BOM – Browser Object Model

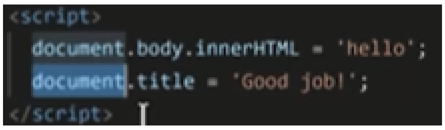
It represents additional objects provided by the browser (host env) for working with everything except the document.
The functions alert/confirm/prompt are also part of the BOM. (this works only in browser)
Window object represents browser window and provides method to control it. It is a global object.

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



```
<script>
  document.body.innerHTML = 'hello';
  document.title = 'Good job!';
</script>
```

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.

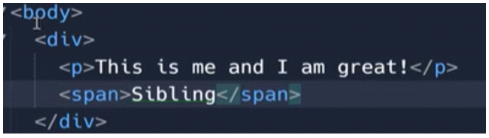
- **Text nodes**
- **Element nodes**
- **Comment nodes**

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.

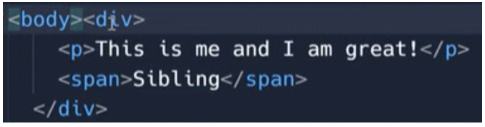
- Child nodes – element that are direct children. For ex, head and body are children of <html>.
- Descendant nodes – all nested elements, children, their children, etc.

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



```
<body>
  <div>
    <p>This is me and I am great!</p>
    <span>Sibling</span>
  </div>
```

In console, type - console.log(document.body.firstChild)
o/p – nothing (text node)



```
<body><div>
    <p>This is me and I am great!</p>
    <span>Sibling</span>
  </div>
```

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)

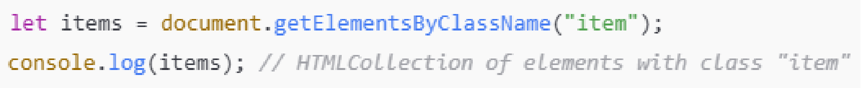
Selecting Elements in the DOM

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

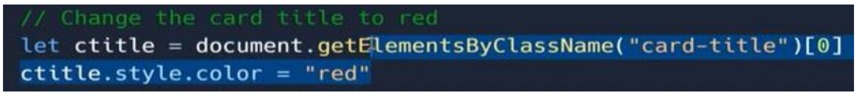
(Method - function saved inside an object)

- **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)



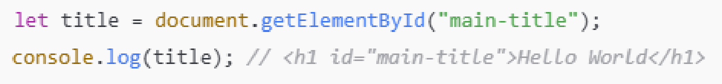
```
let items = document.getElementsByClassName("item");
console.log(items); // HTMLCollection of elements with class "item"
```



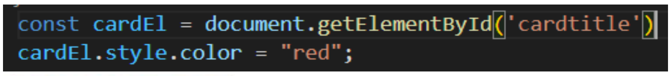
```
// Change the card title to red
let ctitle = document.getElementsByClassName("card-title")[0]
ctitle.style.color = "red"
```

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

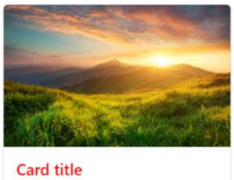
Selects an element by its id



```
let title = document.getElementById("main-title");
console.log(title); // <h1 id="main-title">Hello World</h1>
```



```
const cardEl = document.getElementById('cardtitle')
cardEl.style.color = "red";
```



