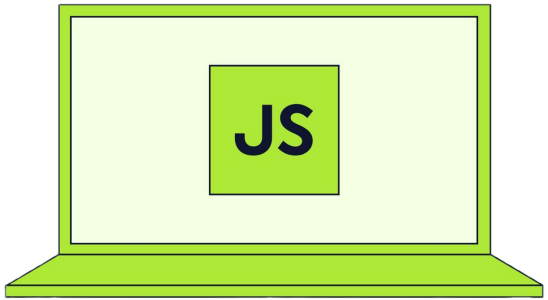
Newton School of Technology

The Complete Javascript Course



@newtonschool

Lecture 21: DOM Introduction & Selectors

- Nishchal Gupta

JS

What we have learnt so far

Newton School of Technology

- Array Basics
  - Declaration, indexing, accessing, updating
- Object Basics
  - Key-value pairs, accessing and modifying values

- Function Basics
  - Declaring a function (function greet() {})
  - Parameters and return values

    Arguments
  - Function invocation

Before the DOM – The Static Web

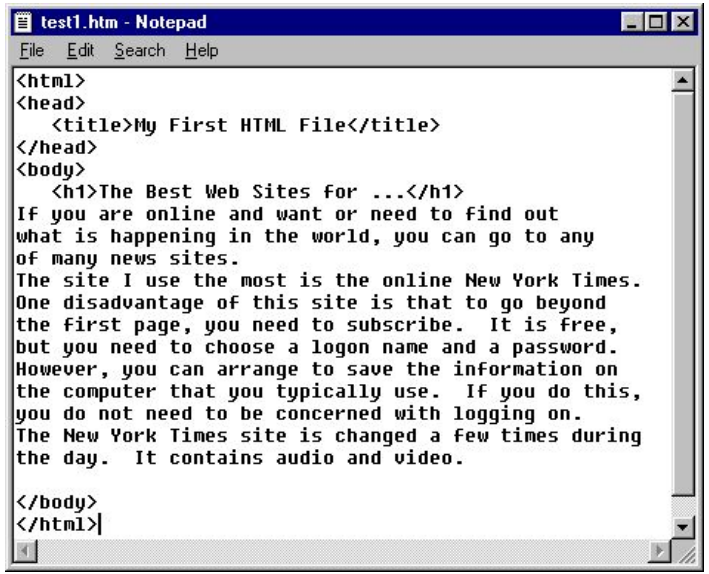
Newton School of Technology

In the early days (1990s), websites were just fixed documents written in HTML. They were like printed books — once published, you couldn't change them unless you rewrote the content.

Before the DOM – The Static Web

Newton School of Technology

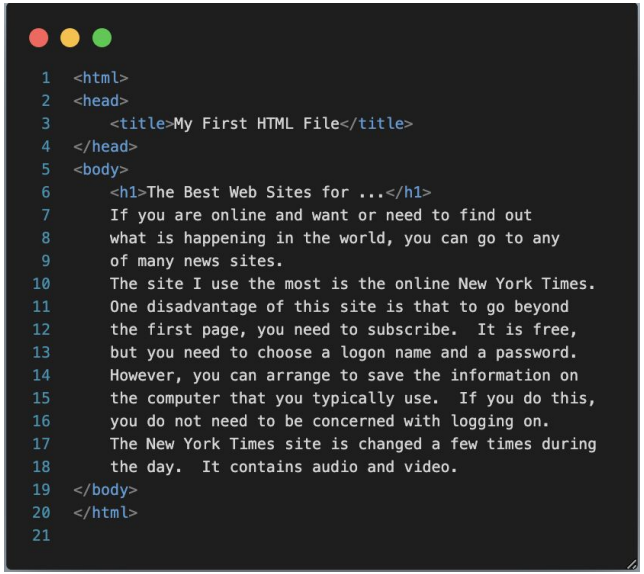
A web page loaded into your browser was the final version, not meant to change.



Before the DOM – The Static Web

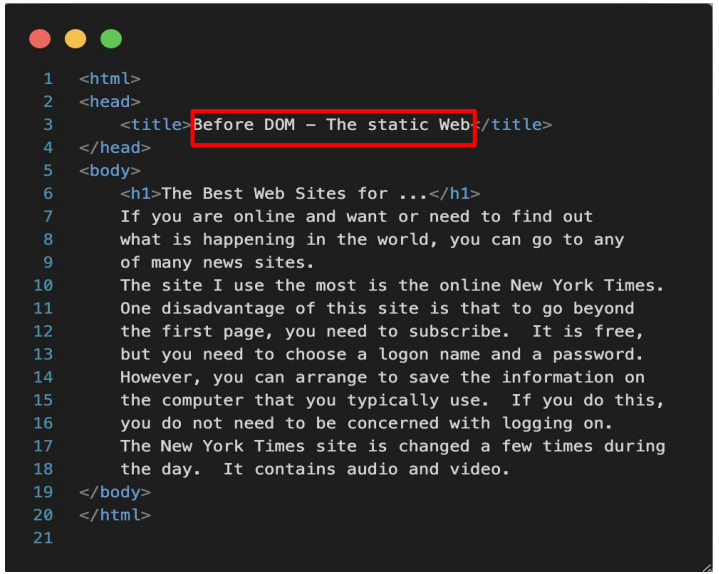
Newton School of Technology

If the owner wanted to update even one word, they had to:



```
<html>
<head>
    <title>My First HTML File</title>
</head>
<body>
    <h1>The Best Web Sites for ...</h1>
    If you are online and want or need to find out
    what is happening in the world, you can go to any
    of many news sites.
    The site I use the most is the online New York Times.
    One disadvantage of this site is that to go beyond
    the first page, you need to subscribe.  It is free,
    but you need to choose a logon name and a password.
    However, you can arrange to save the information on
    the computer that you typically use.  If you do this,
    you do not need to be concerned with logging on.
    The New York Times site is changed a few times during
    the day.  It contains audio and video.
</body>
</html>

```

Manually open the HTML file.



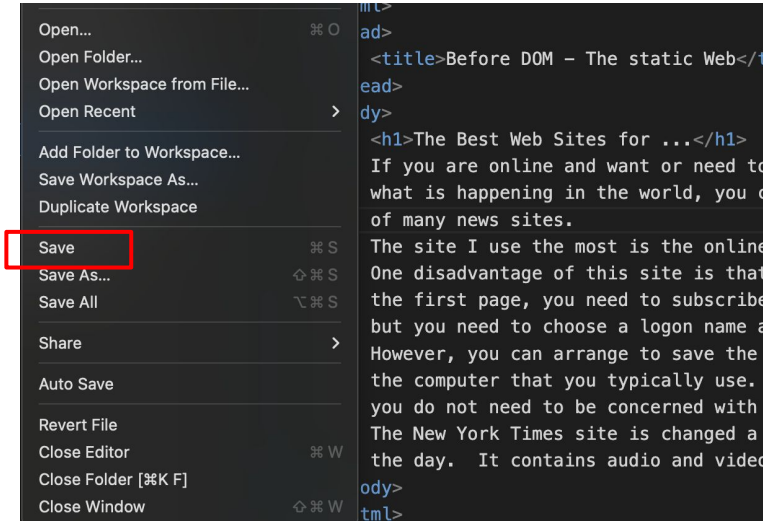
```
<html>
<head>
    <title>Before DOM – The static Web</title>
</head>
<body>
    <h1>The Best Web Sites for ...</h1>
    If you are online and want or need to find out
    what is happening in the world, you can go to any
    of many news sites.
    The site I use the most is the online New York Times.
    One disadvantage of this site is that to go beyond
    the first page, you need to subscribe.  It is free,
    but you need to choose a logon name and a password.
    However, you can arrange to save the information on
    the computer that you typically use.  If you do this,
    you do not need to be concerned with logging on.
    The New York Times site is changed a few times during
    the day.  It contains audio and video.
</body>
</html>

```

Edit the text in a code editor.

