🔟

Day-10:DOM-II

getElementsByClassName():

- The `getElementsByClassName()` method returns a collection of elements with a specified class name(s).
- The `getElementsByClassName()` method returns an array-like collection (list) of HTML elements.

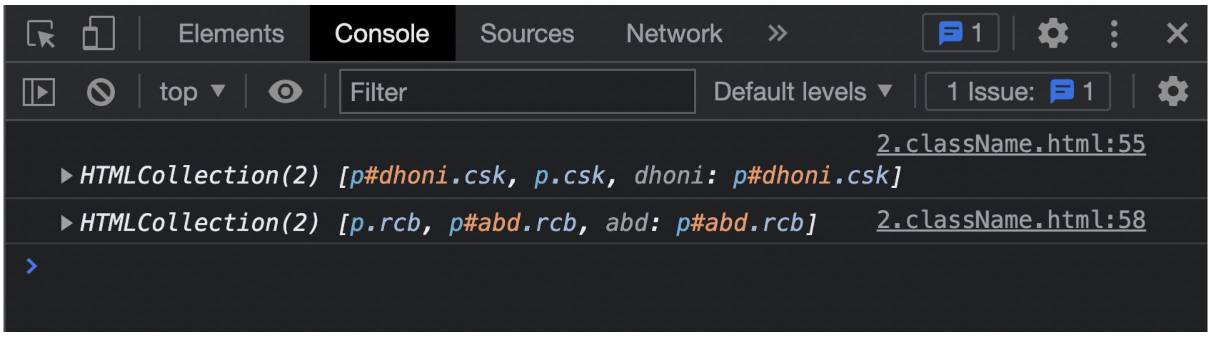
Syntax:

```
document.getElementsByClassName(classname)
```

Example:

```
<html lang="en">
  <head>
    <title>Document</title>
  </head>
  <body>
    <p class="csk" id="dhoni">I am Dhoni from Chennai</p>
    <p class="mi">I am Sachin from Mumbai</p>
    <p class="rcb">I am Kohli from Banglore</p>
    <p class="mi">I am Rohit from Mumbai</p>
    <p class="rcb" id="abd">I am ABD from Banglore</p>
    <p class="csk">I am Raina from Chennai</p>
  </body>
</html>

<script>
  var chennai = document.getElementsByClassName("csk");
  console.log(chennai);

  var banglore = document.getElementsByClassName("rcb");
  console.log(banglore);
</script>
```

Output:



```
▶HTMLCollection(2) [p#dhoni.csk, p.csk, dhoni: p#dhoni.csk]      2.className.html:55
▶HTMLCollection(2) [p.rcb, p#abd.rcb, abd: p#abd.rcb]            2.className.html:58
>
```

Live code: Codepen

- The length Property returns the number of elements in the collection.

```
  console.log(chennai.length) //2
```

- The elements in a collection can be accessed by index (starts at 0).
- If you want to access Dhoni from collection, index of dhoni paragraph is 0, so you can access as

```
  <script>
  var chennai = document.getElementsByClassName("csk");
  console.log(chennai[0]);
  console.log(chennai[1]);
</script>

Output:
<p class="csk" id="dhoni">I am Dhoni from Chennai</p>
<p class="csk">I am Raina from Chennai</p>
```

- You can also use for loop to access all elements at once

```
for (let i = 0; i < chennai.length; i++) {
  console.log(chennai[i])
}
```

getElementsByTagName()

- The `getElementsByTagName()` method returns a collection of elements with a specified class name(s).
- The `getElementsByTagName()` method returns an array-like collection (list) of HTML elements.

