**BOM** – Browser Object Model

It represents additional objects provided by the browser (host env) for working with everything except the document. The functions alert/confirm/prompt are also part of the BOM. (this works only in browser)

Window object represents browser window and provides method to control it. It is a global object.

**DOM**

The Document Object Model (DOM) is a tree-like structure that represents an HTML document as JavaScript objects. It allows JavaScript to interact with and manipulate web pages dynamically.

Your whole HTML page becomes an object called document. (Apke pure page ko ek javascript object bana diya hai and usko document naam diya hai)

The DOM represents:

1. **HTML elements** as objects (<div>, <p>, <h1>, etc.).
2. **Nested structure** like a tree.
3. **JavaScript can modify the content, structure, and styles** of the webpage.

It contains properties like body, title that we can access using dot notation



Try – console.log(document), console.log(document.body)

1- Rewrite the whole body document

2- Will change the title in the browser

DOM is a programming interface for HTML and XML documents.

When the browser tries to render an HTML document, it creates an object based on the HTML document called DOM. Using this DOM, we can manipulate or change various elements inside the HTML document.

DOM tree refers to the HTML page where all the nodes are object.

Can be many types of nodes.

* **Text nodes**
* **Element nodes**
* **Comment nodes**

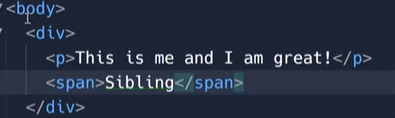
In a HTML page, <html> is the root and <head> and <body> are the children. A text node is always a leaf of the tree.

Children of an element –

Direct as well as deeply nested elements of an element are called its children.

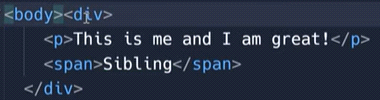
* Child nodes – element that are direct children. For ex, head and body are children of <html>.
* Descendant nodes – all nested elements, children, their children, etc.

How to access first child, last child, and child nodes?



In console, type - console.log(document.body.firstChild)

o/p – nothing (text node)



o/p – div. previously o/p was not coming div bcoz when we use first child, text nodes are also counted.

Instead of doing this, we can use (document.body.firstElementChild) – this will give element only.

Means if we want child of the body, we will get div no matter how many space we give or write any comments (no need of writing together)

**Selecting Elements in the DOM**

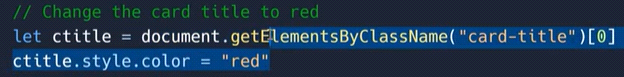
To manipulate the DOM, you first need to **select** elements. Here are the main methods:

(Method - function saved inside an object)

* **getElementByClassName** – this method returns elements that have the given CSS value (have to define index value, as multiple elements can have same class)

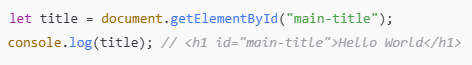
Selects elements by class name (returns a collection)



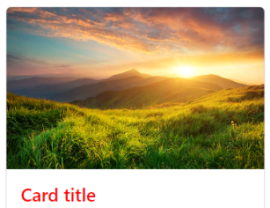


* **getElementById** – this method is used to get the element with a given “id” attribute

Selects an element by its id







The example below "finds" an HTML element (with id="demo"), and changes the element content (innerHTML) to "Hello JavaScript"

* JavaScript Can Change HTML Content

<button onclick="document.getElementById('demo').innerHTML='Hello JavaScript'">Click Here</button>

<p id="demo"></p>

<button id="btn" onclick="document.getElementById('btn').innerHTML=Date()">Click Me to Display Date</button>

* JavaScript Can Change HTML Attribute Values

<button onclick="document.getElementById('bulb').src='pic\_bulbon.gif'">Turn On the bulb</button>

<img id="bulb" src="pic\_bulboff.gif" alt="bulb">

<button onclick="document.getElementById('bulb').src='pic\_bulboff.gif'">Turn Off the bulb</button>

* JavaScript Can Change HTML Styles (CSS)

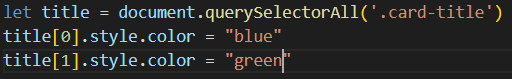
<p id="demo">JavaScript can change the style of an HTML element.</p>

<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Click Me!</button>



* **querySelectorAll** – returns all elements inside an element matching the given CSS selector

Selects **all** matching elements (returns a NodeList)





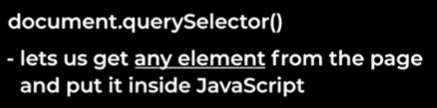
If we are targeting class attribute, then add (.), if using id attribute add (#)

* **querySelector** – returns the first element for the given CSS selector. A efficient version of -element.querySelectorAll(css)[0]

Selects the **first** element matching a CSS selector







**console.log(document.querySelector('button'));** // this string tells js which html element to get queryselector. this will get the first button element from the page and put it inside a js



**document.querySelector('button');** //Using querySelector, we can get any element from the age and put it inside js **document.querySelector('button').innerHTML = Changed’;** // and we can control the html inside that element.