Before the DOM – The Static Web

Newton School of Technology

If the owner wanted to update even one word, they had to:



Save the file.

Upload it again to the server (re-deploy).

Ask users to refresh or revisit the site.

Before the DOM – The Static Web

Imagine a shop prints “**SALE 10% OFF!**” on a banner in a newspaper ad. Now they want to update it to “**SALE 20% OFF**”. But to do that, they’d have to **reprint the entire newspaper** — just to fix one number.

Before the DOM – The Static Web

Newton School of Technology

That's exactly what early websites felt like:

- Even small changes = big effort
- Websites were **hard-coded**
- No user interaction (no dropdowns, buttons, real-time updates)

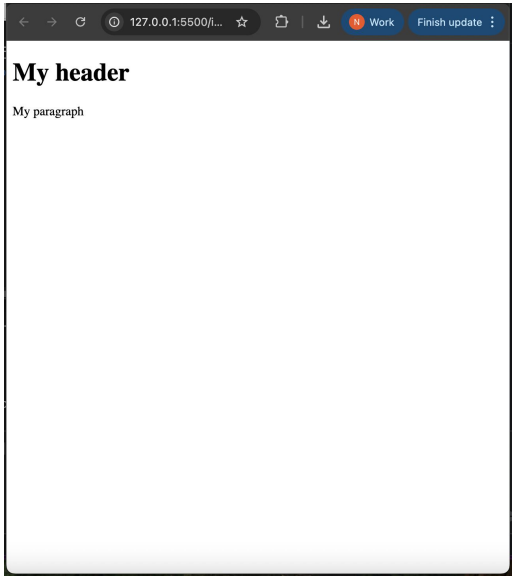
What changed: The DOM

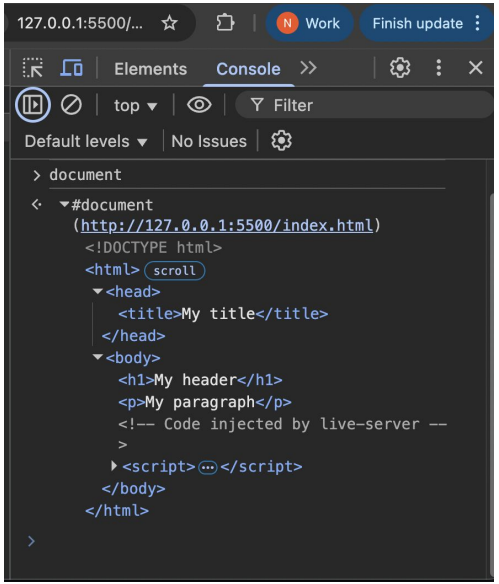
Newton School of Technology

DOM = Document Object Model

The DOM is like a live version of your webpage that sits in the browser's memory.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>My title</title>
5    </head>
6    <body>
7      <h1>My header</h1>
8      <p>My paragraph</p>
9    </body>
10 </html>
11
```

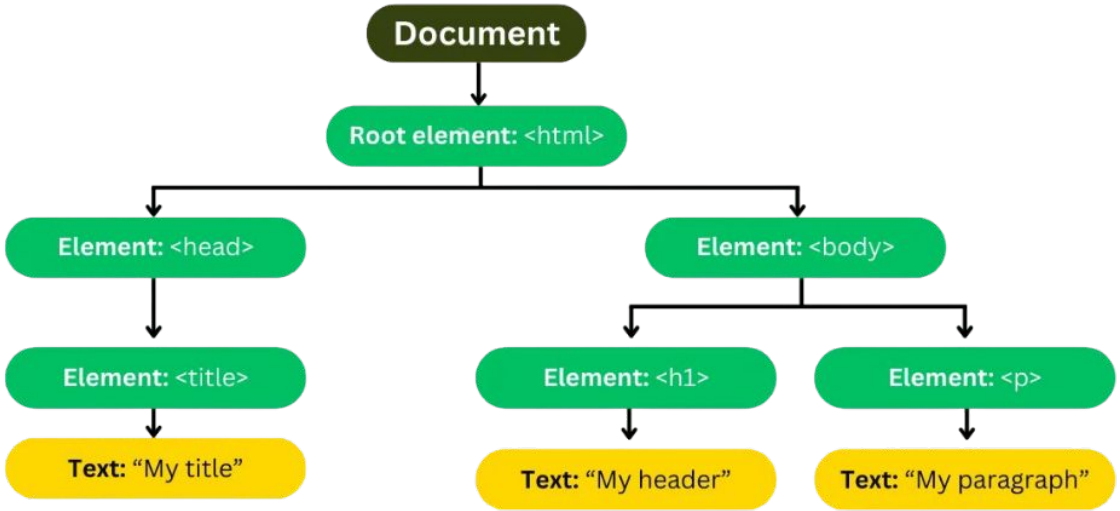




What changed: The DOM

Newton School of Technology

When a web-page loads:
The browser turns the HTML code into a **tree-like structure**
That structure is what JavaScript can talk to — **read from it** or **change it.**



What changed: The DOM

Newton School of Technology

It means you no longer have to touch the original file like before, when you had to edit the file, save it, and re-upload it to update the sale percentage.

```
<!DOCTYPE html>
<html>
<head>
  <title>Static Sale Banner</title>
</head>
<body>
  <h1>SALE 10% OFF</h1>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Static Sale Banner</title>
</head>
<body>
  <h1>SALE 20% OFF</h1>
</body>
</html>
```

What changed: The DOM

Newton School of Technology

With JavaScript and the DOM, **you don’t need to re-edit HTML manually** — just update it in memory while the page is running.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My First Page</title>
  </head>
  <body>
    <h1>Hello DOM</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

