

CSE-465

Web Programming

JavaScript DOM Manipulation

[Reference](#)
[More Details](#)

Md. Saidul Hoque Anik
anik@cse.uiu.ac.bd

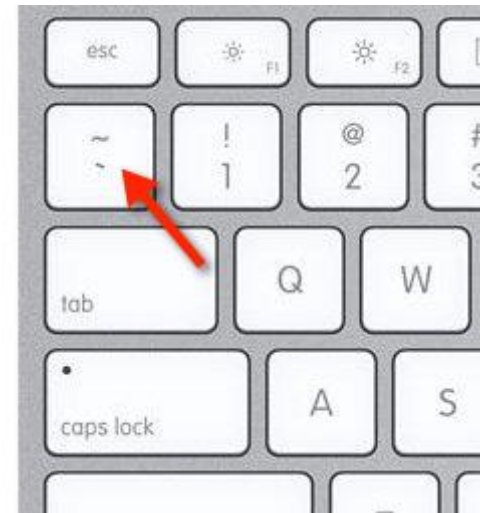
Template Literal

String literals allowing embedded expressions

```
let a = 10, b = 20;
```

```
console.log(`The value of a is ${a}, and value  
of b is ${b}`);
```

To write a template literal, you must enclose it with back quote (```). The variable/expression has to be written inside `${ }` to be parsed.



Objects

Putting multiple information in a single variable using objects.

//creating a object

```
let person1 = {name: "Mr. A", phone: 1234, address: "Dhaka"};
```

Property



Value

Comma between
multiple property-
value pair

Separated
by colon

Enclosed by
Braces { }

Objects

Putting multiple information in a single variable using objects.

```
//creating a object
```

```
let person1 = {name: "Mr. A", phone: 1234, address: "Dhaka"};
```

```
//you can also put string as attribute
```

```
let person2 = {'name': "Mr. B", 'phone': 1234, 'address': "Dhaka"};
```

```
//output
```

```
console.log(person1.name);
```

```
//accessing in array-like manner
```

```
console.log(person1['name']);
```

```
//adding new property
```

```
person1.blood_group = 'A+';
```

for .. in

To access all the properties of an object

```
let person1 = {name: "Mr. A", phone: 1234, address: "Dhaka"};

for (let prop in person1)
    console.log(person1[prop]);
```

Array of Objects

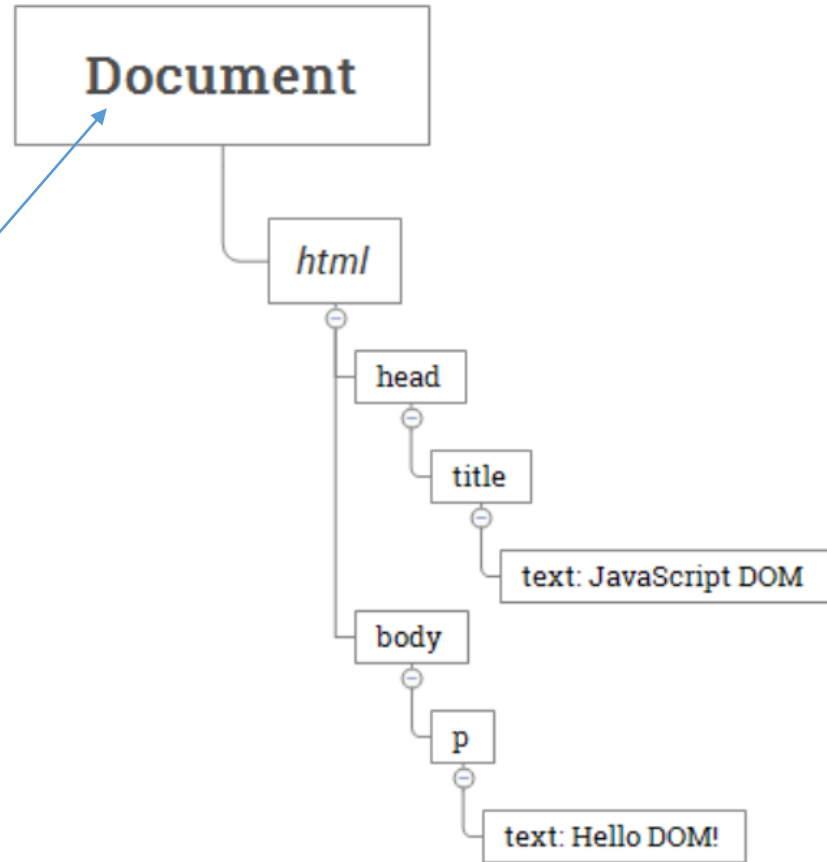
```
let people =  
  [  
    {name: "Mr. A", age: 20},  
    {name: "Mr. B", age: 25},  
    {name: "Mr. C", age: 23}  
  ];  
  
//Printing values:  
for (let i of people)  
  for (let j in i)  
    console.log(`Name=${j.name}, Age=${j.age}`);  
  
//Inserting value:  
people.push({name: "Mr. D", age: 24});  
  
//How do you find the age of Mr. B? Hint: use filter()
```

