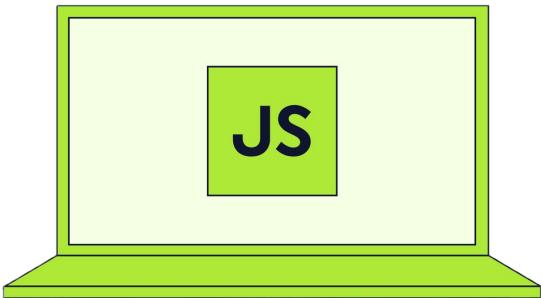




**Newton School  
of Technology**

# The Complete Javascript Course



@newtonschool

# Lecture 21: DOM Introduction & Selectors

- Nishchal Gupta



# What we have learnt so far

- 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



# JavaScript

# Before the DOM – The Static Web

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.



JULY 14

 Welcome to Apple 1997

EMATE 300  
Mobile, Affordable, & Smart

BMW

Introducing CyberDrive

Register today for a free CD-ROM.

MOVIES FROM MARS  
QuickTime VR takes you out of this world

What's Hot

Preorder Mac OS 8  
Now you can [preorder Mac OS 8](#), described by Macworld as "the most comprehensive update to the Mac OS in years, sporting a bold new look, a speedier Finder, more shortcuts and integrated Internet functions."

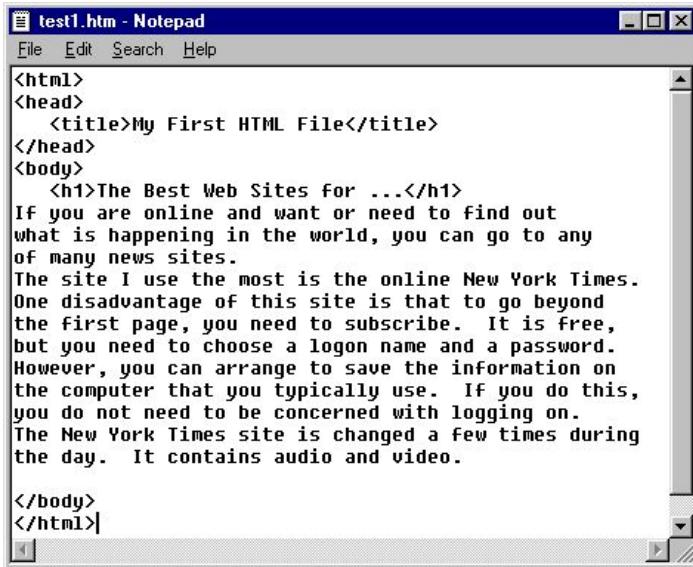
Be the First to Know  
Learn about new Macintosh software releases the moment they become available. Check [Hot Mac Products](#) to hear about programs like Speed Demon, ReBirth RB-338 and QuickCRC.

Newton Connects  
Newton, Inc., will enhance

Where to Buy

# Before the DOM – The Static Web

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



test1.htm - Notepad

File Edit Search Help

```
<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>
```



# Before the DOM – The Static Web

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

```
1 <html>
2 <head>
3   <title>My First HTML File</title>
4 </head>
5 <body>
6   <h1>The Best Web Sites for ...</h1>
7   If you are online and want or need to find out
8   what is happening in the world, you can go to any
9   of many news sites.
10  The site I use the most is the online New York Times.
11  One disadvantage of this site is that to go beyond
12  the first page, you need to subscribe. It is free,
13  but you need to choose a logon name and a password.
14  However, you can arrange to save the information on
15  the computer that you typically use. If you do this,
16  you do not need to be concerned with logging on.
17  The New York Times site is changed a few times during
18  the day. It contains audio and video.
19 </body>
20 </html>
21
```

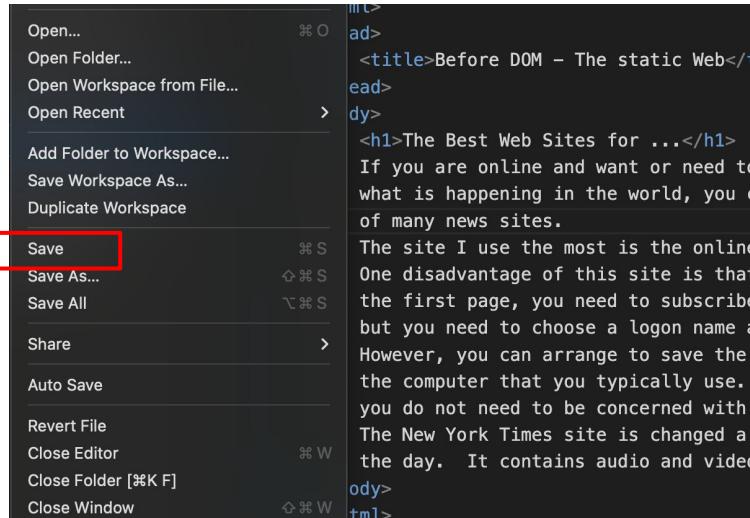
Manually open the HTML file.

```
1 <html>
2 <head>
3   <title>Before DOM – The static Web</title>
4 </head>
5 <body>
6   <h1>The Best Web Sites for ...</h1>
7   If you are online and want or need to find out
8   what is happening in the world, you can go to any
9   of many news sites.
10  The site I use the most is the online New York Times.
11  One disadvantage of this site is that to go beyond
12  the first page, you need to subscribe. It is free,
13  but you need to choose a logon name and a password.
14  However, you can arrange to save the information on
15  the computer that you typically use. If you do this,
16  you do not need to be concerned with logging on.
17  The New York Times site is changed a few times during
18  the day. It contains audio and video.
19 </body>
20 </html>
21
```

Edit the text in a code editor.

# Before the DOM – The Static Web

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



Save the file.



