DOM- Document Object Model

DOM is a standard **object** model that allows programs and scripts to dynamically access and update the content, structure, and style of a document

Document Object Model (**DOM**) connects web pages to scripts languages by representing the structure of a document

The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree. With them, you can change the document's structure, style, or content.

Here's a breakdown of some key concepts related to the JavaScript DOM:

1. **Document**: The top-level object in the DOM hierarchy, representing the entire HTML document. It serves as an entry point to access and manipulate the document's content.

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

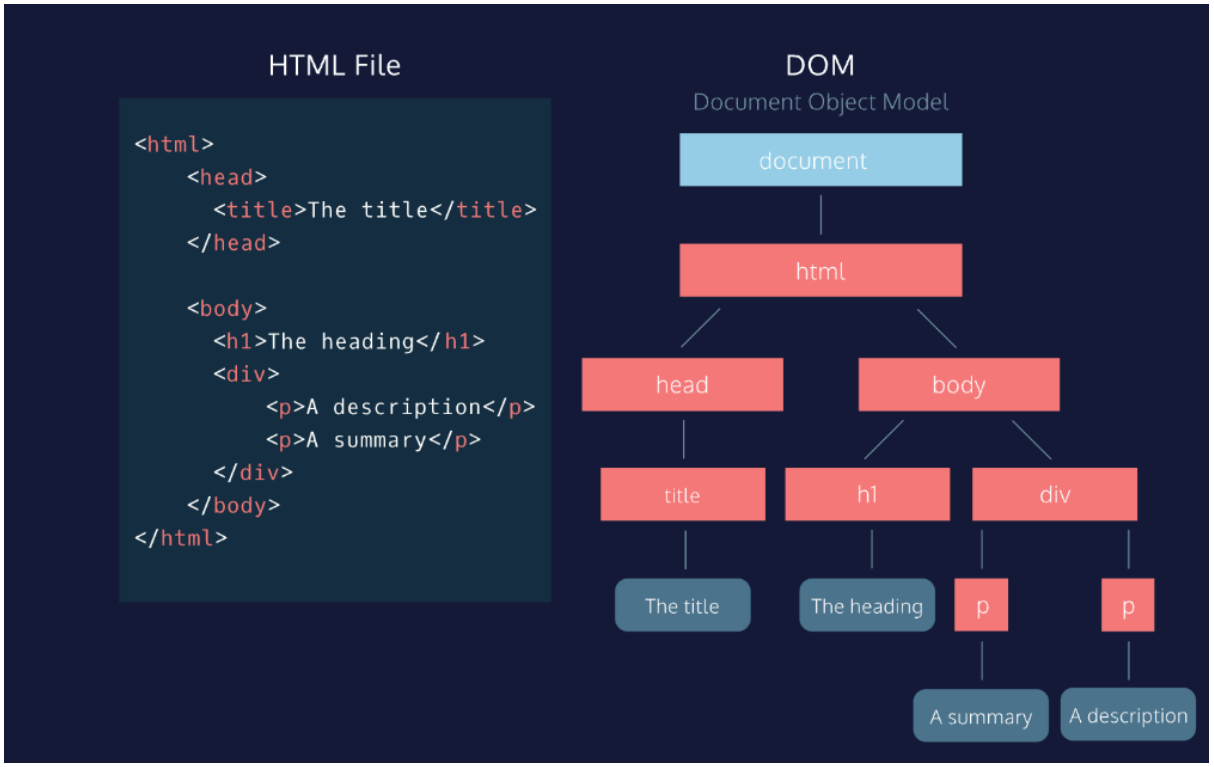
Logging **document** to the console in JavaScript will display the entire Document Object Model (DOM) of the current HTML page.

```
# document
<!DOCTYPE html>
<html>
  <head>
    <!-- Any meta tags, title, stylesheets, etc., in the head section -->
  </head>
  <body>
    <!-- The body of the HTML document -->
  </body>
</html>
```

2. **Node**: Every part of an HTML document, such as elements, attributes, and text, is represented by a node in the DOM tree. Nodes can be of different types, including element nodes, text nodes etc.
3. **Element**: Elements are the building blocks of an HTML document, such as ** **, **<p>**, ****, etc. They are represented as element nodes in the DOM tree.
4. **Attributes**: Elements can have attributes like **id**, **class**, **src**, etc. These attributes are accessible and modifiable through the DOM.
5. **Methods for Accessing Elements:**
    - document.getElementById(): Retrieves an element by its unique ID.
    - document.getElementsByClassName(): Retrieves elements by their class name.
    - document.getElementsByTagName(): Retrieves elements by their tag name.
    - document.querySelector(): Retrieves the first element that matches a CSS selector.
    - document.querySelectorAll(): Retrieves all elements that match a CSS selector
6. **Manipulating Elements**:
    - Changing element attributes (**element.attribute**).