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

- JavaScript Can Change HTML Content

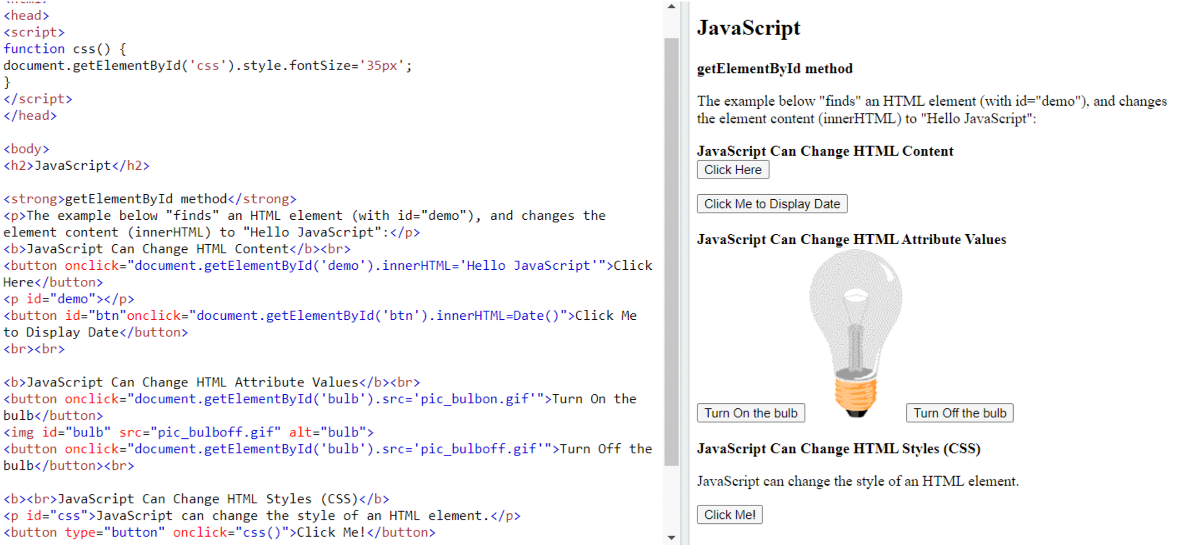
<button onclick="document.getElementById('demo').innerHTML='Hello JavaScript'">Click Here</button>
<p id="demo"></p>
<button id="btn" onclick="document.getElementById('btn').innerHTML=Date()">Click Me to Display Date</button>

- 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>



```
<head>
<script>
function css() {
document.getElementById('css').style.fontSize='35px';
}
</script>
</head>

<body>
<h2>JavaScript</h2>

<strong>getElementById method</strong>
<p>The example below "finds" an HTML element (with id="demo"), and changes the
element content (innerHTML) to "Hello JavaScript":</p>
<b>JavaScript Can Change HTML Content</b><br>
<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>
<br><br>

<b>JavaScript Can Change HTML Attribute Values</b><br>
<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><br>

<b><br>JavaScript Can Change HTML Styles (CSS)</b>
<p id="css">JavaScript can change the style of an HTML element.</p>
<button type="button" onclick="css()">Click Me!</button>
```

JavaScript

getElementById method

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

JavaScript Can Change HTML Content

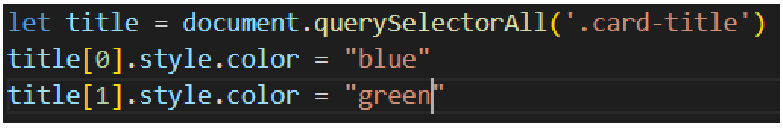
JavaScript Can Change HTML Attribute Values

JavaScript Can Change HTML Styles (CSS)

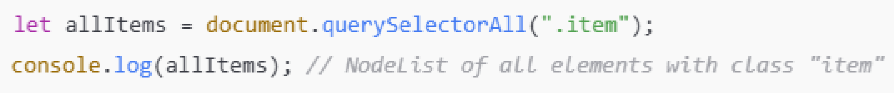
JavaScript can change the style of an HTML element.

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

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



```
let title = document.querySelectorAll('.card-title')
title[0].style.color = "blue"
title[1].style.color = "green"
```

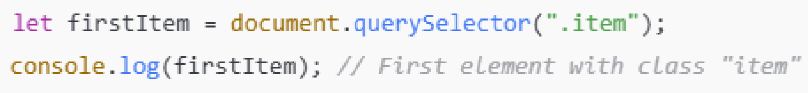


```
let allItems = document.querySelectorAll(".item");
console.log(allItems); // NodeList of all elements with class "item"
```

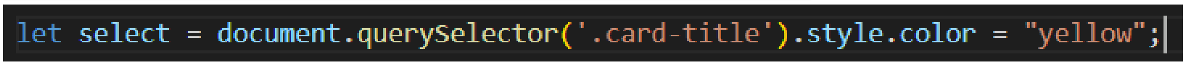
If we are targeting class attribute, then add (.), if using id attribute add (#)