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

 Refresh

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

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)

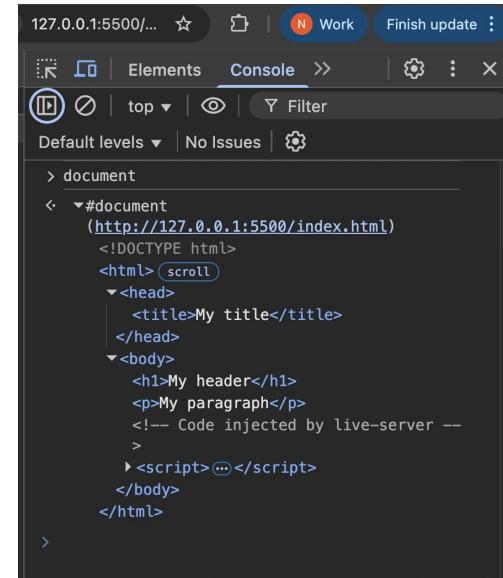
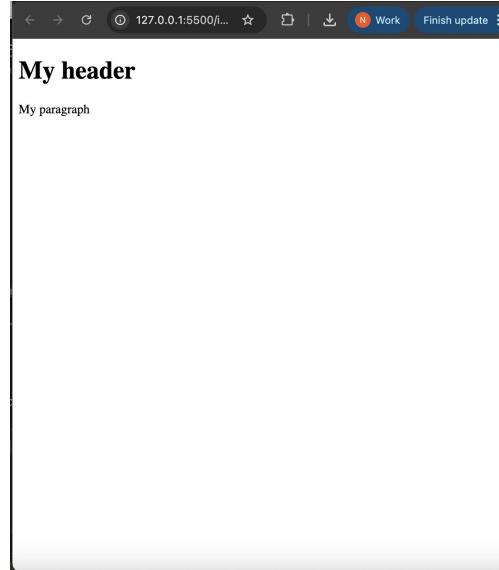
# What changed: The DOM

**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
```

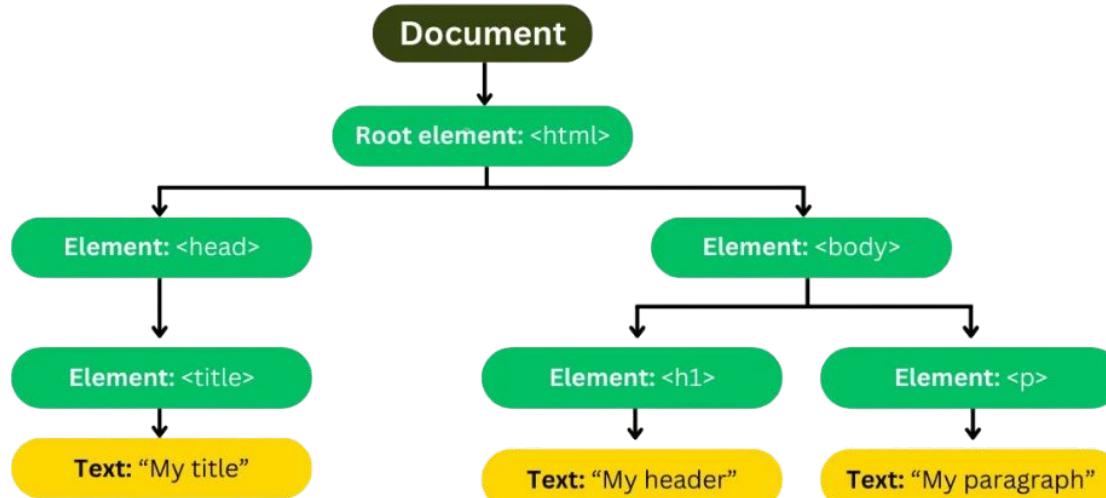


# What changed: The DOM

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

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.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Static Sale Banner</title>
5 </head>
6 <body>
7   <h1>SALE 10% OFF</h1>
8 </body>
9 </html>
10
```