Array of Objects

```
let people =  
  [  
    {name: "Mr. A", age: 20},  
    {name: "Mr. B", age: 25},  
    {name: "Mr. C", age: 23}  
  ];  
  
//Printing values:  
for (let i of people)  
  for (let j in i)  
    console.log(`Name=${j.name}, Age=${j.age}`);  
  
//Inserting value:  
people.push({name: "Mr. D", age: 24});  
  
//Find the age of Mr. B  
console.log(people.filter(arg => arg.name == "Mr. A")[0].age);
```

Document as hierarchy of nodes

```
<html>
  <head>
    <title>JavaScript DOM</title>
  </head>
  <body>
    <p>Hello DOM!</p>
  </body>
</html>
```



This is the document object.
(Thus the name Document Object
Model, or **DOM**)

DOM Overview

- The Document Object Model (DOM) is an application programming interface (API) for manipulating HTML and XML documents.
- The DOM represents a document as a tree of nodes. It provides API that allows you to add, remove, and modify parts of the document effectively.
- Note that the DOM is cross-platform and language-independent ways of manipulating HTML and XML documents.

Selecting a DOM Element

- `document.querySelector("span")`
Get the first span element in the document
- `document.querySelectorAll("span");`
Get all span elements in the document

Read More:

https://www.w3schools.com/jsref/dom_obj_document.asp

What can you do with a DOM element inside JavaScript?

After getting a reference of an HTML element in a variable, you can-

- Change the inside text.
- Add/Change the attributes.
- Add/Change the CSS styles.
- Add/Change the class.
- Add/remove event-listener.
- Jump to any other node in the DOM Tree (sibling/parent/child).
 - Create/delete/modify any node in the DOM Tree
- Do a `querySelector()/querySelectorAll()` on the children node.

Basically everything.

A DOM Element

```
<p class="facts" style="text-align: center;">  
    This is a <strong>paragraph</strong> tag.  
</p>
```



The DOM element

A DOM Element

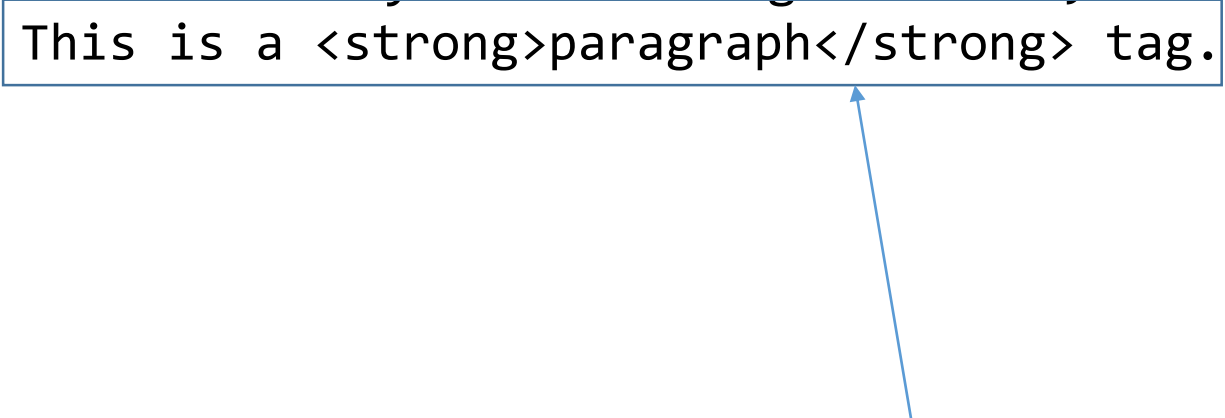
```
<p class="facts" style="text-align: center;">  
    This is a <strong>paragraph</strong> tag.  
</p>
```