- Changing element content (**element.innerHTML**, **element.innerText**, **element.textContent**).
- Adding or removing classes (**element.classList.add()**, **element.classList.remove()**).
- Creating new elements (**document.createElement()**).
- Appending or removing child nodes (**parentNode.appendChild()**, **parentNode.removeChild()**).

7. **Event Handling**: DOM allows attaching event handlers to elements to listen for specific events like click, hover, keypress, etc., and execute JavaScript code in response to those events.
8. **Traversing the DOM**: You can navigate through the DOM tree by accessing parent, child, or sibling nodes using properties like **parentNode**, **childNodes**, **firstChild**, **lastChild**, **nextSibling**, and **previousSibling**.



DOM

It is a object model used to manipulate the document and there are two ways to create document object

1) Field Names – document level object creation

2) Methods – element level object creation

Create dom object by field names

Property	Description
document.body	Returns all <body> element
document.head	Returns the <head> element
document.scripts	Returns all <script> elements
document.anchors	Returns all <a> elements that have a name attribute
document.forms	Returns all <form> elements
document.images	Returns all <img> elements
document.links	Returns all <area> and <a> elements that have a href
document.title	Returns the <title> element

document.tables	Returns the tables elements

Get methods using dom

1)document.getElementById(): Retrieves an element by its unique ID

```
<div id="myDiv"></div>
```

```
var elementById = document.getElementById("myDiv"); //line gets the element by id
console.log(elementById);//below is the ouput
```

```
<h1 id="demo">hello color</h1>  index.html:41
```

2)document.getElementsByClassName(): Retrieves elements by their class name.

```
<p class="myClass">Paragraph 1</p>
<p class="myClass">Paragraph 2</p>
```

```
var elementsByClassName =
document.getElementsByClassName("myClass");
console.log(elementByClassname);
```

//here the point to note is classnames are always in collections

You can get the element by their index numbers

Var elementsByClassName= document.getElementByClassName(“myClass”)[0]

```
▼ HTMLCollection(2) i                index.html:67
  ▶ 0: p.myClass
  ▶ 1: p.myClass
    length: 2
```

3) document.getElementsByTagName(): Retrieves elements by their tag name.

```
<h1>Heading</h1>
<p>Paragraph 1</p>
<p>Paragraph 2</p>
```

```
var elementsByTagName = document.getElementsByTagName("p");
console.log(elementsByTagName);
```

//here the point to note is tagnames are always in collections

You can get the element by their index numbers

Var elementsByTagName= document.getElementByTagName(“p”)[0]

```
▼ HTMLCollection(2) i                    index.html:64
  ▶ 0: p
  ▶ 1: p
    length: 2
  ▶ [[Prototype]]: HTMLCollection
```

4)Accessing Elements by CSS Selector:

```
<div class="container">
  <p class="para">Paragraph 1</p>
  <p class="para">Paragraph 2</p>
</div>
```

querySelector() method allows you to select the first element in the document

```
var elementBySelector = document.querySelector(".para");//selects by classname
var myDiv = document.querySelector("#myDiv");//select by id
var elselector = document.querySelector("div");//select by element name
```

querySelectorAll -iIt operates similarly to **querySelector**(), but instead of returning only the first matching element, it returns a list of all matching elements.

```
var paragraphs = document.querySelectorAll(".para");//select all elements by class names
```

```
var divs = document.querySelectorAll("div");//select all div elements in a collections
```

Get content of the html

innerText and **innerHTML** are properties of DOM elements in JavaScript that deal with the content of HTML elements

innerText:

- **innerText** is a property that represents the visible text content of an element.
- It retrieves the text content of the element, excluding any HTML tags.

```
<div id="myDiv">This is <span>some</span> text content.</div>
```

```
var element = document.getElementById("myDiv");
var text = element.innerText;
console.log(text); // Output: "This is some text content."
```

innerHTML:

- **innerHTML** is a property that represents the HTML content of an element.
- It retrieves or sets the HTML markup within the element, including any nested elements and tags.