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

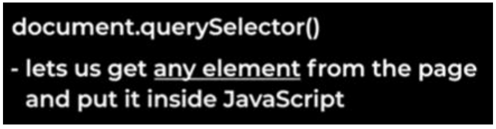
Selects the first element matching a CSS selector



```
let firstItem = document.querySelector(".item");
console.log(firstItem); // First element with class "item"
```



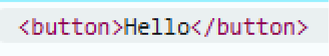
```
let select = document.querySelector('.card-title').style.color = "yellow";
```



document.querySelector()

- lets us get any element from the page and put it inside JavaScript

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



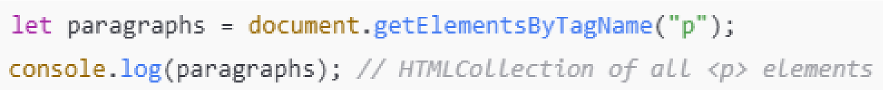
```
<button>Hello</button>
```

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)

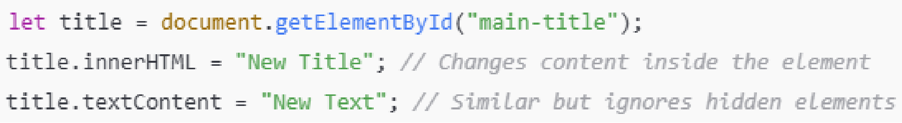


```
let paragraphs = document.getElementsByTagName("p");
console.log(paragraphs); // HTMLCollection of all <p> elements
```

Modifying the DOM

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

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

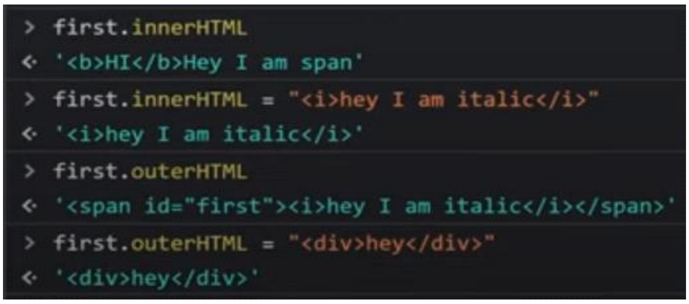


```
let title = document.getElementById("main-title");
title.innerHTML = "New Title"; // Changes content inside the element
title.textContent = "New Text"; // Similar but ignores hidden elements
```

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>

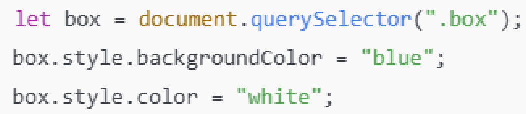


```
> first.innerHTML
< '<b>HI</b>Hey I am span'
> first.innerHTML = "<i>hey I am italic</i>"
< '<i>hey I am italic</i>'
> first.outerHTML
< '<span id="first"><i>hey I am italic</i></span>'
> first.outerHTML = "<div>hey</div>"
< '<div>hey</div>'
```

Text Content – provide access to the text inside the element: only text, minus all tags.
console.log(document.body.textContent)

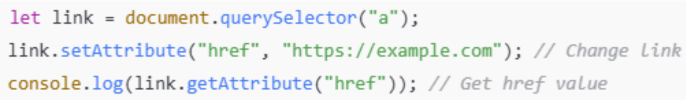
hidden property – hidden attribute and the DOM specifies whether the element is visible or not.
Ex- <span id=”first” hidden><b>HI</b>I am span</span> -- this will hide the whole element part

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

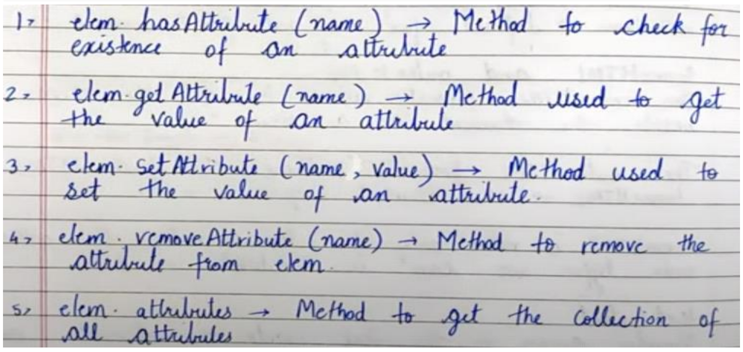


```
let box = document.querySelector(".box");
box.style.backgroundColor = "blue";
box.style.color = "white";
```