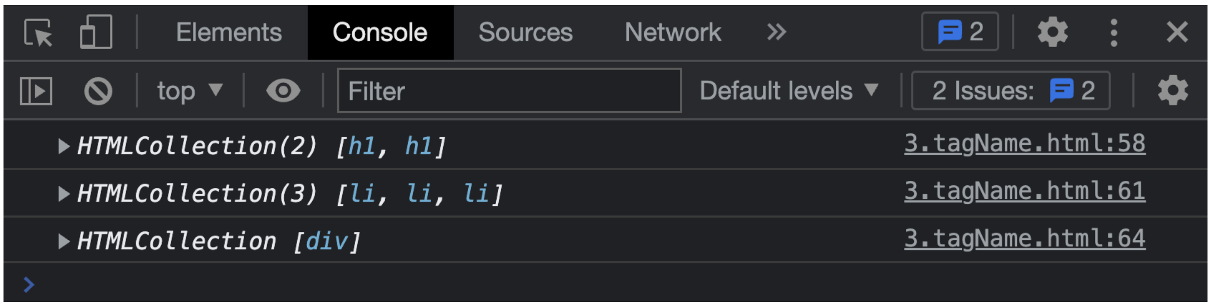
Syntax:

```
document.getElementsByTagName(tagname)
```

Example:

```
<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <h1>The Document Object</h1>
    <h1>The getElementsByTagName() Method</h1>
    <ul>
      <li>Coffee</li>
      <li>Tea</li>
      <li>Milk</li>
    </ul>
    <p>I am p1</p>
    <p>I am p2</p>
    <div>I am box</div>
  </body>
</html>

<script>
  let h1 = document.getElementsByTagName("h1");
  console.log(h1) // it should return 2 h1 tags

  let list = document.getElementsByTagName("li");
  console.log(list) // it should return 3 li tags

  let div = document.getElementsByTagName("div")
  console.log(div). // it should return 1 div tag
</script>
```

Output:



```
▶ HTMLCollection(2) [h1, h1]          3.tagName.html:58
▶ HTMLCollection(3) [li, li, li]      3.tagName.html:61
▶ HTMLCollection [div]                3.tagName.html:64
```

Live code : Codepen

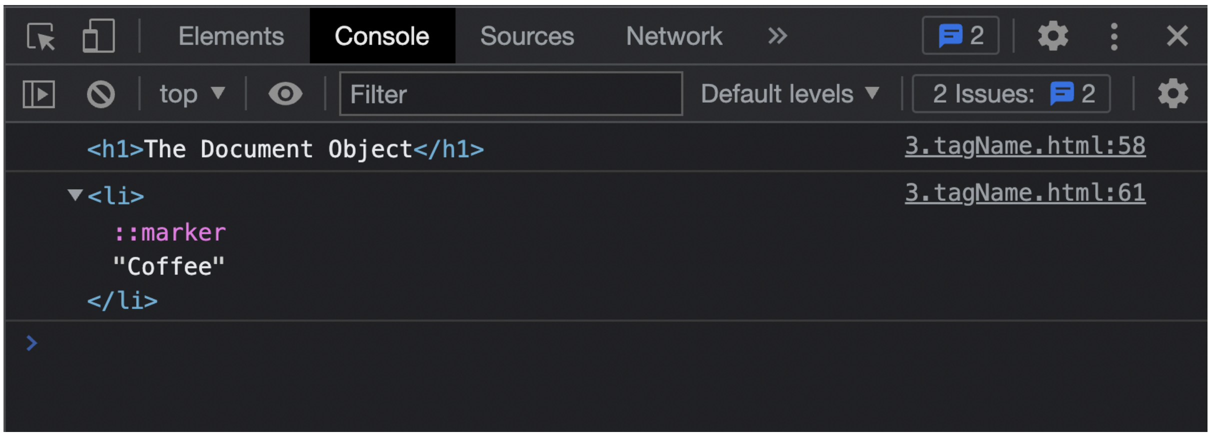
- The length Property returns the number of elements in the collection.

```
console.log(list.length) //2
console.log(h1.length) //2
```

- The elements in a collection can be accessed by index (starts at 0).
- If you want to access first list item from collection of list items, you can access as

```
<script>
  let h1 = document.getElementsByTagName("h1");
  console.log(h1[0])

  let list = document.getElementsByTagName("li");
  console.log(list[0]);
</script>
```

Output:

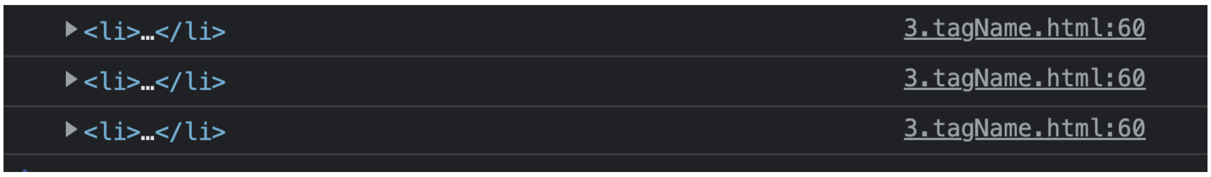


```
<h1>The Document Object</h1>          3.tagName.html:58
▼<li>                                 3.tagName.html:61
   ::marker
   "Coffee"
 </li>
>
```

- You can also use for loop to access all elements at once

```
for (let i = 0; i < list.length; i++) {
  console.log(list[i])
}
```

Output:



```
▶<li>…</li>                           3.tagName.html:60
▶<li>…</li>                           3.tagName.html:60
▶<li>…</li>                           3.tagName.html:60
```

Styling DOM Elements in JavaScript

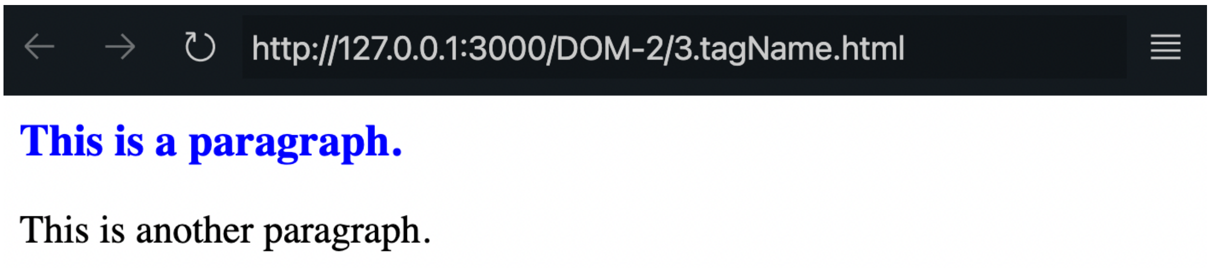
- You can also apply style on HTML elements to change the visual presentation of HTML documents dynamically using JavaScript.
- You can set almost all the styles for the elements like, fonts, colors, margins, borders, background images, text alignment, width and height, position, and so on.

Setting Inline Styles on Elements

- Inline styles are applied directly to the specific HTML element using the style attribute. In JavaScript the `style` property is used to get or set the inline style of an element.
- The following example will set the color and font properties of an element with `id="intro"`.

```
<html lang="en">
<head>
    <title>JS Set Inline Styles Demo</title>
</head>
<body>
    <p id="intro">This is a paragraph.</p>
    <p>This is another paragraph.</p>
</body>
</html>
<script>
    // Selecting element
    var elem = document.getElementById("intro");

    // Appling styles on element
    elem.style.color = "blue";
    elem.style.fontSize = "18px";
    elem.style.fontWeight = "bold";
</script>
```

Output:


http://127.0.0.1:3000/DOM-2/3.tagName.html

This is a paragraph.

This is another paragraph.

Live code : Codepen

Overview of all methods

Method	Syntax
getElementById	document.getElementById(”idname”)
getElementByClassName	document.getElementsByClassName(”classname”)
getElementByTagName	document.getElementsByTagName(”TagName”)

- So one drawback of above methods is, we are using different syntax for different methods.
- So now we will see a method where we can access/catch element by its
  - tagName
  - className
  - id
- These methods are known as `querySelector` and `querySelectorAll`

querySelector():

- The `querySelector()` method returns the `first` element that matches a CSS selector.