Suppose if we have two button and want to access 2nd then we can add class to the second button and can call by the class attribute.

console.log(document.querySelector('.js-button')); // This will select an element with class js-button

const jsElem = document.querySelector('.js-button'); // we can also create variable

console.log(jsElem);

* **getElementByTagName** – returns elements with the given tag name

Selects elements by tag name (returns a collection)



**Modifying the DOM**

Once you select an element, you can change its content, style, attributes, and more.

* 1. **Change HTML Content** (.innerHTML & .textContent)

****

**innerHTML** – this property allows to get the HTML inside the element as a string. It is valid only for element nodes (not for text/comment node). For other node types we can use nodeValue or data.

**outerHTML** – this property contains the full HTML, innerHTML + the element itself.

Ex - <span id=”first”><b>HI</b>I am span</span>



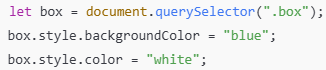
**Text Content** – provide access to the text inside the element: only text, minus all tags.

console.log(document.body.textContent)

**hidden property** – hidden attribute and the DOM specifies whether the element is visible or not.

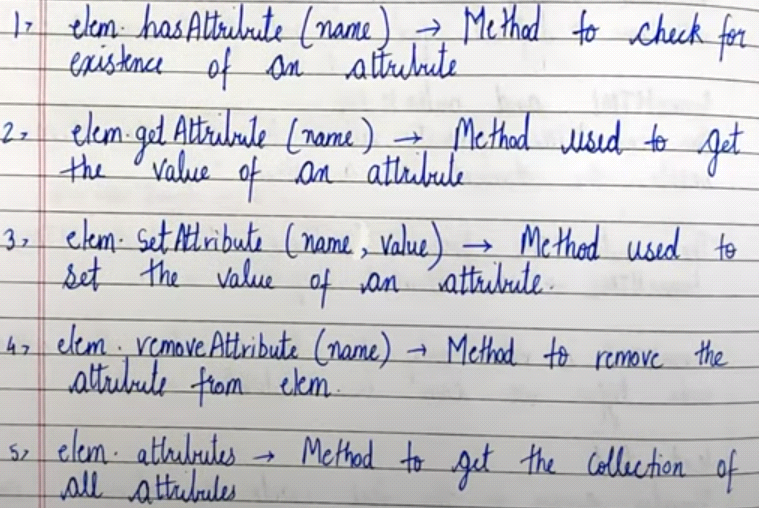
Ex- <span id=”first” hidden><b>HI</b>I am span</span> -- this will hide the whole element part

* 1. **Change CSS Styles** (.style)

****

* 1. **Change Attributes** (setAttribute() & getAttribute())

****



Example - <div id=”first” class=”Hello> Hey, I am first container</div>

Index.js – let first = document.getElementById(‘first’);

* console.log(first.hasAttribute(‘class’) – true

console.log(first.hasAttribute(‘style’) – false, as there is not attribute as style.

* let a = first.getAttribute(‘class’); console.log(a);

o/p – Hey, I am first container

* console.log(first.setAttribute9’class’, ‘class1 class2’);

o/p - now there will be two classes as class1 and class2

* console.log(first.removeAttribute(‘class’);

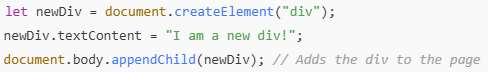
o/p – this will remove class – Hello

* console.log(first.attributes) – will print all attributes like class, id,
  1. **Add or Remove CSS Classes** (classList)

****

**Adding & Removing Elements**

* Create a New Element (createElement(), appendChild())

****

* Remove an Element (remove())