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



```
let link = document.querySelector("a");
link.setAttribute("href", "https://example.com"); // Change link
console.log(link.getAttribute("href")); // Get href value
```



1> elem.hasAttribute(name) → Method to check for existence of an attribute

2> elem.getAttribute(name) → Method used to get the value of an attribute

3> elem.setAttribute(name, value) → Method used to set the value of an attribute.

4> elem.removeAttribute(name) → Method to remove the attribute from elem.

5> elem.attributes → Method to get the collection of all attributes

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’);

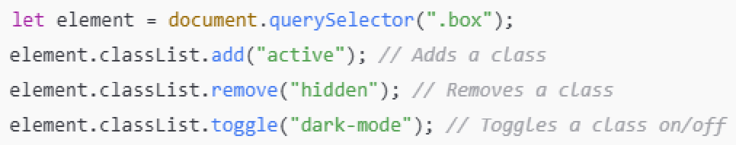
o/p - now there will be two classes as class1 and class2

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

o/p – this will remove class – Hello

- console.log(first.attributes) – will print all attributes like class, id,

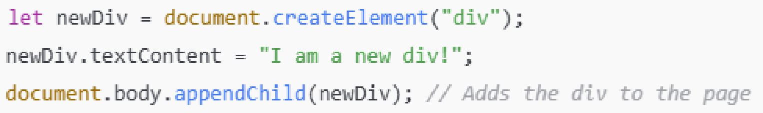
d. **Add or Remove CSS Classes** (classList)



```
let element = document.querySelector(".box");
element.classList.add("active"); // Adds a class
element.classList.remove("hidden"); // Removes a class
element.classList.toggle("dark-mode"); // Toggles a class on/off
```

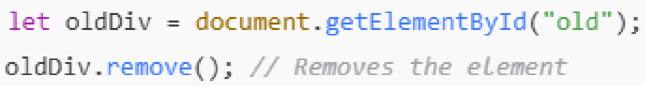
Adding & Removing Elements

- Create a New Element (`createElement()`, `appendChild()`)



```
let newDiv = document.createElement("div");
newDiv.textContent = "I am a new div!";
document.body.appendChild(newDiv); // Adds the div to the page
```

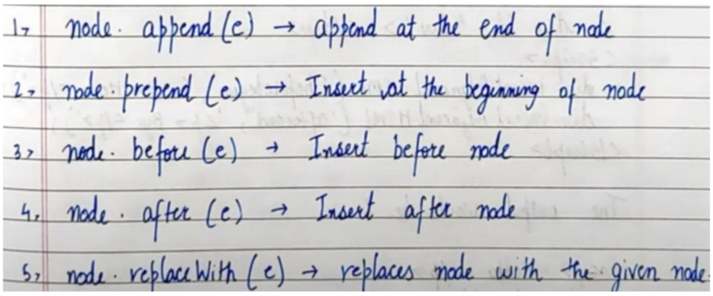
- Remove an Element (remove())



```
let oldDiv = document.getElementById("old");
oldDiv.remove(); // Removes the element
```

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

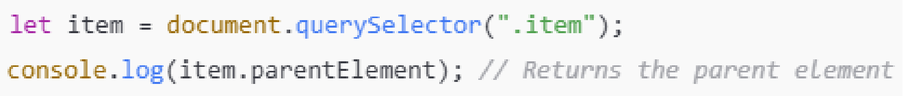
Methods -



1, node. append (e) → append at the end of node
2, node. prepend (e) → Insert at the beginning of node
3, node. before (e) → Insert before node
4, node. after (e) → Insert after node
5, node. replaceWith (e) → replaces node with the given node