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Static Sale Banner</title>
5 </head>
6 <body>
7   <h1>SALE 20% OFF</h1>
8 </body>
9 </html>
10
```

# What changed: The DOM

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



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



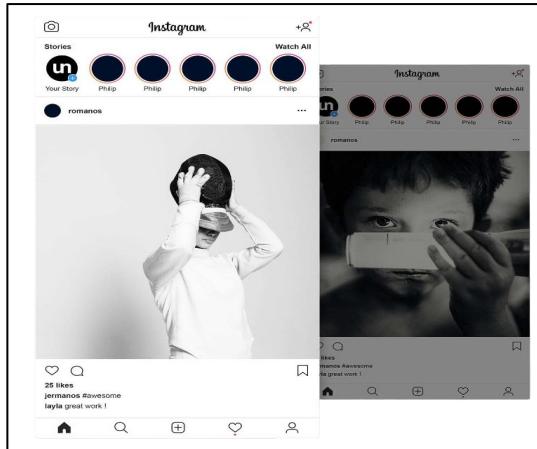
```
1  document.getElementById("sale").innerText = "SALE 20% OFF";
```

# What changed: The DOM

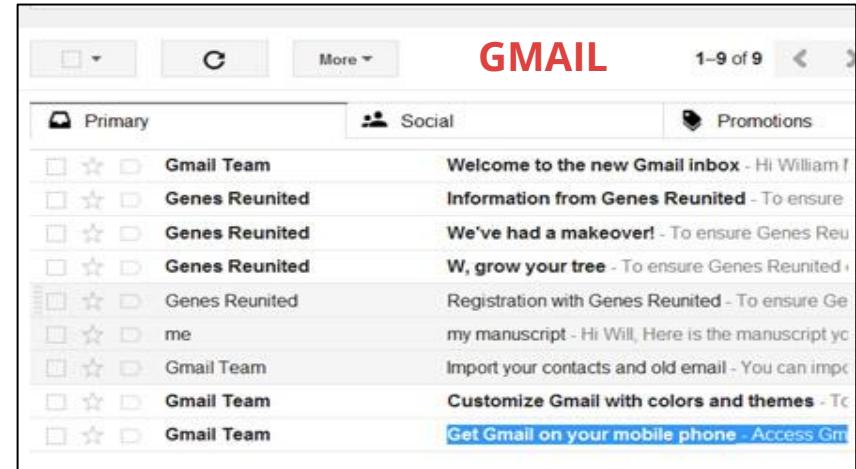
- Before the DOM, web pages were just **static** display boards — what you saw was **fixed** unless someone manually edited the HTML file, saved it, and re-uploaded it to the server.
- There was no way to change the content after the page had loaded.
- With the introduction of the **DOM** and **JavaScript**, 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

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

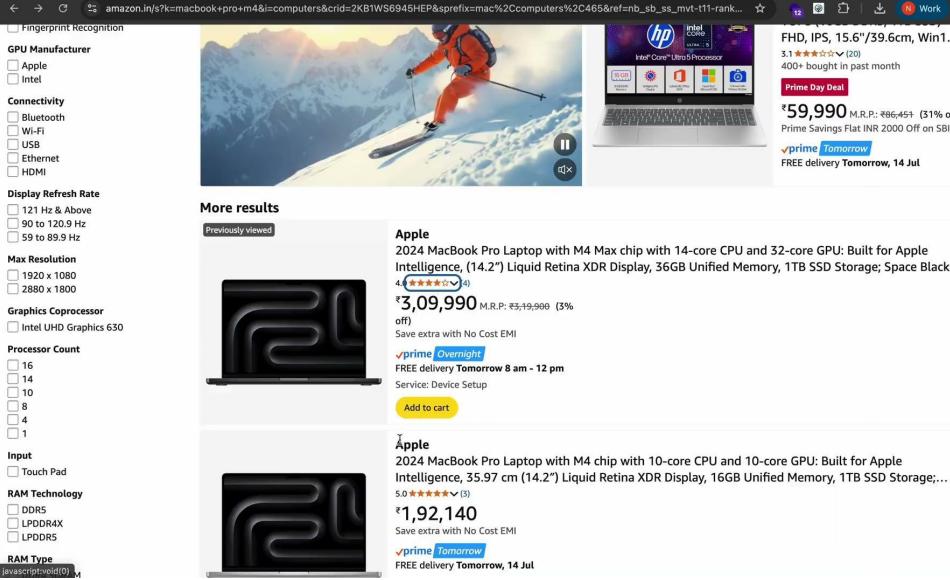


# Modern Examples That Use the DOM

## Amazon:

You add 3 items to cart — total price updates live.

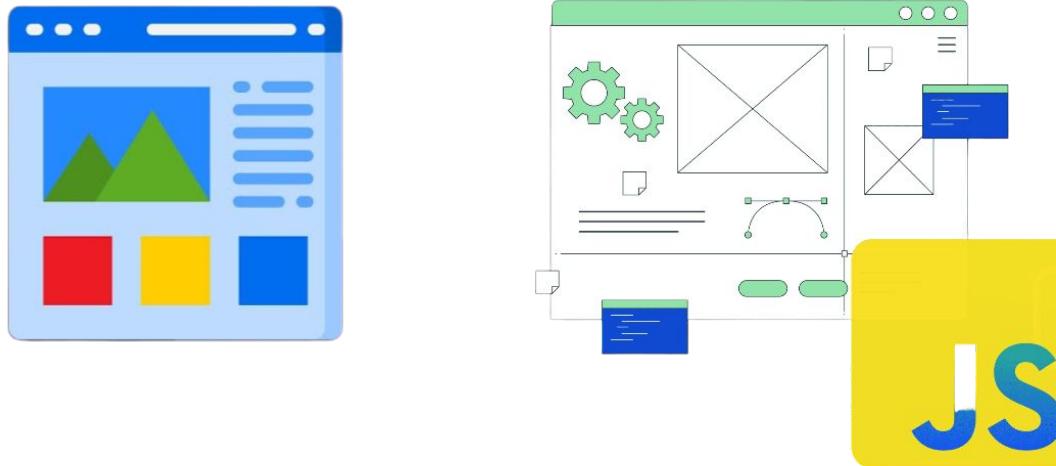
Again, only the number and total update — the whole page doesn't reload.



The screenshot shows an Amazon search results page for "macbook pro+m4". The main content displays two MacBook Pro laptops from Apple. The top listing is a "2024 MacBook Pro Laptop with M4 Max chip with 14-core CPU and 32-core GPU: Built for Apple Intelligence, (14.2") Liquid Retina XDR Display, 36GB Unified Memory, 1TB SSD Storage; Space Black". It is priced at ₹3,09,990 (M.R.P.: ₹3,19,999) and has a 4.4-star rating. The bottom listing is another "Apple 2024 MacBook Pro Laptop with M4 chip with 10-core CPU and 10-core GPU: Built for Apple Intelligence, 35.97 cm (14.2") Liquid Retina XDR Display, 16GB Unified Memory, 1TB SSD Storage; Space Grey". It is priced at ₹1,92,140 and has a 5.0-star rating. Both listings include Prime Day Deal and free delivery information. The left sidebar contains various filtering options such as GPU Manufacturer (Apple, Intel), Connectivity (Bluetooth, Wi-Fi, USB, Ethernet, HDMI), Display Refresh Rate (121 Hz & Above, 90 to 120.9 Hz, 59 to 89.9 Hz), Max Resolution (1920 x 1080, 2880 x 1800), Graphics Coprocessor (Intel UHD Graphics 630), Processor Count (16, 14, 10, 8, 4, 1), Input (Touch Pad), RAM Technology (DDR5, LPDDR4X, LPDDR5), and RAM Type (DDR5).

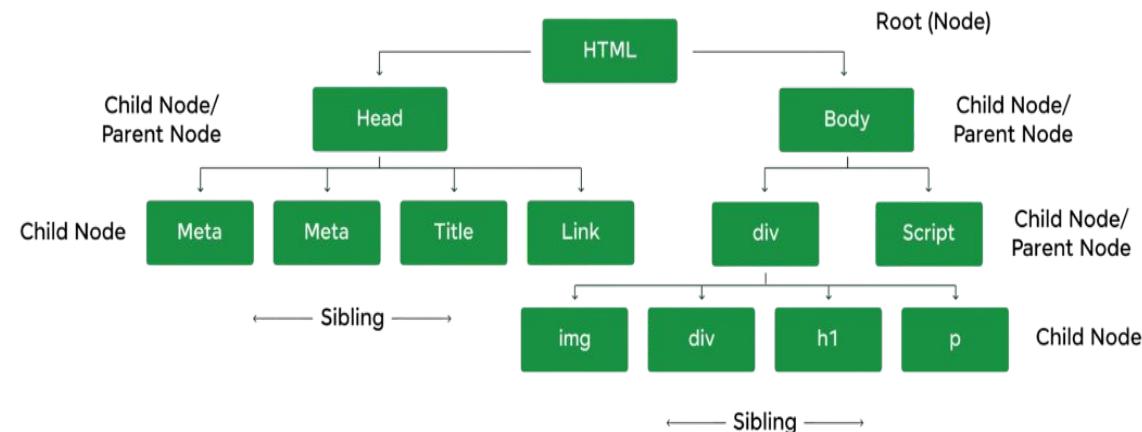
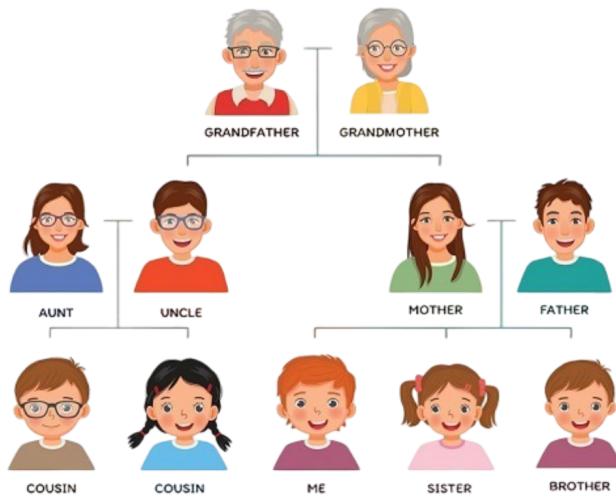
# Modern Examples That Use the DOM

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



# Understanding the DOM

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).

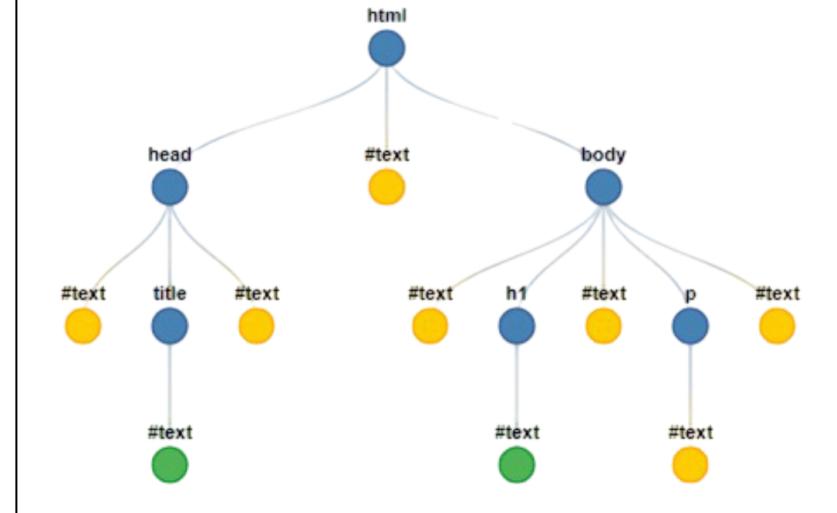


# Understanding the DOM

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

## Access Elements:

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

## Change Content Dynamically:

Update text, HTML, or styles instantly without reloading the page.

E.g. `innerText`,  
`innerHTML`, `value`

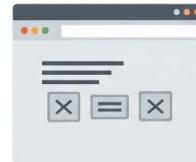
## Add/Remove Elements:

Create or delete elements on the fly to build dynamic UIs.

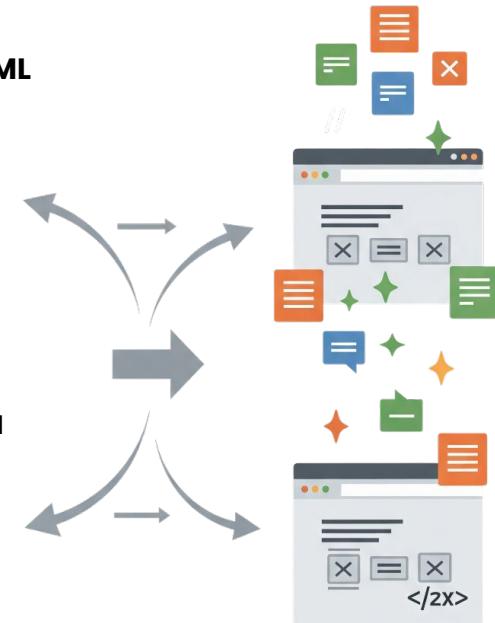
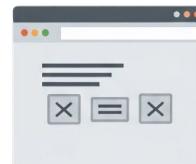
E.g. update lists, generate new content.

## Static HTML

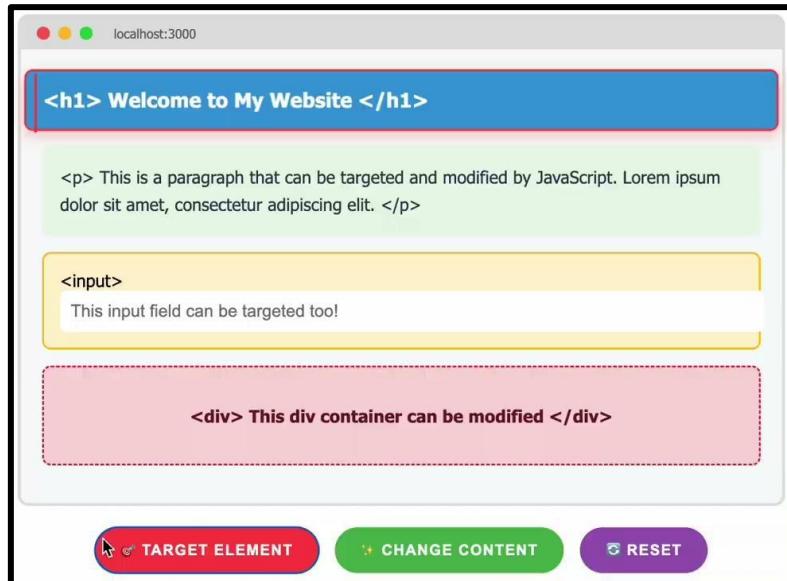
## Static HTML



## With DOM



# Why the DOM Matters



A screenshot of a web application at localhost:3000. The page contains the following elements:

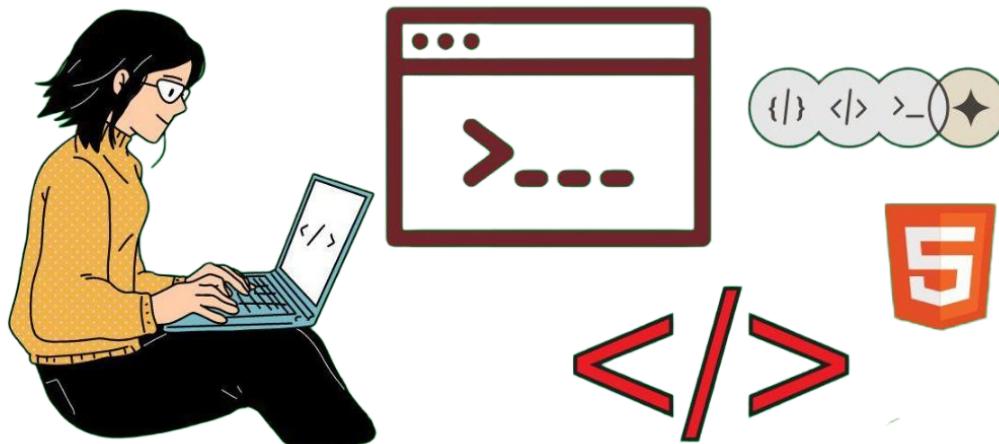
- A blue header bar with the text <h1> Welcome to My Website </h1>.
- A green paragraph area containing the text <p> This is a paragraph that can be targeted and modified by JavaScript. Lorem ipsum dolor sit amet, consectetur adipiscing elit. </p>.
- A yellow input field containing the text "This input field can be targeted too!".
- A pink div container with the text <div> This div container can be modified </div>.
- At the bottom are three buttons: "TARGET ELEMENT" (red), "CHANGE CONTENT" (green), and "RESET" (purple).