Syntax

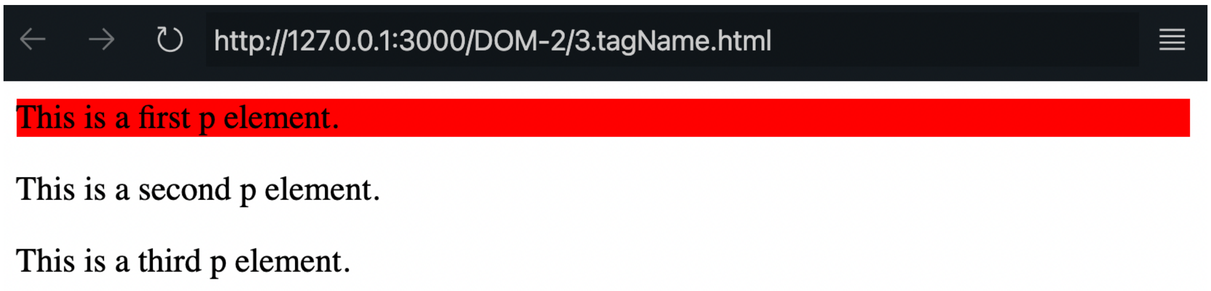
```
document.querySelector(CSS selectors)
```

- Here selectors can be any selector (id,class,tag,universal)

Example

```
<html>
  <body>
    <p>This is a first p element.</p>
    <p>This is a second p element.</p>
    <p>This is a third p element.</p>
  </body>
</html>
<script>
  document.querySelector("p").style.backgroundColor = "red";
</script>
```

Output:


http://127.0.0.1:3000/DOM-2/3.tagName.html

This is a first p element.

This is a second p element.

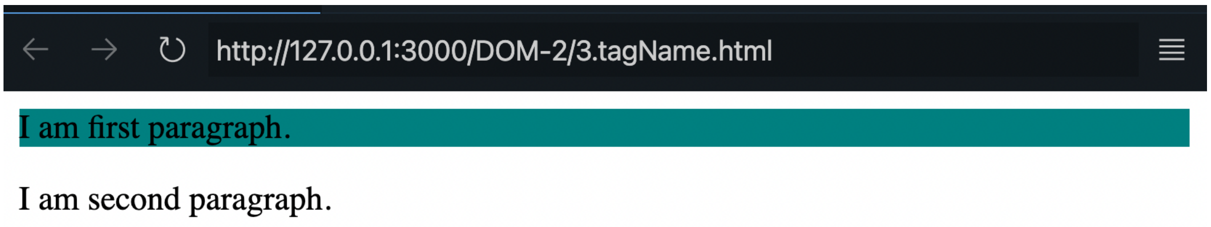
This is a third p element.

Live code : Codepen

- In the above example, `querySelector` is making background color red to only first element that matches
- Let's take one more example

```
<html>
  <body>
    <p class="example">I am first paragraph.</p>
    <p class="example">I am second paragraph.</p>
  </body>
</html>
<script>
  document.querySelector(".example").style.backgroundColor = "teal";
</script>
```

Output


http://127.0.0.1:3000/DOM-2/3.tagName.html

I am first paragraph.

I am second paragraph.