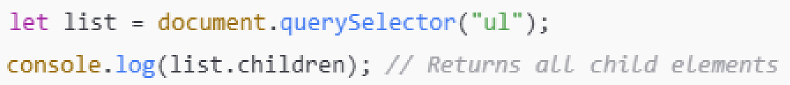
Traversing the DOM (Parent, Child, Sibling)

- Access Parent Element (parentElement)



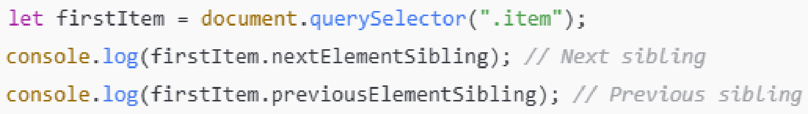
```
let item = document.querySelector(".item");
console.log(item.parentElement); // Returns the parent element
```

- Access Child Elements (children)



```
let list = document.querySelector("ul");
console.log(list.children); // Returns all child elements
```

- Access Siblings (nextElementSibling, previousElementSibling)



```
let firstItem = document.querySelector(".item");
console.log(firstItem.nextElementSibling); // Next sibling
console.log(firstItem.previousElementSibling); // Previous sibling
```

Table Navigation

Table element supports following properties.

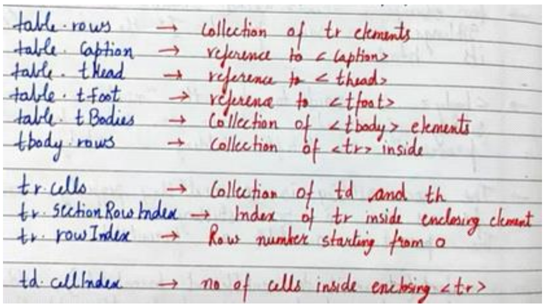
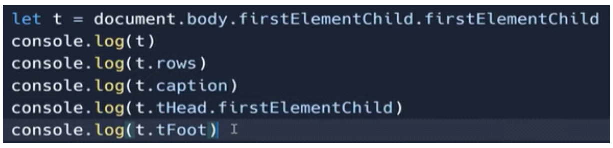


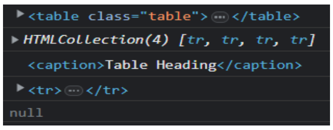
table. rows → Collection of tr elements
table. Caption → reference to <caption>
table. tHead → reference to <thead>
table. tFoot → reference to <tfoot>
table. tBodies → Collection of <tbody> elements
tbody. rows → Collection of <tr> inside

tr. cells → Collection of td and th
tr. sectionRowIndex → Index of tr inside enclosing element
tr. rowIndex → Row number starting from 0

td. cellIndex → no of cells inside enclosing <tr>



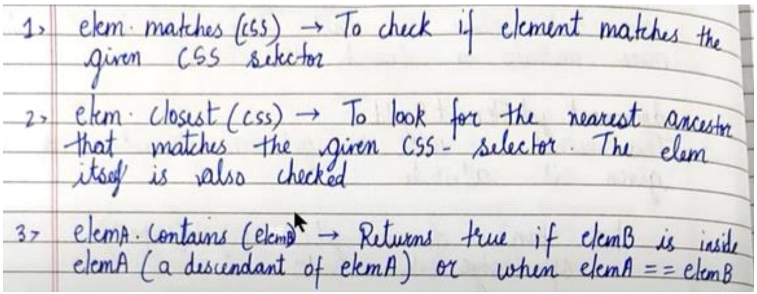
```
let t = document.body.firstElementChild.firstElementChild
console.log(t)
console.log(t.rows)
console.log(t.caption)
console.log(t.tHead.firstElementChild)
console.log(t.tFoot)
```



```
▸<table class="table">…</table>
▸HTMLCollection(4) [tr, tr, tr, tr]
  <caption>Table Heading</caption>
 ▸<tr>…</tr>
null
```

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.