```
document.getElementById("sale").innerText = "SALE 20% OFF";
```

What changed: The DOM

Newton School of Technology

- Before the DOM, web pages were just **static** display boards — what you saw was **fixed** unless someone manually edited the HTML file, saved it, and re-uploaded it to the server.

- There was no way to change the content after the page had loaded.

- With the introduction of the **DOM** and **JavaScript**, web pages transformed into **interactive** dashboards.

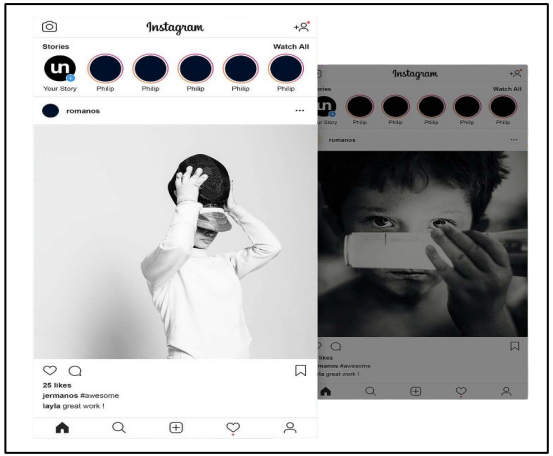
- Developers could now update headlines instantly, **without reloading the page**. Content could be shown or hidden **dynamically** based on user actions, and pages could be animated, styled, or even built on the fly — all while the user remained on the same page.

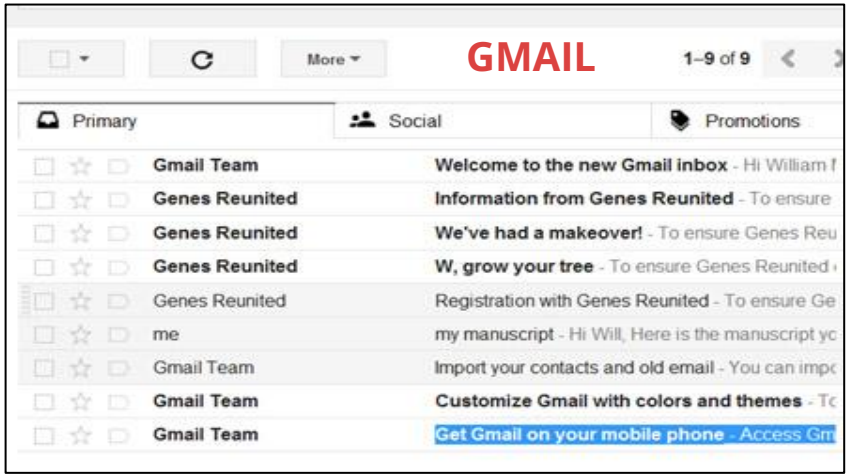
Modern Examples That Use the DOM

Newton School of Technology

New posts appear in your feed **without reloading** the app.That's because **JavaScript**
is updating the **DOM** in the background.

You receive a new email – and it appears instantly
No full page reload — just a small update in the DOM



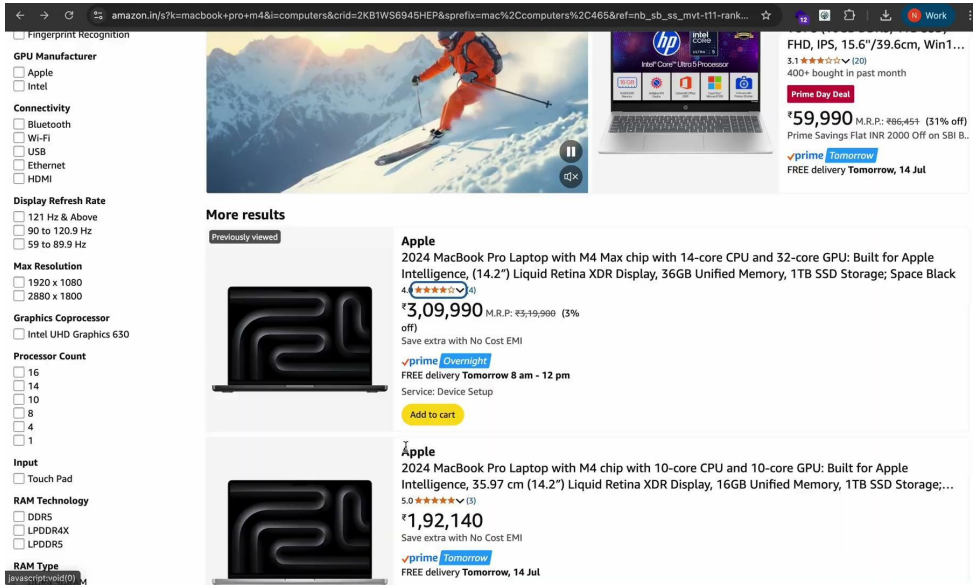


Modern Examples That Use the DOM

Newton School of Technology

Amazon:

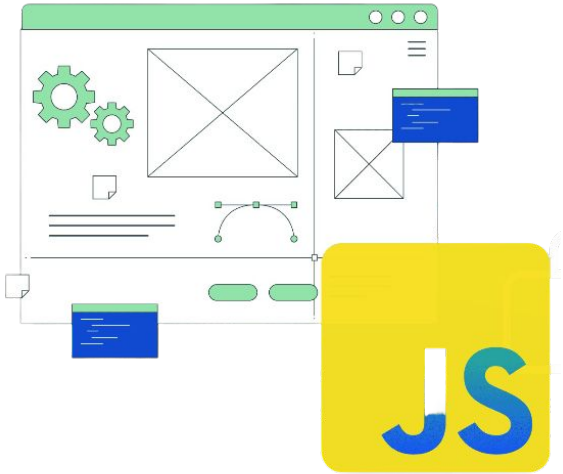
You add 3 items to cart — total price updates live.
Again, only the number and total update — the whole page doesn't reload.



Modern Examples That Use the DOM

Newton School of Technology

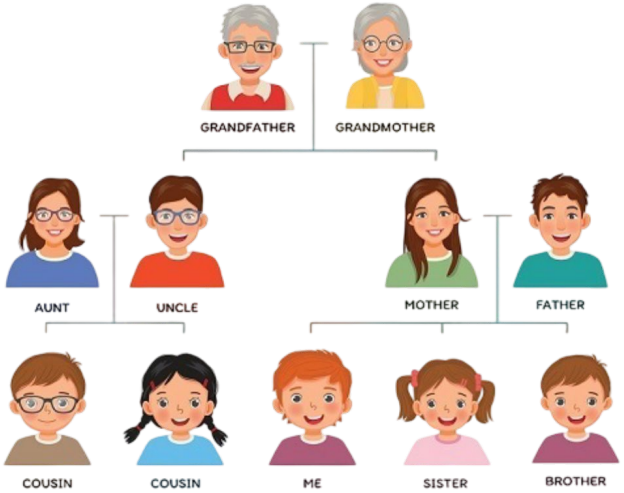
All of this happens through **DOM manipulation** in memory — JavaScript **updates** only the parts that need to change.

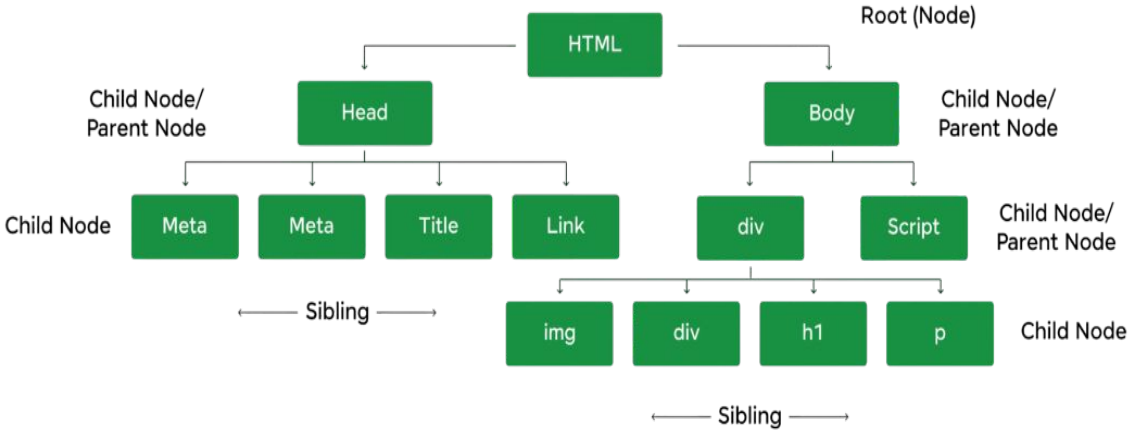


Understanding the DOM

Newton School of Technology

Think of the DOM like a family tree — each person represents an HTML element.
It starts from the **<html>** root (like grandparents), branches into **<head>** and **<body>** (parents), and further into children and siblings (like you, your siblings, and cousins).



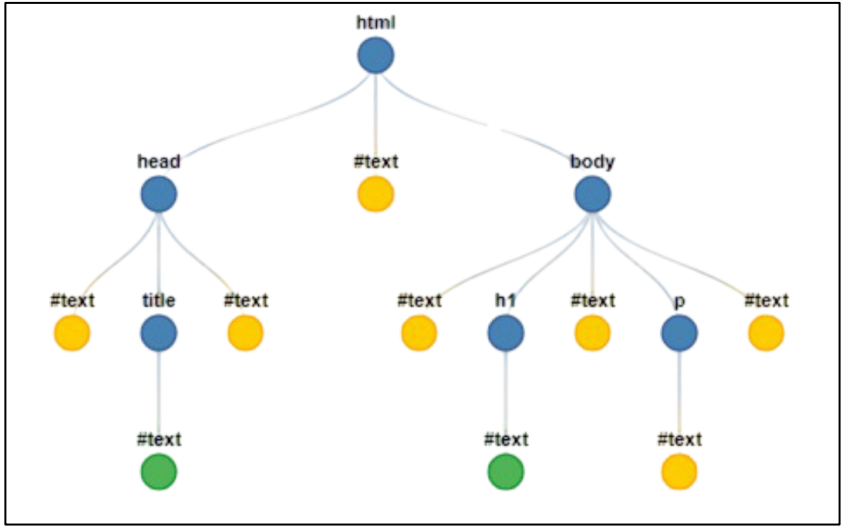


Understanding the DOM

Newton School of Technology

Just as each family member has a unique place and relation in the tree, **every HTML element has a defined position in the DOM hierarchy**, and JavaScript can **"walk" this tree**, **find a node**, and **update it** — like replacing the name of a cousin or moving someone to another branch.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>My Page</title>
5    </head>
6    <body>
7      <h1>Welcome</h1>
8      <p>This is a paragraph.</p>
9    </body>
10 </html>
```