Live code : Codepen

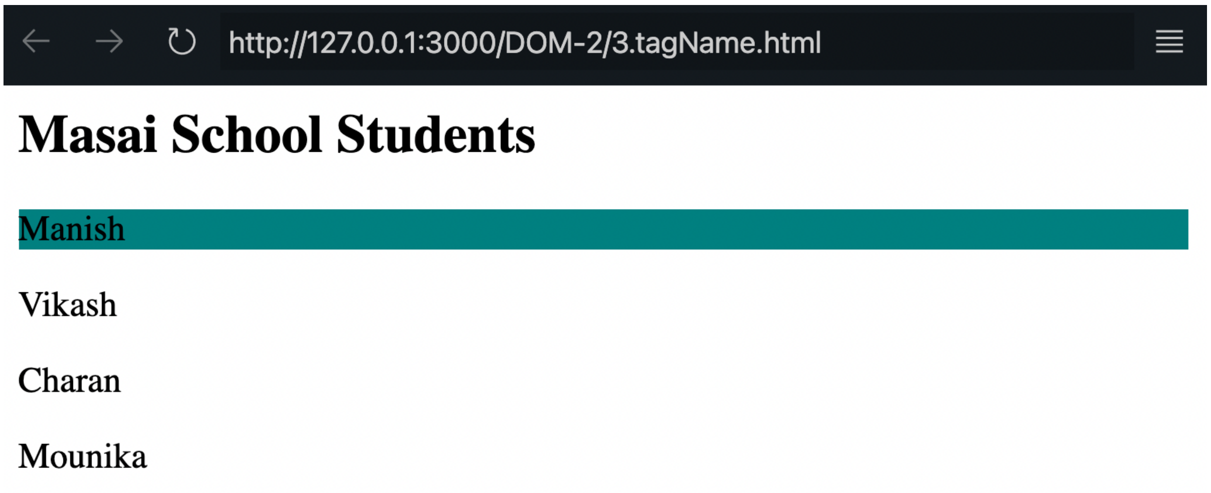
- In the above example, `querySelector` is making background color teal to only first element that matches query class example
- We can also use combinators here, but even with combinators it will only return first element

Example:

```
<html>
  <body>
    <h2>Masai School Students</h2>
    <p>Manish</p>
    <p>Vikash</p>
    <p>Charan</p>
    <p>Mounika</p>
  </body>
</html>
<script>
  document.querySelector("h2~p").style.backgroundColor = "teal";
</script>
```

- Basically `h2~p` should return all p's, but `querySelector` will return only first p, in this case Manish

Output:



Masai School Students

Manish

Vikash

Charan

Mounika

Live code : Codepen

- To return **all** matches (not only the first), use the `querySelectorAll()` instead.

querySelectorAll():

- The `querySelectorAll()` method returns all elements that matches a CSS selector(s).
- The `querySelectorAll()` method returns a NodeList.

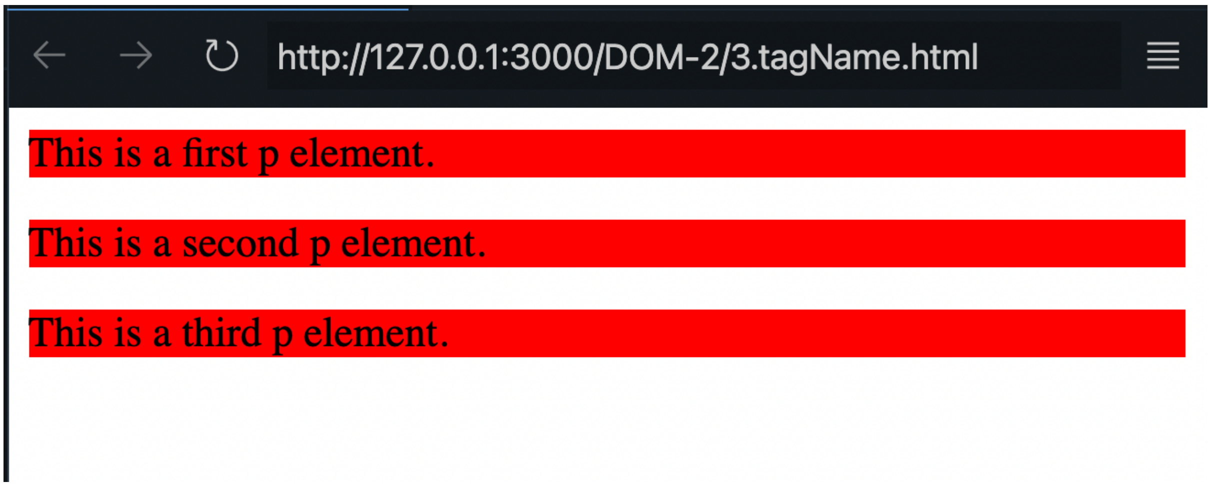
Syntax

```
document.querySelectorAll(CSS selectors)
```

