementById('menu');
menu.removeChild(menu.lastElementChild);
```

Section 5.
Working with Attributes

1ST

setAttribute() –

set the value of a specified attribute on a element.

2ND

getAttribute() –

get the value of an attribute on an element.

3RD

removeAttribute() –

remove an attribute from a specified element.

SECTION 5. WORKING WITH ATTRIBUTES

1st JavaScript setAttribute() method

To set a value of an attribute on a specified element, you use the setAttribute() method:

```
element.setAttribute(name, value);
```

Parameters:-

- The name specifies the attribute name whose value is set. It's automatically converted to lowercase if you call the setAttribute() on an HTML element.
- The value specifies the value to assign to the attribute. It's automatically converted to a string if you pass a non-string value to the method. Example:-

- First, select the button with the id btnSend by using the querySelector() method.
- Second, set the value of the name attribute to send using the setAttribute() method.
- Third, set the value of the disabled attribute so that when users click the button, it will do nothing.

2nd JavaScript getAttribute() method

To get the value of an attribute on a specified element, you call the getAttribute() method of the element:

```
let value = element.getAttribute(name);
```

- Get the value of an attribute of a specified element by calling the getAttribute() method on the element.
- The getAttribute() returns null if the attribute does not exist.

3rd JavaScript getAttribute() method

The removeAttribute() removes an attribute with a specified name from an element:

```
element.removeAttribute(name);
```

Section 6.
Manipulating Element’s Styles

1ST

style property –

get or set inline styles of an element.

2ND

classList property –

manipulate CSS classes of an element.

SECTION 6. MANIPULATING ELEMENT’S STYLES

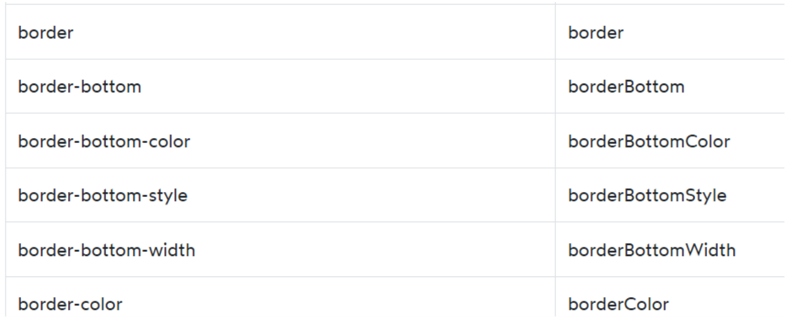
1st JavaScript Style

To set the inline style of an element, you use the **style** property of that element:

```
element.style
```

```
element.style.color = 'red';
```

The style property returns the read-only CSSStyleDeclaration object that contains a list of CSS properties.



border	border
border-bottom	borderBottom
border-bottom-color	borderBottomColor
border-bottom-style	borderBottomStyle
border-bottom-width	borderBottomWidth
border-color	borderColor

To completely override the existing inline style, you set the **cssText** property of the style object.

2nd JavaScript classList property

The classList is a read-only property of an element that returns a live collection of CSS classes:

```
const classes = element.classList;
```

The **classList** is a **DOMTokenList** object that represents the contents of the element's class attribute.

1) Get the CSS classes of an element

Suppose that you have a div element with two classes: main and red.

2) Add one or more classes to the class list of an element

To add one or more CSS classes to the class list of an element, you use the **add()** method of the **classList**.

```
let div = document.querySelector('#content');
div.classList.add('info');
```

3) Remove element's classes

To remove a CSS class from the class list of an element, you use the **remove()** method:

THANK YOU

MUHAMMAD LATIF
ML129KMG@GMAIL.COM