Why the DOM Matters

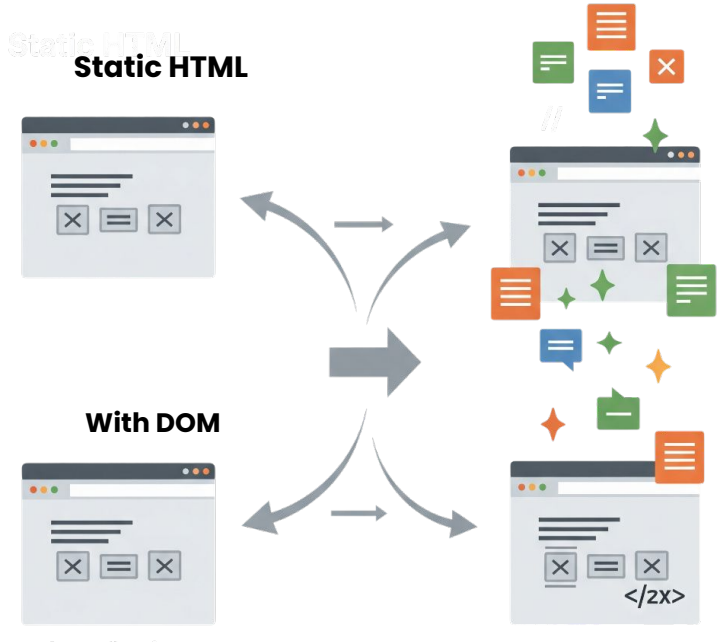
Newton School of Technology

Access Elements:

JavaScript can target any part of the page (like <p>, , inputs) using DOM methods.

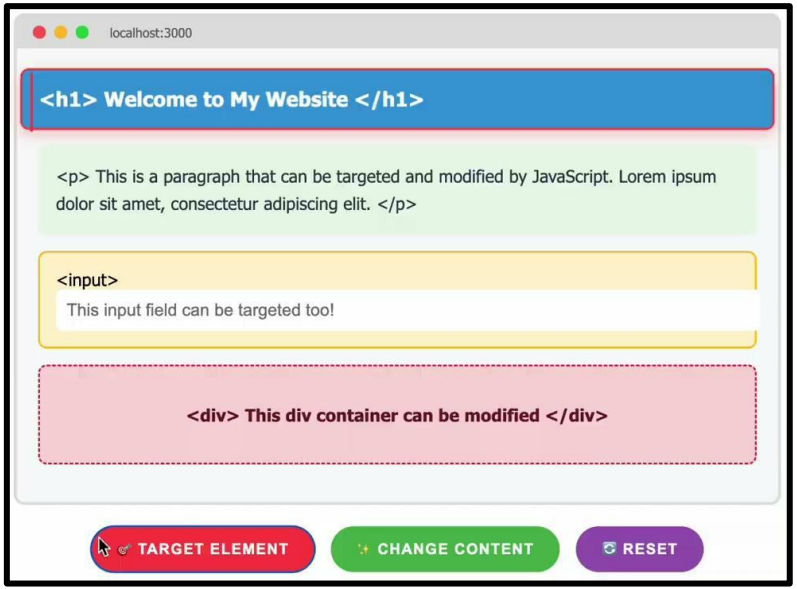
Change Content Dynamically:
Update text, HTML, or styles instantly without reloading the page.
E.g. innerText, innerHTML, value

Add/Remove Elements:
Create or delete elements on the fly to build dynamic UIs.
E.g. update lists, generate new content.



Why the DOM Matters

Newton School of Technology

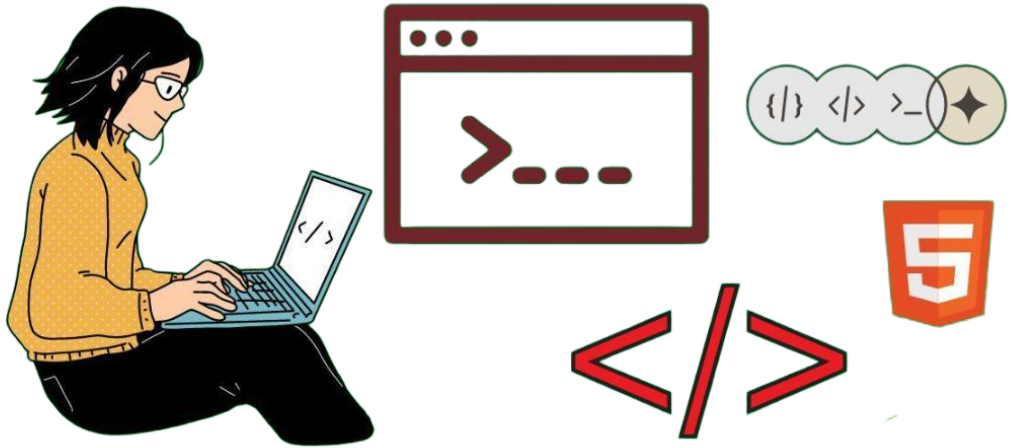



```javascript
// 1. Targeting Header Element
const header = document.getElementById('header-element');
header.innerHTML = '🎯 TARGETED & CHANGED!';
header.style.background = '#e74c3c';

// 2. Targeting Paragraph Element
const paragraph = document.querySelector('.paragraph');
paragraph.innerHTML = '✨ JavaScript dynamically changed this paragraph content!';
paragraph.style.background = '#f39c12';

// 3. Targeting Input Element
const input = document.querySelector('input');
input.value = 'JavaScript changed this input value!';
input.style.background = '#3498db';
input.style.color = 'white';

// 4. Targeting Div Element
const div = document.getElementById('div-element');
div.innerHTML = '🚀 Modified by JS!';
div.style.background = '#27ae60';
```