- Here selectors can be any selector (id,class,tag,universal)
- Let's take the same examples as querySelector

```
<html>
  <body>
    <p>This is a first p element.</p>
    <p>This is a second p element.</p>
    <p>This is a third p element.</p>
  </body>
</html>
<script>
  // This won't work because querySelectorAll() method returns a NodeList.
  // document.querySelectorAll("p").style.backgroundColor = "red";
  // So we need to use for loop to select all elements
  var list = document.querySelectorAll("p");
  for (var i = 0; i < list.length; i++) {
    list[i].style.backgroundColor = "red";
  }
</script>
```

Output



http://127.0.0.1:3000/DOM-2/3.tagName.html

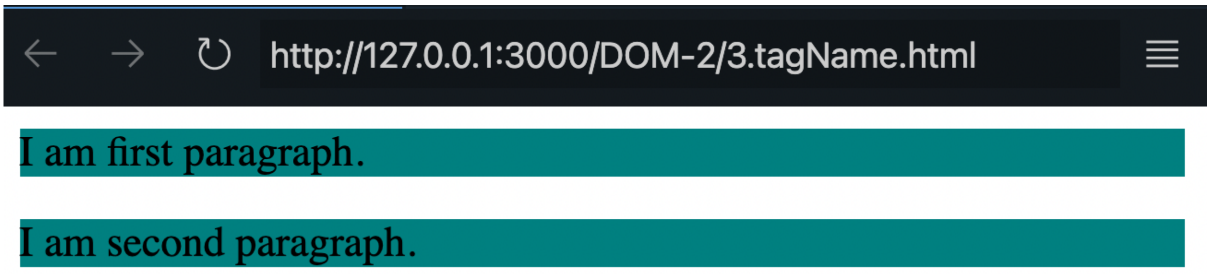
```
This is a first p element.

This is a second p element.

This is a third p element.
```

Live code : Codepen

Example 2

```
<html>
  <body>
    <p class="example">I am first paragraph.</p>
    <p class="example">I am second paragraph.</p>
  </body>
</html>
<script>
  var list = document.querySelectorAll(".example");
  for (var i = 0; i < list.length; i++) {
    list[i].style.backgroundColor = "teal";
  }
</script>
```

Output:


http://127.0.0.1:3000/DOM-2/3.tagName.html

I am first paragraph.

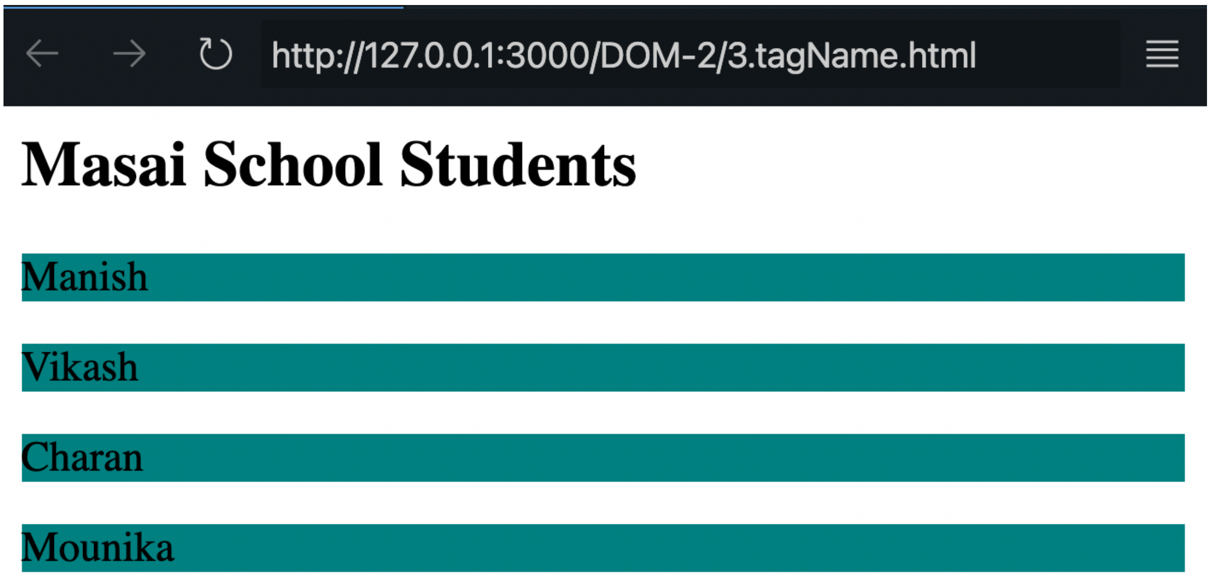
I am second paragraph.