```
1 // 1. Targeting Header Element
2 const header = document.getElementById('header-element');
3 header.innerHTML = 'TARGETED & CHANGED!';
4 header.style.background = '#e74c3c';
5
6 // 2. Targeting Paragraph Element
7 const paragraph = document.querySelector('.paragraph');
8 paragraph.innerHTML = 'JavaScript dynamically changed this paragraph content!';
9 paragraph.style.background = '#f39c12';
10
11 // 3. Targeting Input Element
12 const input = document.querySelector('input');
13 input.value = 'JavaScript changed this input value!';
14 input.style.background = '#3498db';
15 input.style.color = 'white';
16
17 // 4. Targeting Div Element
18 const div = document.getElementById('div-element');
19 div.innerHTML = 'Modified by JS!';
20 div.style.background = '#27ae60';
```

# Selectors in JavaScript

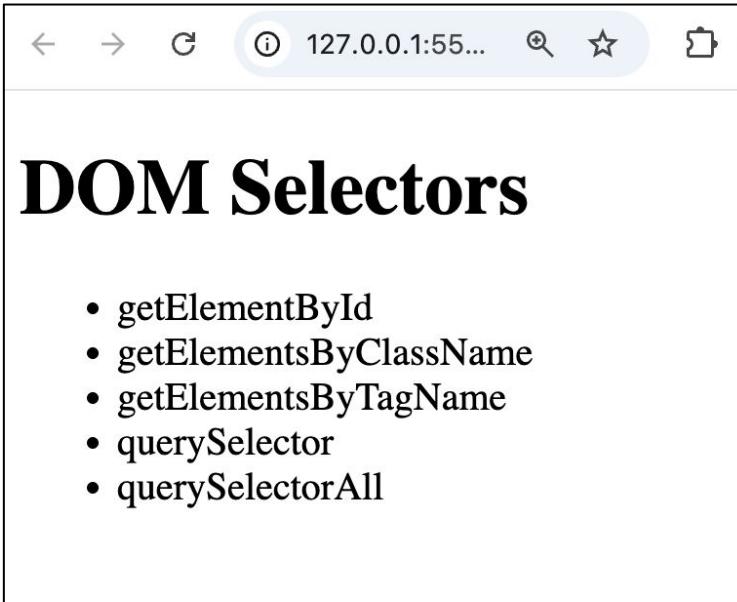
To make a webpage interactive, you first need to access specific parts of it — like a **heading**, **button**, or **paragraph**.

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.



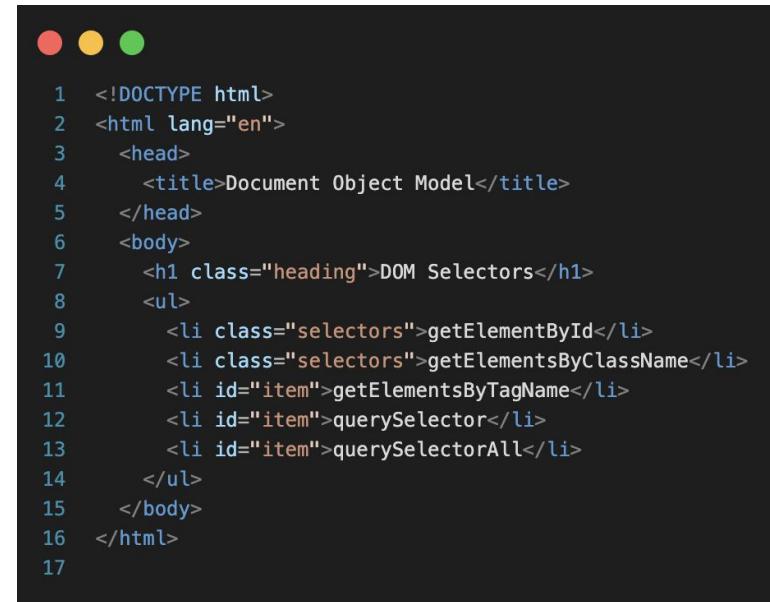
# Selectors in JavaScript

**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



DOM Selectors

- getElementById
- getElementsByClassName
- getElementsByTagName
- querySelector
- querySelectorAll