Selectors in JavaScript

Newton School of Technology

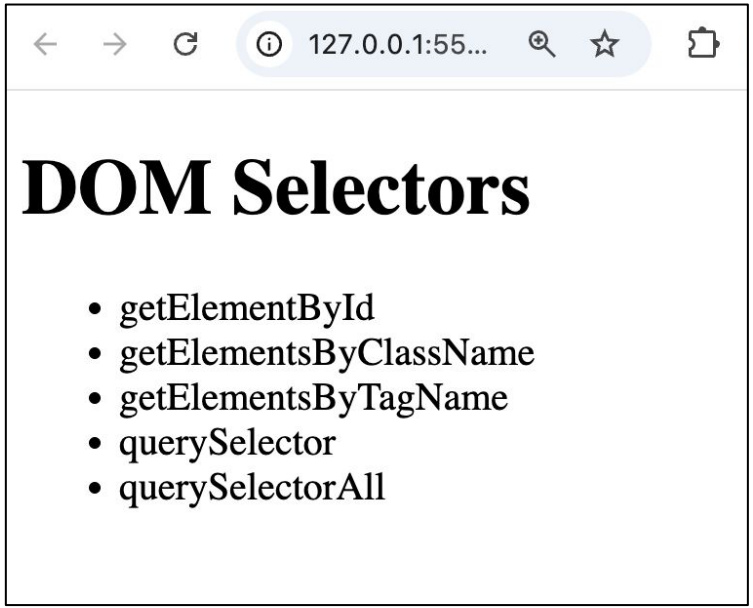
To make a webpage interactive, you first need to access specific parts of it — like a **heading**, **button**, or **paragraph**.
JavaScript gives you tools called **selectors** to do this. Selectors help you find the exact elements on a webpage that you want to work with. Once selected, you can **change their content**, **style**, or behavior using JavaScript.



Selectors in JavaScript

Newton School of Technology

DOM Selectors, as the name suggests is used to **select HTML elements within a document** using JavaScript. There are multiple ways to select elements in the DOM. Here are 5 of the most commonly used approaches




```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Document Object Model</title>
  </head>
  <body>
    <h1 class="heading">DOM Selectors</h1>
    <ul>
      <li class="selectors">getElementById</li>
      <li class="selectors">getElementsByClassName</li>
      <li id="item">getElementsByTagName</li>
      <li id="item">querySelector</li>
      <li id="item">querySelectorAll</li>
    </ul>
  </body>
</html>
```

HTMLCollection

Newton School of Technology

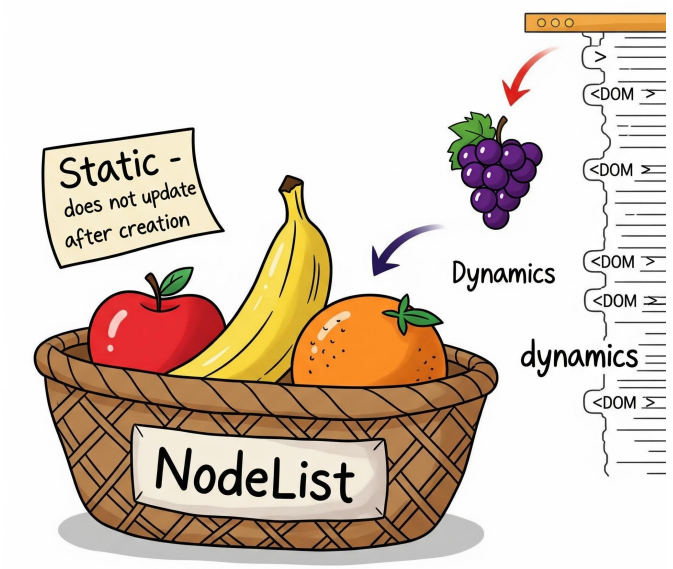
- HTMLCollection is a **live** collection of document elements that **updates automatically** when the DOM changes.
- Items can be accessed by **name, ID,** or **index**. For example, if you add a **<li>** to a list, the HTMLCollection reflects the change instantly.
- The getElementsByClassName() and getElementsByTagName() methods return a live HTMLCollection.

The fruit automatically appears in the basket

Difference between HTMLCollection & NodeList

Newton School of Technology

- A NodeList is a **collection of nodes**, which can be elements, **text**, or attributes from the document.
- NodeList items can only be accessed by their **index** number.
- NodeList is often static (e.g., from `querySelectorAll()`), but **some methods like `childNodes` return live NodeLists**
  Example: If you add a **<li>** element to a list in the DOM, the list in NodeList will not change.

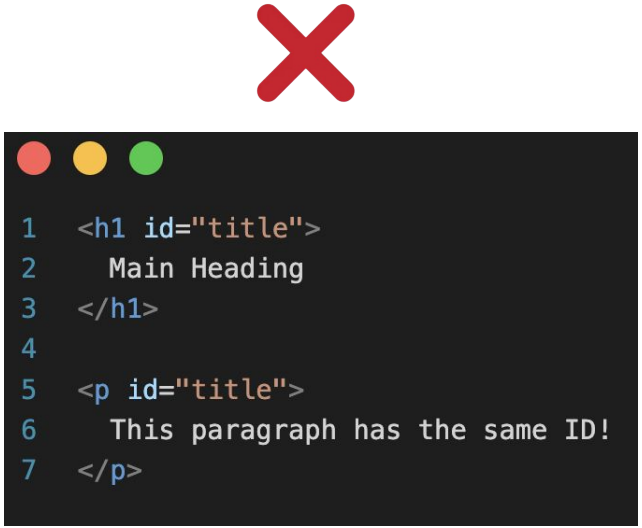


Id attribute in HTML

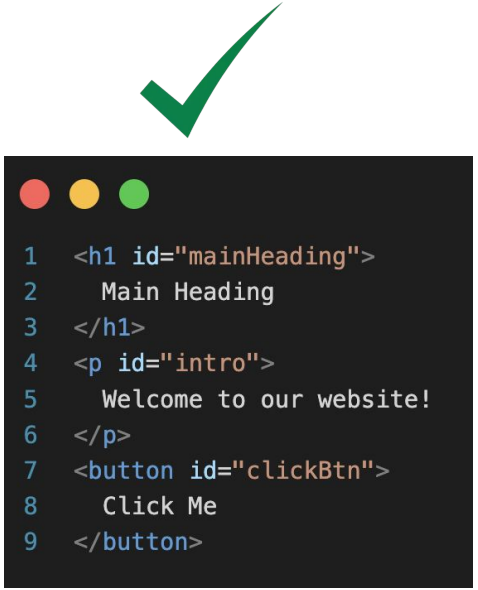
Newton School of Technology

An ID is a **unique identifier** assigned to an HTML element using the id attribute. It **must be unique** across the entire HTML document — meaning **only one element can have a specific ID**.

```
1  <h1 id="title">
2    Welcome
3  </h1>
```



```
1  <h1 id="title">
2    Main Heading
3  </h1>
4
5  <p id="title">
6    This paragraph has the same ID!
7  </p>
```



```
1  <h1 id="mainHeading">
2    Main Heading
3  </h1>
4  <p id="intro">
5    Welcome to our website!
6  </p>
7  <button id="clickBtn">
8    Click Me
9  </button>
```