****

Let div = document.createElement(‘div’) // create

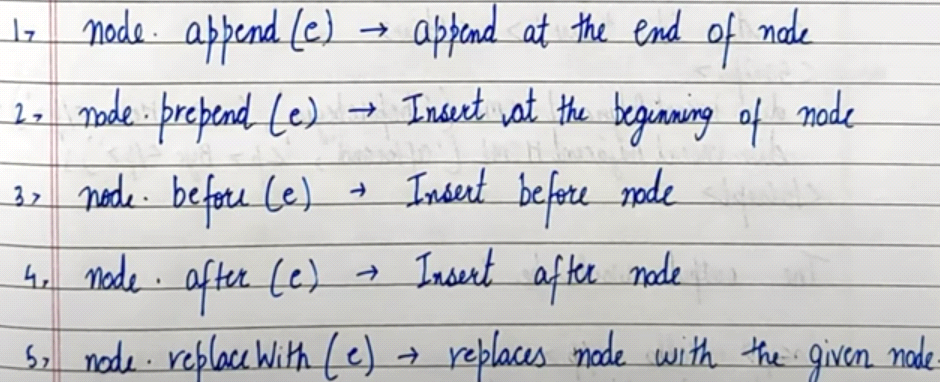
div.className = ‘alert’ //set class

div.innerHTML = ‘<span>Hello</span>’

document.body.append(div)

When there are small changes to be done, we can use innerHTML, if there are many changes, then we can use this method. For ex, we can use in while loop to create 10 elements

**Methods** -



**Traversing the DOM** (Parent, Child, Sibling)

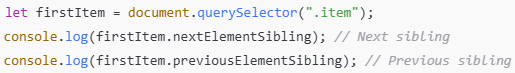
* Access Parent Element (parentElement)



* Access Child Elements (children)

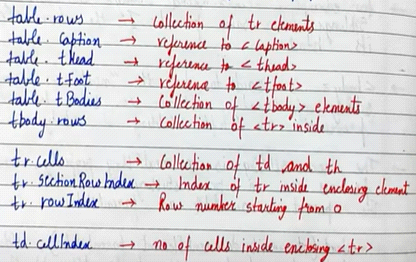


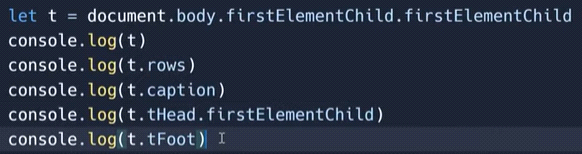
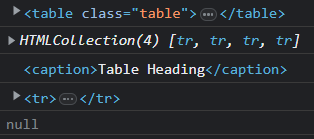
* Access Siblings (nextElementSibling, previousElementSibling)



**Table Navigation**

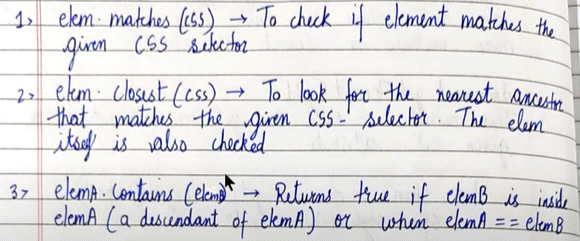
Table element supports following properties.

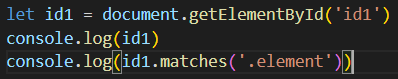


1st o/p – as the first element is table inside div container. 2nd o/p – will print tr elelemnts 3rd o/p – will print caption. 4th o/p – will print first element child of thead element. 5th o/p – as there is no tfoot defined, will print null.

**Matches, Closest and Contains Methods**







**URL Parameters**

URL parameters are key-value pairs added to a URL after a? (question mark). They are used to send data to the server via a GET request.



**query=javascript** → Key: query, Value: javascript

**page=2** → Key: page, Value: 2

Parameters are separated by **& (ampersand)**.

**Why Use URL Parameters?**

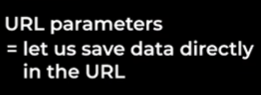
* Pass data to a server without a request body
* Filter or sort data (e.g., ?sort=price)
* Track user activity (e.g., ?utm\_source=google)
* Maintain state in web applications

URL parameters also known as search parameters

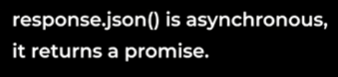


**URL Parameters dynamically**









**Modules**

**A module in JavaScript is a reusable piece of code that is imported and exported between different files.**

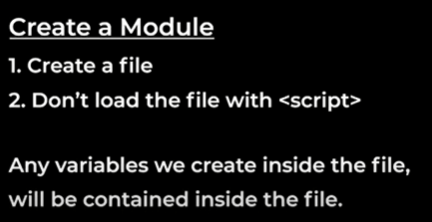
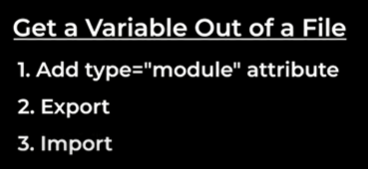
JavaScript modules allow you to break up your code into separate files. This makes it easier to maintain a code-base.