The attributes

A DOM Element

```
<p class="facts" style="text-align: center;">  
  This is a <strong>paragraph</strong> tag.  
</p>
```



The innerHTML

innerHTML

Returns the text, including all spacing and inner element tags.

```
console.log(elem.innerHTML);
```

```
elem.innerHTML = "This is the new message"
```

Read More: https://www.w3schools.com/jsref/prop_html_innerhtml.asp

querySelectorAll() example

Select all spans in the document and print their contents one by one

```
let spans = document.querySelectorAll('span');  
for (let i = 0; i < spans.length; i++)  
    console.log(`The content of ${i + 1}th span is  
    ${spans[i].innerHTML}`);
```

Read More:

https://www.w3schools.com/jsref/met_document_queryselectorall.asp

Modifying attributes

Method	Description
<code>getAttributeNames()</code>	Returns the attribute names of the element as an Array of strings
<code>getAttribute()</code>	Returns the specified attribute value of an element node
<code>setAttribute()</code>	Sets or changes the specified attribute, to the specified value
<code>hasAttribute()</code>	Returns true if an element has the specified attribute, otherwise false
<code>removeAttribute()</code>	Removes a specified attribute from an element

Read More: https://www.w3schools.com/jsref/prop_element_classlist.asp

Attribute Methods Example

- `console.log(btn.getAttributeNames());`
- `btn.setAttribute("class", "democlass");`
- `let btnclassname = btn.getAttribute("class");`

[hasAttribute\(\) Method](#)

[removeAttribute\(\) Method](#)

Read More: https://www.w3schools.com/jsref/met_element_setattribute.asp

Specialized properties

Since style and classes are more frequently used, DOM element has specific APIs for them.

- `elem.style`
- `elem.classList`

Style

```
btn.style.color = "red";
```

Return style properties:

```
element.style.property
```

Set style properties:

```
element.style.property = value
```

See all the CSS styles:

https://www.w3schools.com/jsref/dom_obj_style.asp

Classlist Methods

Example: `btn.classList.add("blue");`

Method	Description
<code>add(class1, class2, ...)</code>	Adds one or more class names to an element.
<code>contains(class)</code>	Returns a Boolean value, indicating whether an element has the specified class name.
<code>item(index)</code>	Returns the class name with a specified index number from an element. Index starts at 0.
<code>remove(class1, class2, ...)</code>	Removes one or more class names from an element.
<code>toggle(class, true false)</code>	<p>Toggles between a class name for an element.</p> <p>The first parameter removes the specified class from an element, and returns false. If the class does not exist, it is added to the element, and the return value is true.</p> <p>The optional second parameter is a Boolean value that forces the class to be added or removed, regardless of whether or not it already existed.</p> <p>Note: The second parameter is not supported in Internet Explorer or Opera 12 and earlier.</p>

Read More: https://www.w3schools.com/jsref/prop_element_classlist.asp

Other properties & methods

Property / Method	Description
focus()	Gives focus to an element
innerHTML	Sets or returns the content of an element
outerHTML	Sets or returns the content of an element (including the start tag and the end tag)
tagName	Returns the tag name of an element
<code>toString()</code>	Converts an element to a string