getElementById

Newton School of Technology

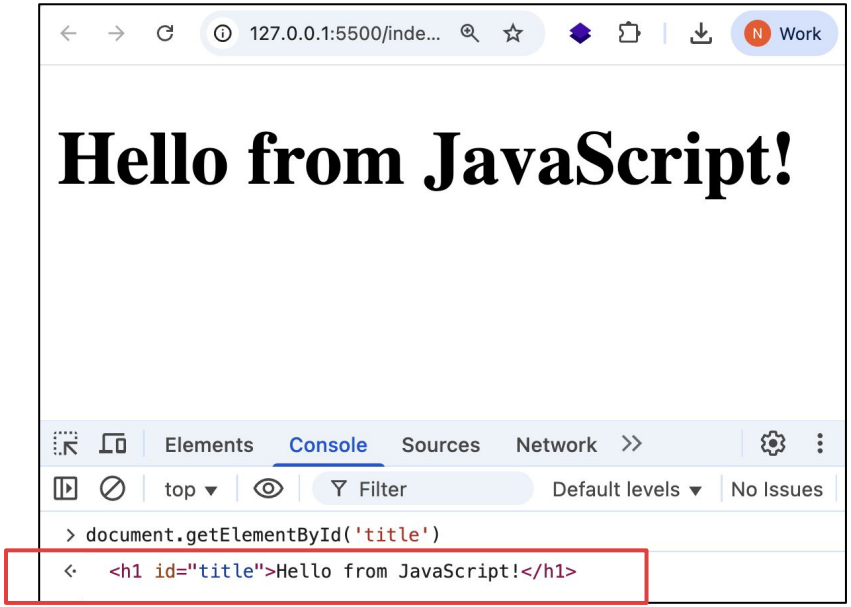
This method **returns exactly one element** — the element with the **matching ID**. If **no element** is found with that ID, it **returns null**.

```
1  <h1 id="title">Welcome</h1>
2
3  <script>
4    const heading = document.getElementById("title");
5    heading.innerText = "Hello from JavaScript!";
6  </script>
```

getElementById

Newton School of Technology

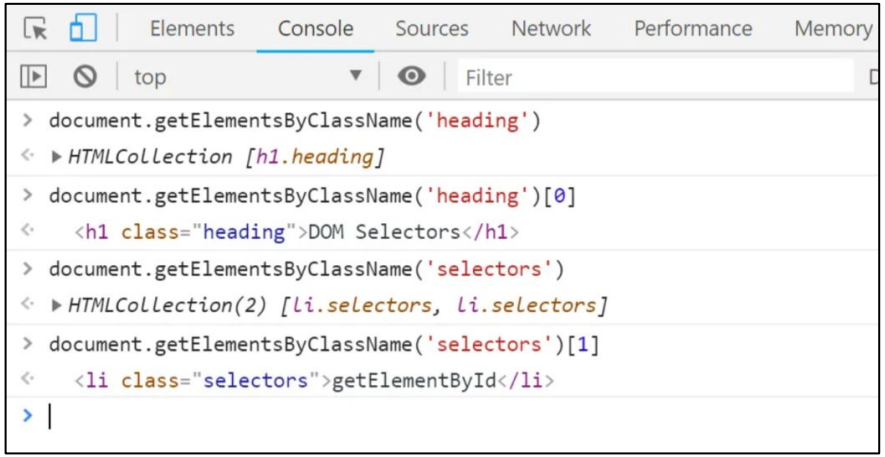
Here the **id="title"** is used to **uniquely identify the <h1> element**, which JavaScript accesses using getElementById("title")



getElementsByClassName

Newton School of Technology

This method returns all the elements **that matches the specified Class name.**
It returns a live **HTMLCollection**, which updates if the DOM changes. You can access elements using indexing like an array.

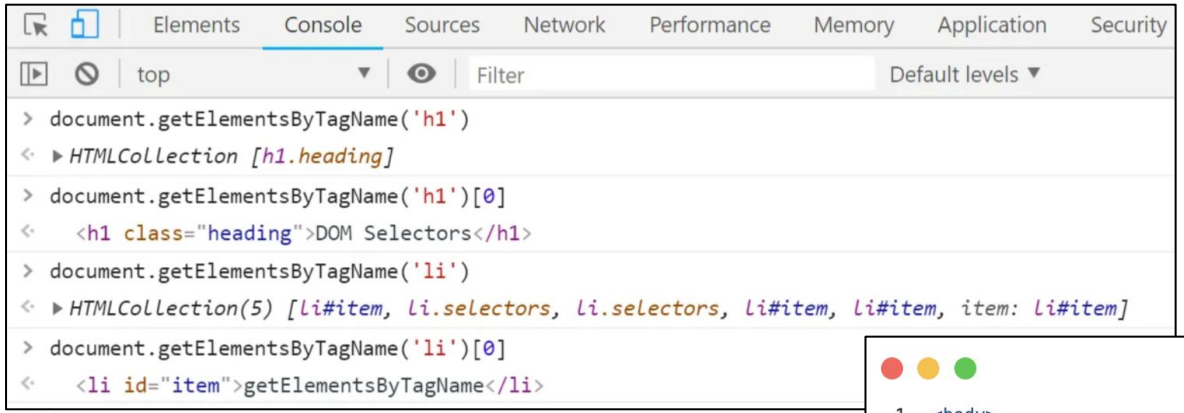


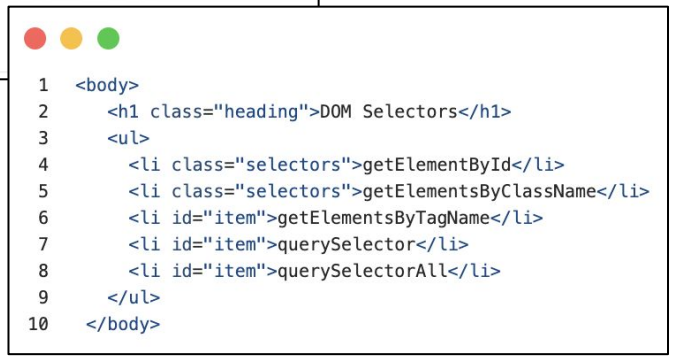

```
1  <body>
2     <h1 class="heading">DOM Selectors</h1>
3     <ul>
4       <li class="selectors">getElementById</li>
5       <li class="selectors">getElementsByClassName</li>
6       <li id="item">getElementsByTagName</li>
7       <li id="item">querySelector</li>
8       <li id="item">querySelectorAll</li>
9     </ul>
10  </body>
```

getElementsByTagName

Newton School of Technology

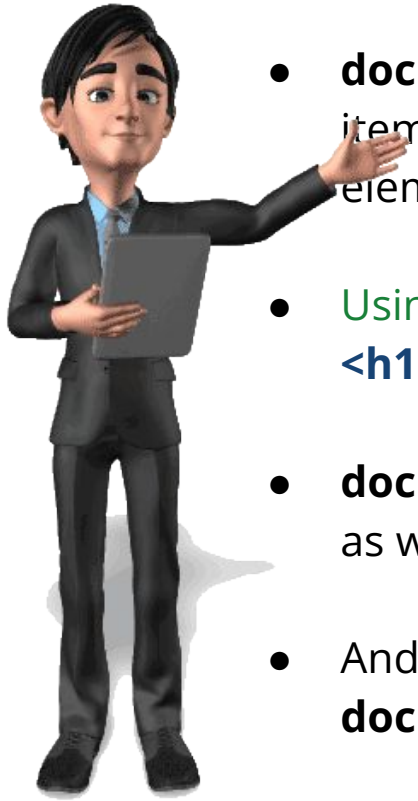
This method selects all elements with a **specific tag** name (like **div, p, li**) and returns a live **HTMLCollection**.





getElementsByTagName

Newton School of Technology



- **document.getElementsByTagName('h1')** returns a collection of items **matching the tag name h1**. And since we got only one h1 element, the list contains only one element.