Modules are imported from external files with the import statement.

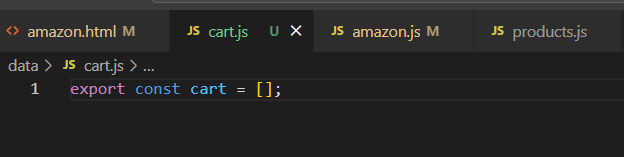
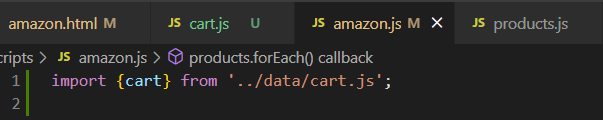
Modules also r**ely on type="module"** in the <script> tag.

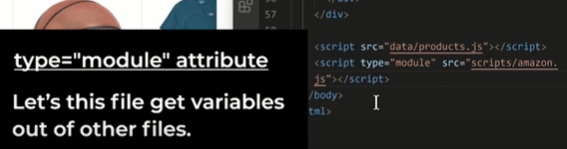
Why Module?

**Code organization** – Keeps code structured and manageable.  
**Re-usability** – Write code once, use it in multiple places.  
**Encapsulation** – Prevents variables from polluting the global scope.  
**Maintainability** – Easier to debug and modify code.

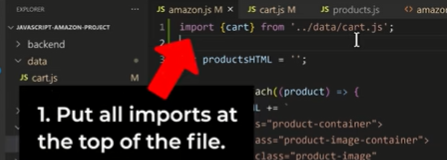
* Add type=’module’ attribute to enable this feature
* We export the variable that we want to get out
* Import the variables where we need them



**2 Important points:**

1. Put all imports at the top of the file.
2. In order to modules to work, we need to use live server. Modules won’t work when we go to particular folder and open directly using Google chrome. We need to open code editor and open using live server.



**Benefits of Modules:**

1. Helps us avoid naming conflicts
2. Don’t have to worry about order of the files
3. Modules are better way to organize our codes for bigger projects

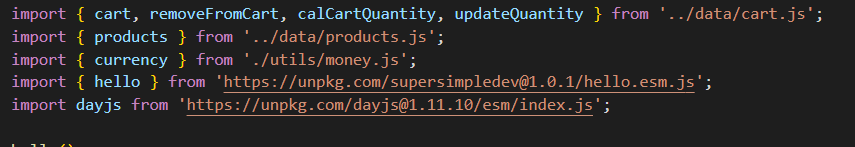
**Type of Export:**

1. Named Export – {name}
2. Default Export - name



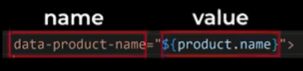
Another way of exporting something from the file.

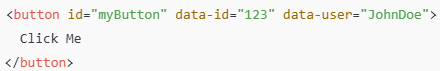
We can use it when we only want to export 1 thing from a file and makes a syntax little bit cleaner. Each file can only have 1 default export



**Data Attribute**

**Data attributes** allow you to store **extra information** directly inside an HTML element. These attributes **do not affect the appearance** but can be accessed via **JavaScript**



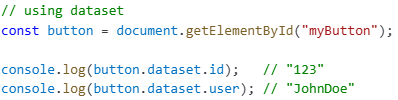
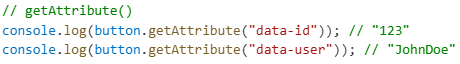
data-id="123" → Custom attribute holding the **ID.**

data-user="JohnDoe" → Custom attribute holding the **username.**

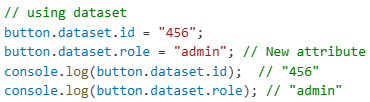
**You can access data attributes in JS using**:

1. dataset property

1. getAttribute() method

**Modify data attributes:**

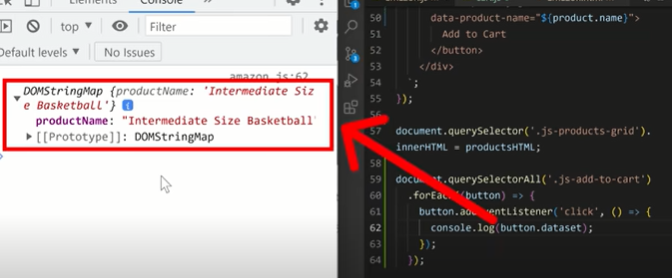
 

**Remove data attributes:**



**Note: Useful for dynamic actions and storing settings**

Dataset property gives us all the data attributes that are attached to the buttons



Product name converted from kebab case to camel case