- It can be used to dynamically change the structure and content of an element.

```
var element = document.getElementById("myDiv");
var html = element.innerHTML;
console.log(html); // Output: "This is <span>some</span> text content."
```

How to modify existing content

```
// Select the element by its ID
var paragraph = document.getElementById("myParagraph");

// Update the text content using innerText
paragraph.innerText = "Updated text!";
```

How to create element and how to append element in dom

```
// Create a new paragraph element
var newParagraph= document.createElement("p");

// Set innertext or other properties if needed
newParagraph.innerText = "This is a dynamically created paragraph.";

// Append the paragraph to the document body
document.body.appendChild(newParagraph);
```

- A new paragraph element is created using **document.createElement("p")**.
- The **innerText** property of the newly created paragraph element is set to "This is a dynamically created paragraph."
- The paragraph element is appended to the document body using **document.body.appendChild(newParagraph)**.

Appendchild and Append

Append and appendChild methods are used in JavaScript to add nodes to the DOM, but they have some differences in terms of usage, accepted parameters, and behavior:

appendChild

syntax :

parentNode.appendChild(newChild);

Parameters:

newChild: A single node (an element, text node, or any other node) that will be appended as the last child of parentNode

Behavior:

If the newChild is already in the DOM, it will be removed from its current position and moved to the new position.

Only accepts a single node.

Append

syntax :

parentNode.append(node1, node2, node3);

Parameters:

nodes: One or more nodes or strings that will be appended as the last children of parentNode.

Behavior:

Can append multiple nodes and/or strings at once.

If a string is provided, it will be added as a text node.

Allows appending a combination of nodes and text.

How to create textNode

```
// Step 1: Access the element where you want to append the text node
var myDiv = document.getElementById("myDiv");

// Step 2: Create a text node
var textNode = document.createTextNode("This is a dynamically created text node.");

// Step 3: Append the text node to the element
myDiv.appendChild(textNode);
```

1. Access the element where you want to append the text node.
2. Create a text node using **document.createTextNode()**.
3. Append the text node to the desired element.

How to apply styles using dom

```
// Step 1: Access the element where you want to append the text node
var myDiv = document.getElementById("myDiv");
```

```
// Step 2: Apply styles
myDiv.style.backgroundColor="red";
```

apply styles using
document.getElementById(“myDIv”).style.backgroundColor=”red”;

How to change attribute values by using setAttribute

```
<div id="myDiv">This is some text and have id myDiv but it will
changed to demo</div>
```

```
var a=document.getElementById("myDiv").setAttribute("id","demo");

console.log(document)//can inspect and check weather it was changed
or not
```

we can change the attribute by using .setAttribute.(“attribute name”,”attribute value”)

//output

```
▼ <body>
    <div id="demo">This is some text and
    have id myDiv but it will changed to
    demo</div>
```

How to get attribute

We can get the element attribute by using get attribute method in dom

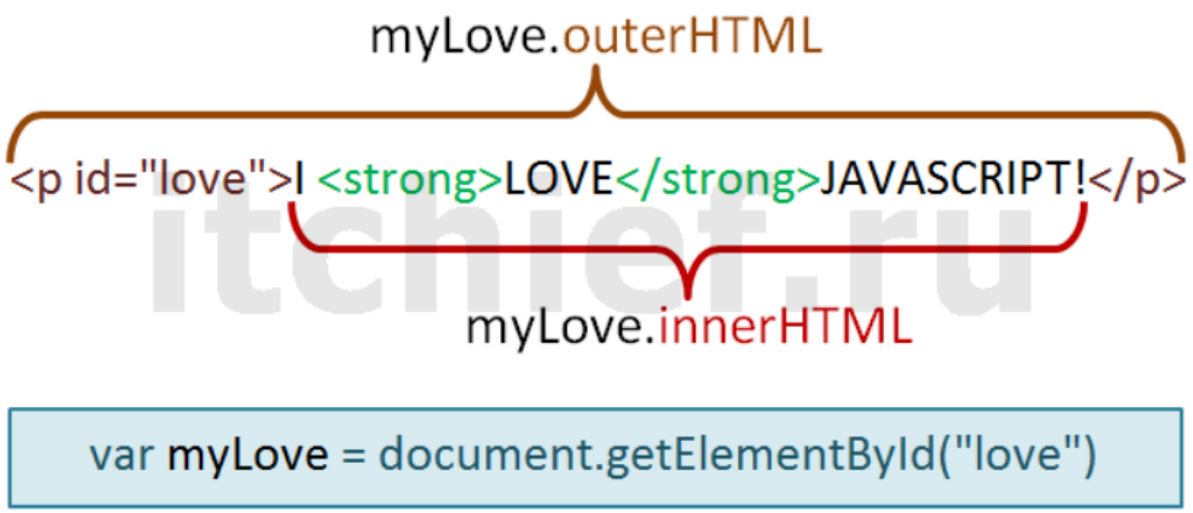
```
<img  id="myElement"
src="https://www.w3schools.com/myl-green-off.png" alt="lkdj">
```

```
 var element = document.getElementById("myElement");

// To get the value of an attribute, such as "src" for an image element:
var srcValue = element.getAttribute("src");
console.log(srcValue);
```