```
1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <title>Document Object Model</title>
5      </head>
6      <body>
7          <h1 class="heading">DOM Selectors</h1>
8          <ul>
9              <li class="selectors">getElementById</li>
10             <li class="selectors">getElementsByClassName</li>
11             <li id="item">getElementsByTagName</li>
12             <li id="item">querySelector</li>
13             <li id="item">querySelectorAll</li>
14         </ul>
15     </body>
16 </html>
17
```

# HTMLCollection

- 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.

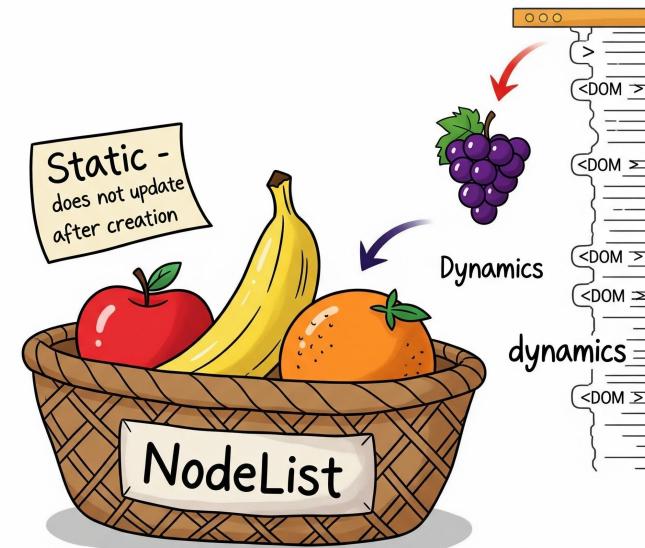


The fruit automatically  
appears in the basket

# Difference between HTMLCollection & NodeList

- 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

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

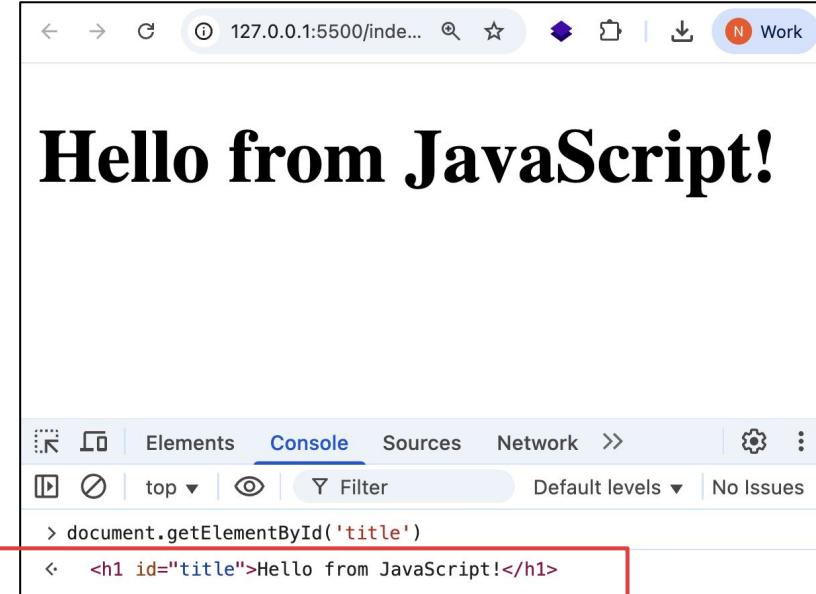
This method **returns exactly one element** — the element with the **matching ID**. If **no element** is found with that ID, it **returns null**.



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

# getElementById

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

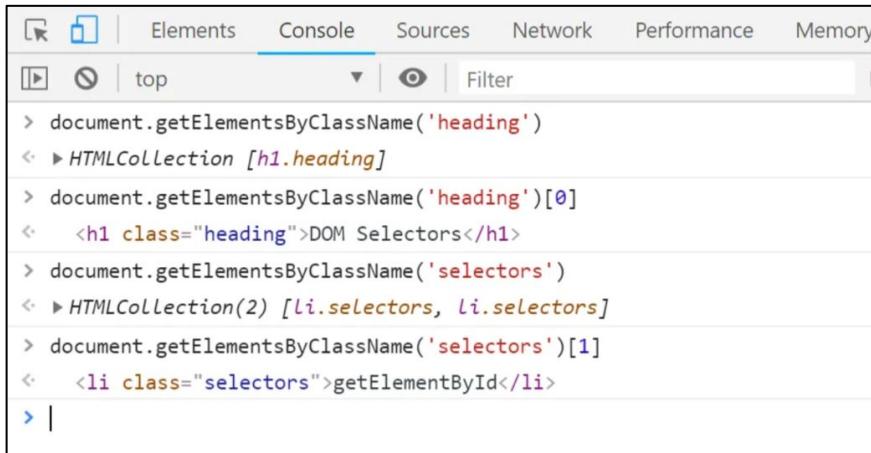


A screenshot of a browser window showing the result of a JavaScript command in the developer tools' Console tab. The browser address bar shows the URL `127.0.0.1:5500/index.html`. The main content area displays the text **Hello from JavaScript!**. Below the content, the developer tools interface is visible, specifically the Console tab. The console output shows the command `> document.getElementById('title')` followed by its result, which is the HTML element `<h1 id="title">Hello from JavaScript!</h1>`. This result is highlighted with a red rectangular box.