Read More: https://www.w3schools.com/jsref/dom_obj_all.asp

DOM Traversal

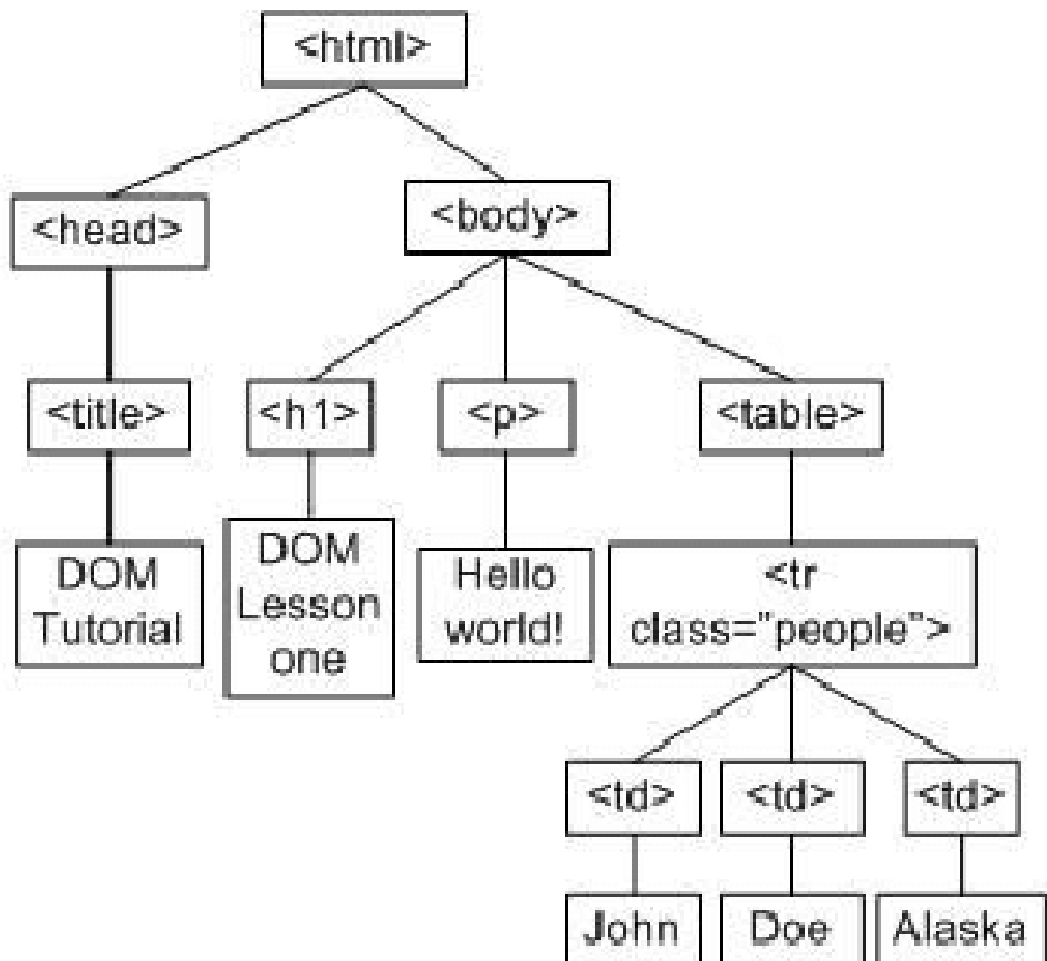
After selecting an element, we can use-

- parentNode : to get the parent element
- firstElementChild : to get the first child element
- lastElementChild : to get the last child element
- nextElementSibling : to get the next sibling element
- previousElementSibling : to get the previous sibling element
- children : get a collection of an element's child elements

DOM Tree

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
    <table>
      <tr class="people">
        <td>John</td>
        <td>Doe</td>
        <td>Alaska</td>
      </tr>
    </table>
  </body>
</html>
```

test.html



DOM Tree

Task

```
<table>
<thead>
  <th>Ser</th><th>Name</th><th>Age</th><th>Salary</th>
</thead>
<tbody>
  <tr> <td>1</td><td>Rob</td><td>22</td><td>10000</td> </tr>
  <tr> <td>2</td><td>Peterson</td><td>20</td><td>20000</td> </tr>
  <tr> <td>3</td><td>Jack</td><td>26</td><td>15000</td> </tr>
</tbody>
</table>
```

From the table above-

- Find the sum of all salaries
- Find the largest age
 - Find the person having the largest age
- Find the name of the person with the longest name
 - Find the salary of the person with the longest name

Anonymous Function in EventListener

```
btn.addEventListener('click', ()=>{  
    //your code;  
});
```

Anonymous Function in EventListener

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Some Web Page</title>
  </head>
  <body>
    <h1>Some Web Page</h1>
    <button id="btn1">Set name</button>
    <script>
      let btn = document.querySelector('#btn1');
      btn.addEventListener('click', () => {
        console.log("You clicked btn1");
      });
    </script>
  </body>
</html>
```

Event Information

```
btn.addEventListener('click', (e) => {  
    console.log(e);  
});
```

The event details are passed as the first argument in the anonymous function.

See all event properties and methods:

https://www.w3schools.com/jsref/dom_obj_event.asp

Event Information

```
btn.addEventListener('click', (e) => {  
    console.log(e.ctrlKey);  
});
```

If Ctrl is kept pressed during the click, the output will be true

See all event properties and methods:

https://www.w3schools.com/jsref/dom_obj_event.asp

Event Information

```
btn.addEventListener('click', (e) => {  
    console.log(e.target.tagName); //BUTTON  
});
```

- e.target gives the element on which the event occurred.
- If a button is clicked, e.target will provide reference to the button element.