Live code : Codepen

Example 3:

```
<html>
  <body>
    <h2>Masai School Students</h2>
    <p>Manish</p>
    <p>Vikash</p>
    <p>Charan</p>
    <p>Mounika</p>
  </body>
</html>
<script>
    var list = document.querySelectorAll("h2~p");
    for (var i = 0; i < list.length; i++) {
    list[i].style.backgroundColor = "teal";
  }

</script>
```

Output:


http://127.0.0.1:3000/DOM-2/3.tagName.html

Masai School Students

Manish

Vikash

Charan

Mounika

Live code : Codepen

addEventListener()

- Untill now, we are using onClick event listener, but we are using it inline to HTML element.

```
<button onClick="likeMe()">Like👍</button>
```

- But this method of adding event is not recommended, since we are writing JS inside HTML tags.
- So, to improve this, we have a method called as `addEventListener()`
- The `addEventListener()` method attaches an event handler to a document, same functionality as onClick,it's just a syntax change.

Syntax

```
document.addEventListener(event, function)
```

Parameters

Parameter	Description
event	The event name. Do not use the "on" prefix. Use "click" instead of "onclick".
function	The function to run when the event occurs. When the event occurs, an event object is passed to the function as the first parameter. The type of the event object depends on the specified event. For example, the "click" event belongs to the MouseEvent object.

- So, let's try to rewrite DOM-1 example using `addEventListener`

```
<html>
<head>
      <title>OnClick</title>
</head>
<body>
      <button>Like👍</button>
</body>
</html>
```

- To add `addEventListener` we need to catch the element to which we want to add and then add an event to that element

```
<script>
      document.querySelector("button").addEventListener("click",likeMe)

      function likeMe() {
            console.log("Someone liked me")
      }
</script>
```

- Here is a list of some common HTML events:

Event	Description
change	An HTML element has been changed
click	The user clicks an HTML element
mouseover	The user moves the mouse over an HTML element
mouseout	The user moves the mouse away from an HTML element
keydown	The user pushes a keyboard key
load	The browser has finished loading the page

- For more events - Click here

Event Object:

- When the event occurs, an event object is passed to the function as the first parameter. The type of the event object depends on the specified event. For example, the "click" event belongs to the MouseEvent object.

What is event.target in JavaScript?

- `target`, is a property of an event which is a reference to the element upon which the event was fired. Just as 'target' means 'aiming at something', it's used to 'aim' at that particular element.
- This property gives us access to the properties of that element.
- Since the target property has given us access to the element, we could then read some of the properties (which are the attributes) and also display them somewhere else.

Syntax

The `target` property can only be obtained if the event of an `element` is being listened to.

```
element.addEventListener("click",myFunction)

function myFunction(){
  console.log(event) // //event.target is now accessible
}
```

Importance of event.target

It is necessary to have the `target` property when an event is fired. We can do the following with the `target` property of the event.

- Get the `element` that fired the event.
- Access the properties of the `element`.
- Modify some properties of the `element`, such as the CSS, the attributes, etc.

```
<!DOCTYPE html>
<html>
  <body>
    <p>
      Click on button in this document to find out which element triggered the
      onclick event.
    </p>

    <button>This is a button</button>

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

    <script>
      document.querySelector("button").addEventListener("click", myFunction);
      function myFunction() {
        var x = event.target;
        document.getElementById("demo").innerHTML =
          "Triggered by a " + x.tagName + " element";
      }
    </script>
  </body>
</html>
```

Live code : Codepen