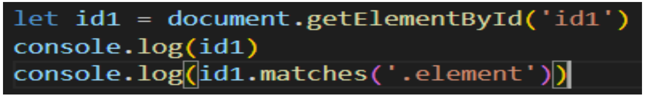
Matches, Closest and Contains Methods



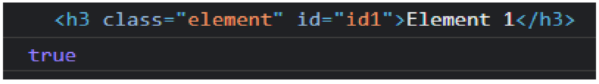
1> elem.matches (css) → To check if element matches the given CSS selector

2> elem.closest (css) → To look for the nearest ancestor that matches the given CSS-selector. The elem itself is also checked

3> elemA.contains (elemB) → Returns true if elemB is inside elemA (a descendant of elemA) or when elemA == elemB



```
let id1 = document.getElementById('id1')
console.log(id1)
console.log(id1.matches('.element'))
```



```
  <h3 class="element" id="id1">Element 1</h3>
 true
```

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.

```
https://example.com/search?query=javascript&page=2
```

query=javascript → Key: query, Value: javascript
page=2 → Key: page, Value: 2
Parameters are separated by **& (ampersand)**.

Why Use URL Parameters?

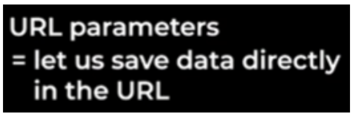
- Pass data to a server without a request body
- Filter or sort data (e.g., ?sort=price)
- Track user activity (e.g., ?utm_source=google)
- Maintain state in web applications

URL parameters also known as search parameters

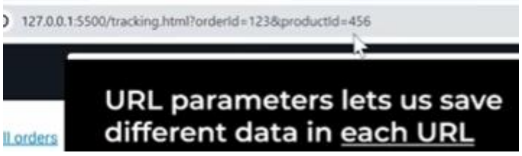
```
const params = new URLSearchParams(window.location.search);
console.log(params.get("query")); // Output: "javascript"
console.log(params.get("page"));  // Output: "2"
```

URL Parameters dynamically

```
const baseURL = "https://example.com/search";
const params = new URLSearchParams({
  query: "javascript",
  page: 2,
  sort: "latest"
});

console.log(`${baseURL}?${params.toString()}`);
// Output: https://example.com/search?query=javascript&page=2&sort=latest
```

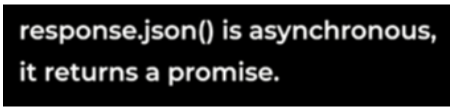


URL parameters
= let us save data directly
in the URL



127.0.0.1:5500/tracking.html?orderId=123&productId=456

URL parameters lets us save different data in each URL

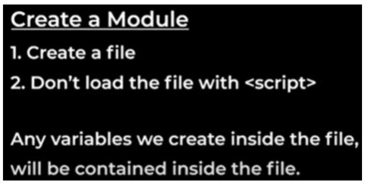


response.json() is asynchronous, it returns a promise.

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.

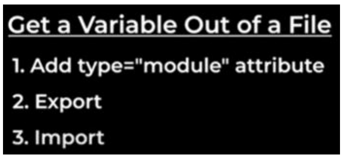
Why Module?
Code organization – Keeps code structured and manageable.
Re-usability – Write code once, use it in multiple places.
Encapsulation – Prevents variables from polluting the global scope.
Maintainability – Easier to debug and modify code.



Create a Module

1. Create a file
2. Don't load the file with <script>

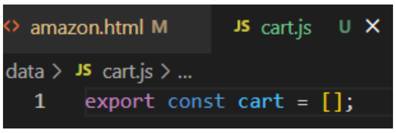
Any variables we create inside the file, will be contained inside the file.



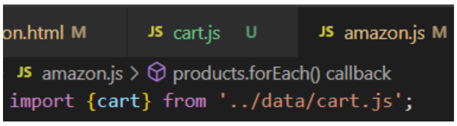
Get a Variable Out of a File

1. Add type="module" attribute
2. Export
3. Import

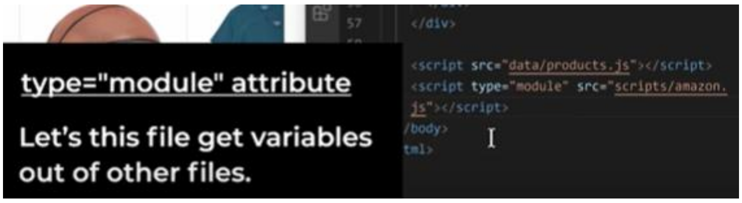
- Add type='module' attribute to enable this feature
- We export the variable that we want to get out
- Import the variables where we need them



```
export const cart = [];
```



```
import {cart} from '../data/cart.js';
```