//output

```
                                        index.html:77
https://www.w3schools.com/myl-green-off.png
```



==========

Reading the elements

```
// getElementById()
const elementById = document.getElementById('myElementId');

// getElementsByClassName()
const elementsByClass = document.getElementsByClassName('myClassName');

// getElementsByTagName()
const elementsByTag = document.getElementsByTagName('div');

// querySelector()
const elementByQuery = document.querySelector('.myClass');

// querySelectorAll()
const elementsByQueryAll = document.querySelectorAll('p');
```

Create an element

```
// Function to add a new task
function addTask(taskText) {
  const taskList = document.getElementById('taskList');
  const newTask = document.createElement('li');
  newTask.textContent = taskText;
  taskList.appendChild(newTask);
}
```

Update the element

```javascript
// Function to update a task
function updateTask(oldTaskText, newTaskText) {
  const taskList = document.getElementById('taskList');
  const taskItems = taskList.getElementsByTagName('li');

  for (let i = 0; i < taskItems.length; i++) {
    if (taskItems[i].textContent === oldTaskText) {
      const newTask = document.createElement('li');
      newTask.textContent = newTaskText;

      // Replace the old task with the new one
      taskList.replaceChild(newTask, taskItems[i]);
      break;
    }
  }
}
```

```javascript
// Function to update a task
function updateTask(oldTaskText, newTaskText) {
  const taskList = document.getElementById('taskList');
  const taskItems = taskList.getElementsByTagName('li');

  for (let i = 0; i < taskItems.length; i++) {
    if (taskItems[i].textContent === oldTaskText) {
      taskItems[i].textContent = newTaskText;
      break;
    }
  }
}
```

Delete the element

```
// Function to delete a task
function deleteTask(taskText) {
  const taskList = document.getElementById('taskList');
  const taskItems = taskList.getElementsByTagName('li');

  for (let i = 0; i < taskItems.length; i++) {
    if (taskItems[i].textContent === taskText) {
      taskItems[i].remove();
      break;
    }
  }
}
```

Manipulating DOM Elements:

- Accessing Properties: Changing properties like `innerHTML`, `textContent`, `innerText`, `value`, etc.

- Modifying Attributes: Changing attributes like `src`, `href`, `class`, `id`, etc.

- Adding and Removing Elements: Creating new elements with `createElement()`, appending with `appendChild()`, removing with `removeChild()`, etc.

- Modifying Styles: Changing CSS styles using `style` property or `classList`.

```
// Accessing Properties
elementById.innerHTML = 'New content';

// Modifying Attributes
elementById.setAttribute('src', 'newImage.jpg');

// Adding and Removing Elements
const newElement = document.createElement('div');
newElement.textContent = 'New Element';
document.body.appendChild(newElement);

// Modifying Styles
elementById.style.color = 'blue';
elementById.classList.add('highlight');
```

Event Handling:

- addEventListener(): Attaches an event handler to an element.

- Event Types: Various events like `click`, `mouseover`, `keydown`, etc.

- Event Object: Provides information about the event and its target element.

- Event Propagation: Understanding event capturing and bubbling phases.

```
// addEventListener()
elementById.addEventListener('click', () => {
  console.log('Element clicked!');
});

// Event Object
elementById.addEventListener('mouseover', (event) => {
  console.log('Mouse over element:', event.target);
});
```

DOM Traversal and Manipulation:

- Parent, Child, and Sibling Nodes: Accessing and navigating through different parts of the DOM tree.

- Traversal Methods: `parentNode`, `childNodes`, `firstChild`, `lastChild`, `nextSibling`, `previousSibling`, etc.

- Walking the DOM: Techniques to iterate over DOM elements.

```
// Parent, Child, and Sibling Nodes
const parentElement = elementById.parentNode;
const firstChild = parentElement.firstChild;
const nextSibling = firstChild.nextSibling;

// Walking the DOM
let currentNode = elementById;
while (currentNode) {
  console.log(currentNode);
  currentNode = currentNode.nextSibling;
}
```