- Using [0] as index, we can access the first element in the list which is **<h1 class="heading">DOM Selectors</h1>.**

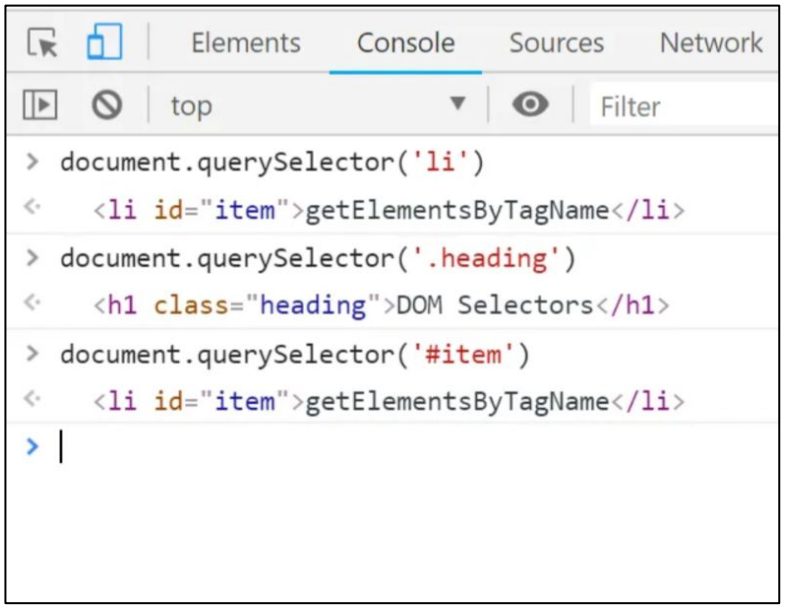
- **document.getElementsByTagName('li')** returns a list of 5 elements as we have five li tags in our page.

- And any individual element can be selected by using it's index such as **document.getElementsByTagName('li')[0]**.

querySelector

Newton School of Technology

This method **returns the first element** that matches a CSS selector. It's very flexible: you can use **#id, .class, tag**, or even attribute selectors.




```
1   <body>
2      <h1 class="heading">DOM Selectors</h1>
3      <ul>
4        <li class="selectors">getElementById</li>
5        <li class="selectors">getElementsByClassName</li>
6        <li id="item">getElementsByTagName</li>
7        <li id="item">querySelector</li>
8        <li id="item">querySelectorAll</li>
9      </ul>
10   </body>
```

querySelector

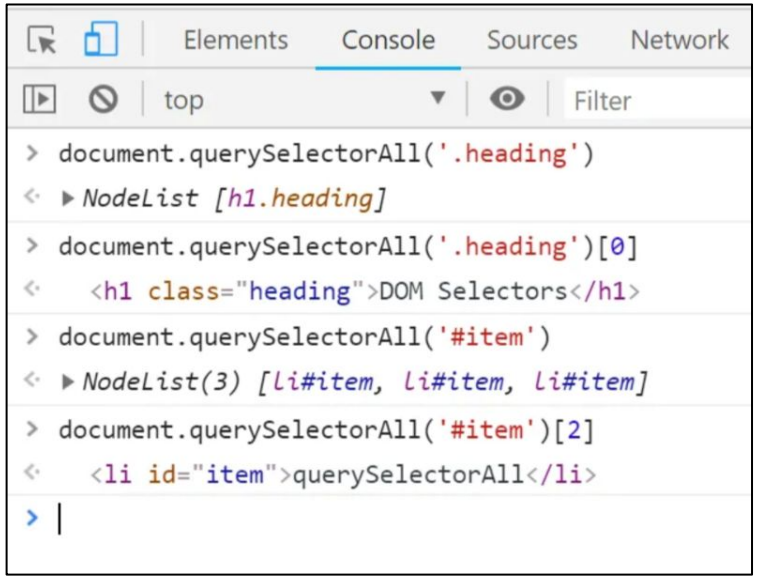
Newton School of Technology

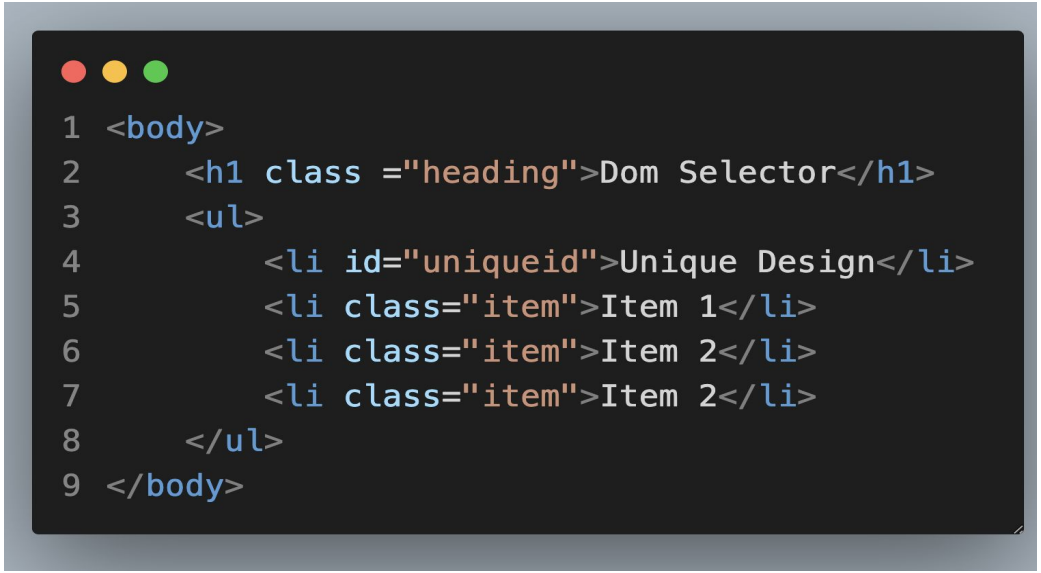
- **document.querySelector('li')** returns the first element that matches the CSS selector li. Remaining elements are ignored.
- **document.querySelector('.heading')** returns the first element that matches the CSS selector **.heading.**
- **document.querySelector('#item')** returns the first element that matches the CSS selector **#heading.**
- As you can see, we can use all kinds of CSS selectors within the querySelector method that we will use in a normal CSS file.

querySelectorAll

Newton School of Technology

This **returns all matching elements** using any valid CSS selector. Unlike class/tag methods, it returns a **static NodeList**, so changes to the DOM won't reflect automatically.