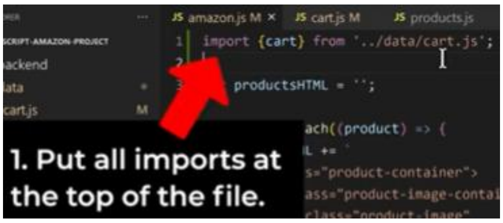


type="module" attribute

Let's this file get variables out of other files.

```
</div>

<script src="data/products.js"></script>
<script type="module" src="scripts/amazon.
js"></script>
/body>
tml>
```

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.



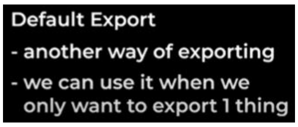
```
import {cart} from '../data/cart.js';

productsHTML = '';

ach((product) => {
L +=
s="product-container">
ass="product-image-contai
```

1. Put all imports at the top of the file.

Benefits of Modules:

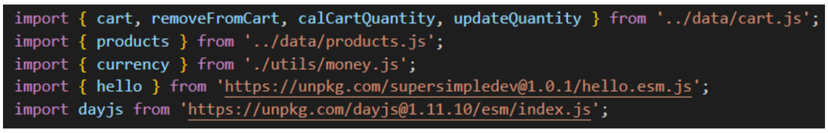
1. Helps us avoid naming conflicts
2. Don't have to worry about order of the files
3. Modules are better way to organize our codes for bigger projects

Type of Export:
1. Named Export – {name}
2. Default Export - name



Default Export
- another way of exporting
- we can use it when we only want to export 1 thing

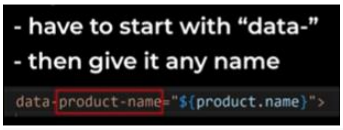
Another way of exporting something from the file.
We can use it when we only want to export 1 thing from a file and makes a syntax little bit cleaner. Each file can only have 1 default export



```
import { cart, removeFromCart, calCartQuantity, updateQuantity } from '../data/cart.js';
import { products } from '../data/products.js';
import { currency } from './utils/money.js';
import { hello } from 'https://unpkg.com/supersimpledev@1.0.1/hello.esm.js';
import dayjs from 'https://unpkg.com/dayjs@1.11.10/esm/index.js';
```

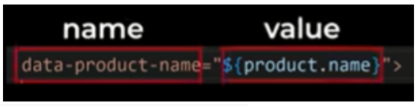
Data Attribute

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**

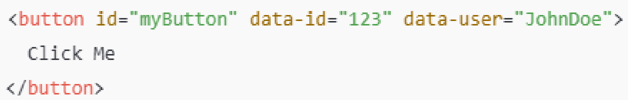


- have to start with "data-"
- then give it any name

```
data-product-name="${product.name}">
```



```
data-product-name="${product.name}">
```



```
<button id="myButton" data-id="123" data-user="JohnDoe">
  Click Me
</button>
```

`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

```
// using dataset
const button = document.getElementById("myButton");

console.log(button.dataset.id);   // "123"
console.log(button.dataset.user); // "JohnDoe"
```

```
// getAttribute()
console.log(button.getAttribute("data-id")); // "123"
console.log(button.getAttribute("data-user")); // "JohnDoe"
```

Modify data attributes:

```
// using dataset
button.dataset.id = "456";
button.dataset.role = "admin"; // New attribute
console.log(button.dataset.id);  // "456"
console.log(button.dataset.role); // "admin"
```