```
> document.getElementById('title')
< h1 id="title">Hello from JavaScript!</h1>
```

# getElementsByClassName

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.



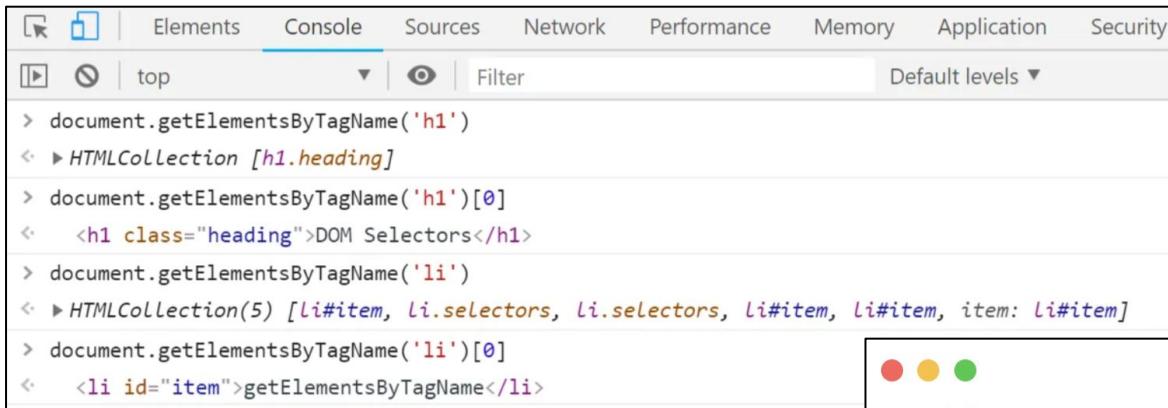
```
Elements Console Sources Network Performance Memory
top Filter
> document.getElementsByClassName('heading')
< HTMLCollection [h1.heading]
> document.getElementsByClassName('heading')[0]
< h1 class="heading">DOM Selectors</h1>
> document.getElementsByClassName('selectors')
< HTMLCollection(2) [li.selectors, li.selectors]
> document.getElementsByClassName('selectors')[1]
< li class="selectors">getElementById</li>
> |
```