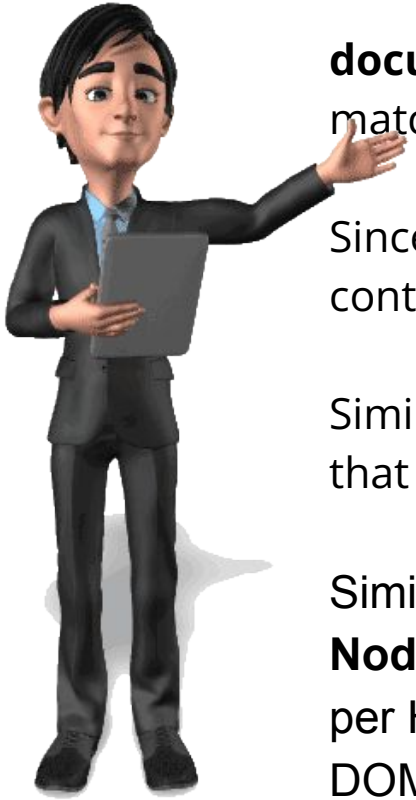




```
1 <body>
2     <h1 class ="heading">Dom Selector</h1>
3     <ul>
4         <li id="uniqueid">Unique Design</li>
5         <li class="item">Item 1</li>
6         <li class="item">Item 2</li>
7         <li class="item">Item 2</li>
8     </ul>
9 </body>
```

querySelector

Newton School of Technology



document.querySelectorAll('.heading') returns a list of all elements that matches the specified CSS selector.

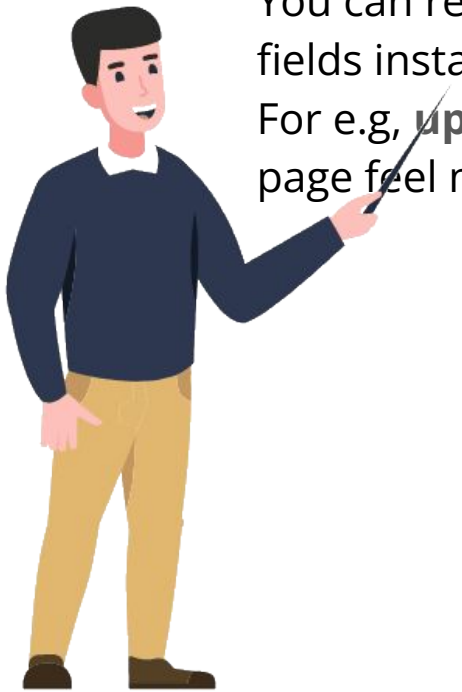
Since we have only one element under the class name .**heading,** the list contains one element. And it can be accessed by it's **index**.

Similarly, **document.querySelectorAll('.item')** returns a list of 3 items that matches the CSS selector.

Similarly, **`document.querySelectorAll('#item')`** returns a **NodeList** of all elements that have an `id` equal to **`"item"`** (although, as per HTML standards, an `id` should be unique for each element in the DOM).

Accessing & Updating DOM Content

Newton School of Technology



JavaScript lets you interact with a webpage's content after it loads.
You can read or change elements like **headings**, **paragraphs**, or **input** fields instantly — without reloading the page.
For e.g, **update a title** or **show a message dynamically**, making the page feel more interactive.

You can access and update DOM content using built-in JavaScript properties.

1. innerText
2. innerHTML
3. value

innerText

Newton School of Technology

innerText lets you get or change the visible text inside an HTML element. It ignores any HTML tags and only deals with what the user actually sees on the screen.

```html
<h2 id="title">Welcome!</h2>
```

```javascript
const title = document.getElementById('title');
title.innerText = "Hello, DOM!";
```

innerText

Newton School of Technology

innerText lets you get or change the visible text inside an HTML element. It ignores any HTML tags and only deals with what the user actually sees on the screen.

```
1   <h2 id="title">Welcome!</h2>
```

```
1   const title = document.getElementById('title');
2   title.innerText = "Hello, DOM!";
```

innerHTML

Newton School of Technology

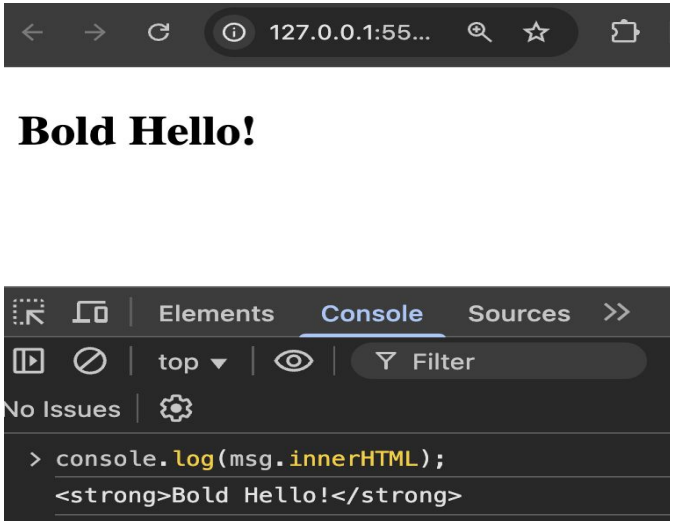
innerHTML is used when you want to **get** or **set** both text and HTML tags inside an element.

```
1   <p id="msg">Hello!</p>
```

```
1   const msg = document.getElementById('msg');
2   msg.innerHTML = "<strong>Bold Hello!</strong>";
3
```

innerHTML

Newton School of Technology

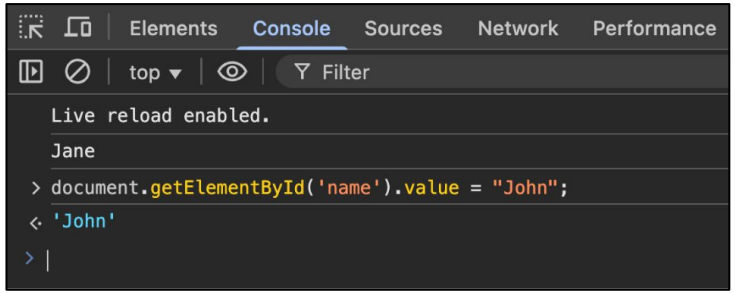


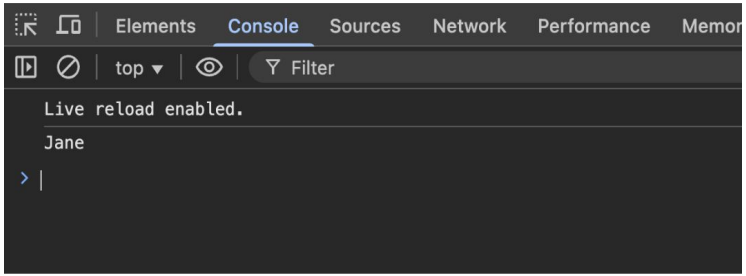
value

Newton School of Technology

The **value** property lets you **read** or **update** the content of input fields, like text boxes or forms. It is the go-to method for accessing what the user typed in.

<input type="text" id="name" value="Jane">





Newton School of Technology

Quiz Time

References

Newton School of Technology

I. Introduction & Motivation
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

https://javascript.info/dom-nodes

II. Understanding the DOM
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

https://javascript.info/dom-nodes#dom-tree

III. Selectors in JavaScript
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents#finding_elements_in_the_dom

https://javascript.info/searching-elements-dom

IV. Accessing & Updating DOM Content
https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/innerText

https://javascript.info/modifying-document

Feedback

Newton School of Technology

Now it's time to give your valuable feedback

Newton School of Technology

Thanks for watching!