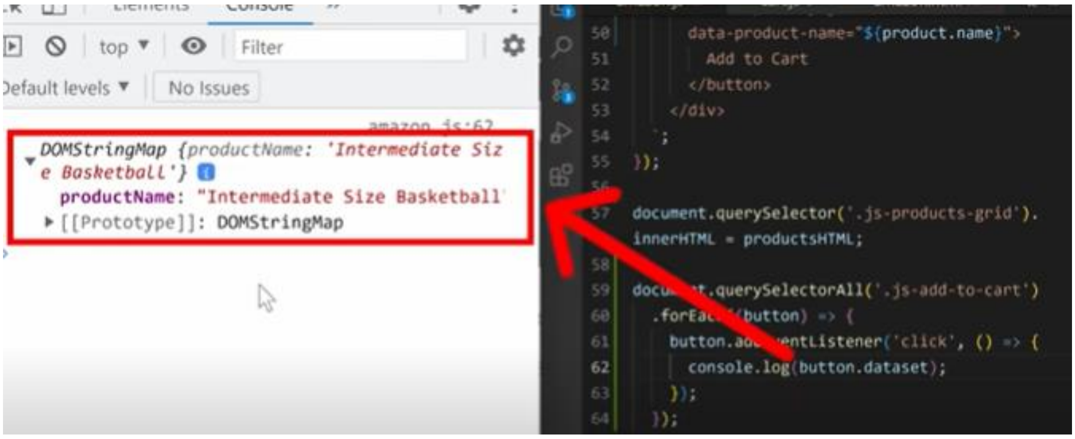
```
// setAttribute()
button.setAttribute("data-user", "JaneDoe");
console.log(button.dataset.user); // "JaneDoe"
```

Remove data attributes:

```
button.removeAttribute("data-id");
console.log(button.dataset.id); // undefined
```

Note: Useful for dynamic actions and storing settings

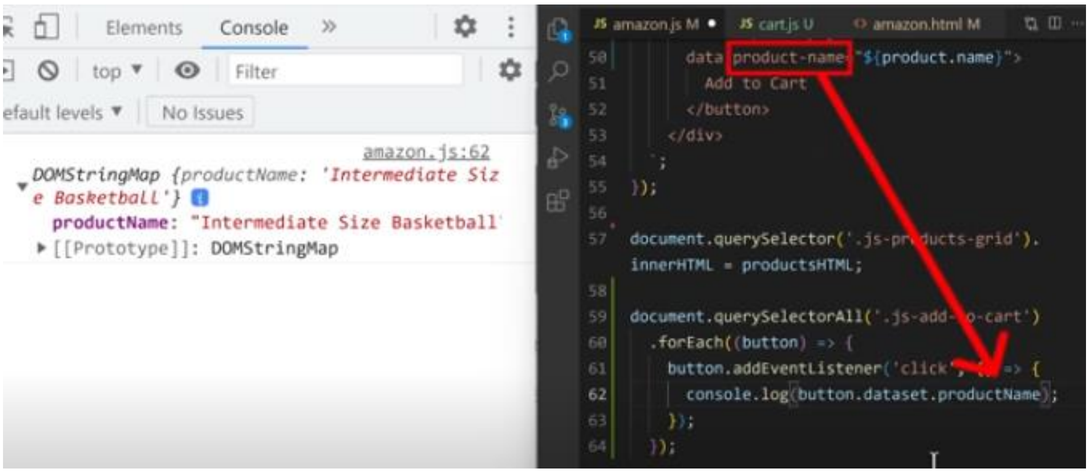
Dataset property gives us all the data attributes that are attached to the buttons



```
DOMStringMap {productName: 'Intermediate Size Basketball'}
  productName: "Intermediate Size Basketball"
  ▸ [[Prototype]]: DOMStringMap
```

```
          data-product-name="${product.name}">
          Add to Cart
        </button>
      </div>
    `;
  });

  document.querySelector('.js-products-grid').
  innerHTML = productsHTML;

  document.querySelectorAll('.js-add-to-cart')
    .forEach((button) => {
      button.addEventListener('click', () => {
        console.log(button.dataset);
      });
    });
```

Product name converted from kebab case to camel case



```
amazon.js:62
DOMStringMap {productName: 'Intermediate Size Basketball'}
  productName: "Intermediate Size Basketball"
▸ [[Prototype]]: DOMStringMap
```

```
          data-product-name="${product.name}">
            Add to Cart
          </button>
        </div>
      `;
  });

  document.querySelector('.js-products-grid').
  innerHTML = productsHTML;

  document.querySelectorAll('.js-add-to-cart')
    .forEach((button) => {
      button.addEventListener('click', () => {
        console.log(button.dataset.productName);
      });
    });
```