```
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

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



The screenshot shows the Chrome DevTools Console tab with the following code examples:

```
> document.getElementsByClassName('h1')
< ▶ HTMLCollection [h1.heading]
> document.getElementsByClassName('h1')[0]
<  <h1 class="heading">DOM Selectors</h1>
> document.getElementsByClassName('li')
< ▶ HTMLCollection(5) [li#item, li.selectors, li.selectors, li#item, li#item, item: li#item]
> document.getElementsByClassName('li')[0]
<  <li id="item">getElementsByTagName</li>
```

A callout box highlights the first example: `document.getElementsByClassName('h1')`. Below the callout box is the corresponding DOM tree:

```
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



- **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

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.



```
Elements Console Sources Network
top Filter
> document.querySelector('li')
<li id="item">getElementsByTagName</li>
> document.querySelector('.heading')
<h1 class="heading">DOM Selectors</h1>
> document.querySelector('#item')
<li id="item">getElementsByTagName</li>
>
```



```
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>
```

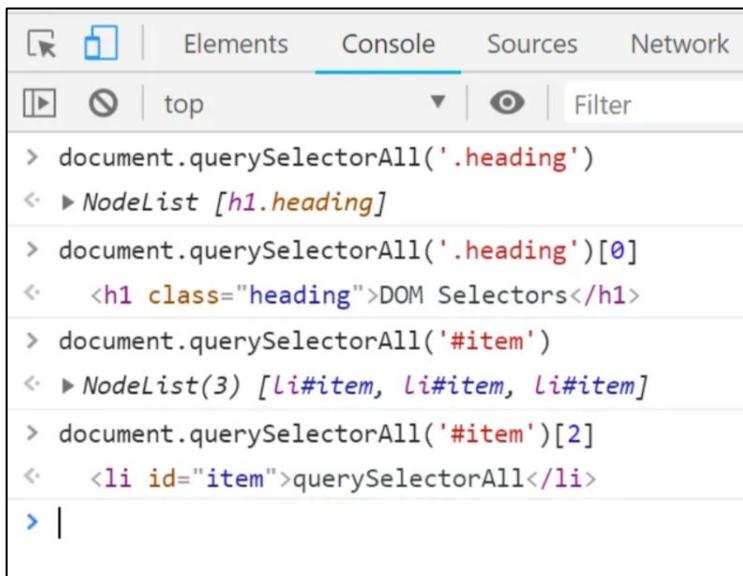
# querySelector



- **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.

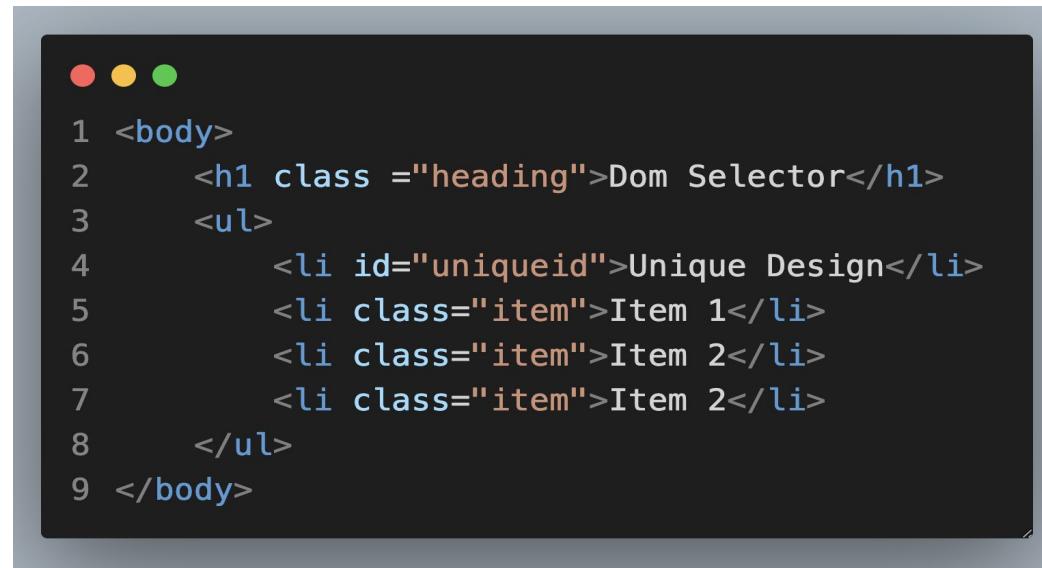
# querySelectorAll

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.



The screenshot shows the Chrome DevTools Elements tab. The console output is as follows:

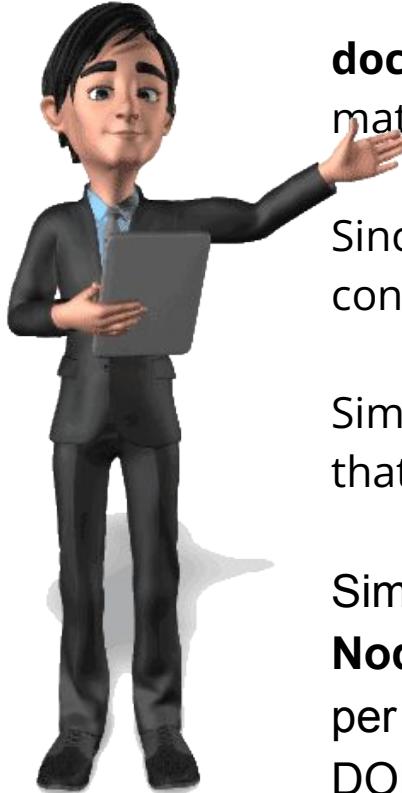
```
> document.querySelectorAll('.heading')
< NodeList [h1.heading]
> document.querySelectorAll('.heading')[0]
< <h1 class="heading">DOM Selectors</h1>
> document.querySelectorAll('#item')
< NodeList(3) [li#item, li#item, li#item]
> document.querySelectorAll('#item')[2]
< <li id="item">querySelectorAll</li>
> |
```



The screenshot shows a code editor with the following HTML code:

```
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



**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 its **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



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

**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!";
3
```

# innerText

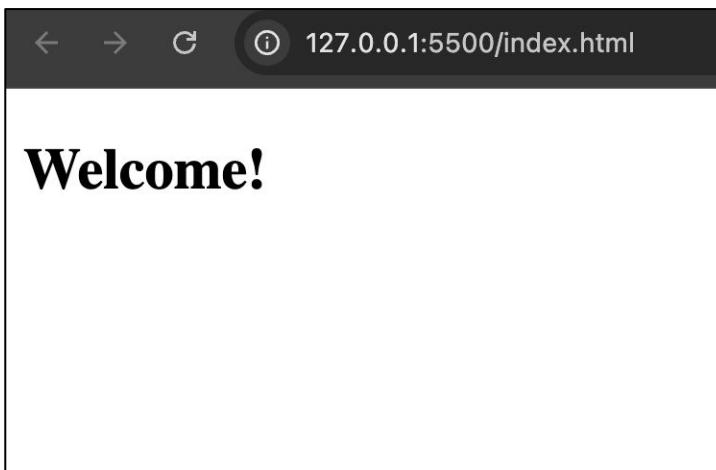
**innerText** lets you get or change the visible text inside an HTML element. It ignores any HTML tags and only deals with what the user actually sees on the screen.



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



```
1 const title = document.getElementById('title');
2 title.innerText = "Hello, DOM!";
3
```



# innerHTML

**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
```

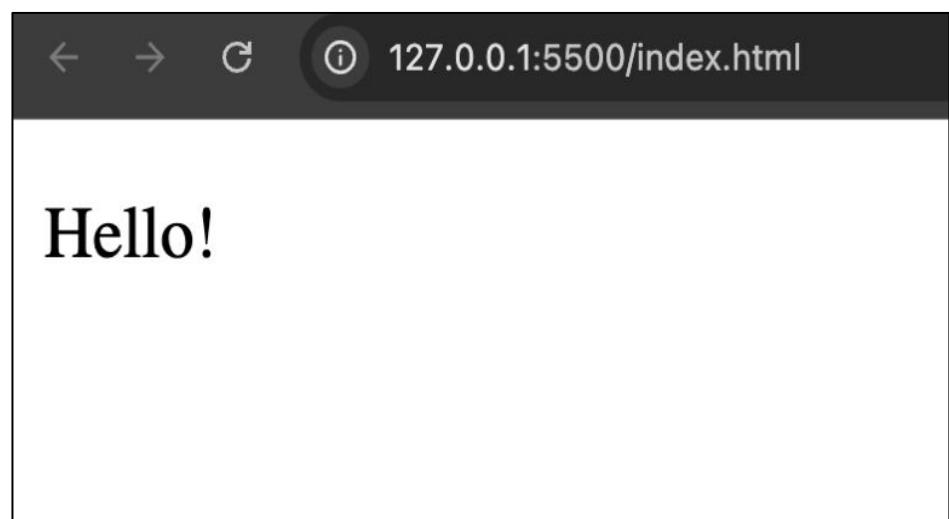
# innerHTML



**Bold Hello!**



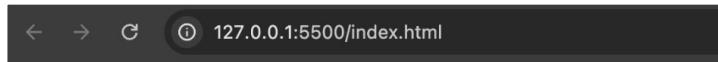
```
Elements Console Sources >
top ▾ Filter
No Issues | ⚙️
> console.log(msg.innerHTML);
<strong>Bold Hello!</strong>
```



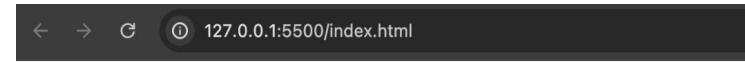
Hello!

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.

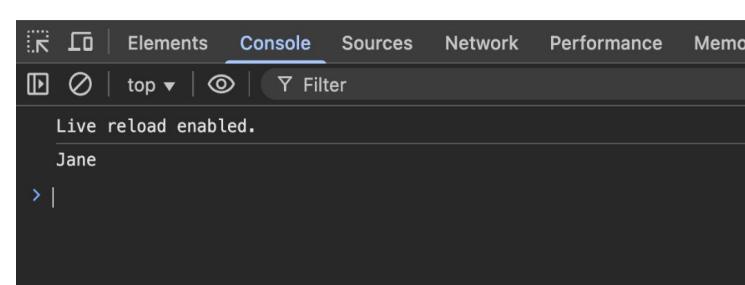
```
<input type="text" id="name" value="Jane">
```



John



Jane



# Quiz Time

# References

## I. Introduction & Motivation

[https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Introduction](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://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://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

Now it's time to give your valuable feedback



**Thanks  
for  
watching!**