See all event properties and methods:

https://www.w3schools.com/jsref/dom_obj_event.asp

Practice

Task:

To-do:

1. Task 1

2. Task 2

3. Task 3

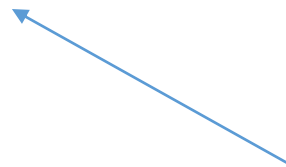
Home Practice

Product: Price:

Grocery List:

Product	Price	Modify
Books	100	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Pencil	50	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Sharpner	20	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Total Cost: 170 Taka

 This should be updated automatically

Hint: Create object and push in array

MDN Example

Number guessing game

We have selected a random number between 1 and 100. See if you can guess it in 10 turns or fewer. We'll tell you if your guess was too high or too low.

Enter a guess:

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash

DOM Manipulation

After selecting an element, we can use-

- `cloneNode(true)` : to deep clone/copy an element
- `appendChild()` : to add an element inside
- `removeChild()` : to remove an element

We can also directly create children using `innerHTML`.

Other Concepts

Comments

The syntax of **comments** is the same as in C++ and in many other languages:

```
1  // a one line comment
2
3  /* this is a longer,
4     * multi-line comment
5     */
6
7  /* You can't, however, /* nest comments */ SyntaxError */
```

Comments behave like whitespace, and are discarded during script execution.

Variable Declaration

JavaScript has three kinds of variable declarations.

var

Declares a variable, optionally initializing it to a value.

let

Declares a block-scoped, local variable, optionally initializing it to a value.

const

Declares a block-scoped, read-only named constant.

Variable Declaration : var

The **var statement** declares a function-scoped or globally-scoped variable, optionally initializing it to a value.



JavaScript Demo: Statement - Var

```
1 var x = 1;
2
3 if (x === 1) {
4   var x = 2;
5
6   console.log(x);
7   // expected output: 2
8 }
9
10 console.log(x);
11 // expected output: 2
12
```

Variable Declaration : let

The **let** statement declares a block-scoped local variable, optionally initializing it to a value.



JavaScript Demo: Statement - Let

```
1 let x = 1;
2
3 if (x === 1) {
4   let x = 2;
5
6   console.log(x);
7   // expected output: 2
8 }
9
10 console.log(x);
11 // expected output: 1
12
```

Variable Declaration : const

Constants are block-scoped, much like variables declared using the `let` keyword. The value of a constant can't be changed through reassignment, and it can't be redeclared.



JavaScript Demo: Statement - Const

```
1 const number = 42;
2
3 try {
4   number = 99;
5 } catch (err) {
6   console.log(err);
7   // expected output: TypeError: invalid assignment to const `number`
8   // Note - error messages will vary depending on browser
9 }
10
11 console.log(number);
12 // expected output: 42
```


Variable Hoisting

Because variable declarations (and declarations in general) are processed before any code is executed, declaring a variable anywhere in the code is equivalent to declaring it at the top. This also means that a variable can appear to be used before it's declared. This behavior is called "hoisting", as it appears that the variable declaration is moved to the top of the function or global code.

```
1 | bla = 2;  
2 | var bla;  
3 |  
4 | // ...is implicitly understood as:  
5 |  
6 | var bla;  
7 | bla = 2;
```

For that reason, it is recommended to always declare variables at the top of their scope (the top of global code and the top of function code) so it's clear which variables are function scoped (local) and which are resolved on the scope chain.

The difference between var and let

At this point you may be thinking "why do we need two keywords for defining variables?? Why have `var` and `let`?".

The reasons are somewhat historical. Back when JavaScript was first created, there was only `var`. This works basically fine in most cases, but it has some issues in the way it works — its design can sometimes be confusing or downright annoying. So, `let` was created in modern versions of JavaScript, a new keyword for creating variables that works somewhat differently to `var`, fixing its issues in the process.

The difference between var and let

For a start, if you write a multiline JavaScript program that declares and initializes a variable, you can actually declare a variable with `var` after you initialize it and it will still work. For example:

```
1 myName = 'Chris';  
2  
3 function logName() {  
4   console.log(myName);  
5 }  
6  
7 logName();  
8  
9 var myName;
```

Hoisting no longer works with `let`. If we changed `var` to `let` in the above example, it would fail with an error. This is a good thing — declaring a variable after you initialize it results in confusing, harder to understand code.

The difference between == and ===

The difference between == and === is that: == converts the variable values to the same type before performing comparison. This is called type coercion. === does not do any type conversion (coercion) and returns true only if both values and types are identical for the two variables being compared.

Example:

```
console.log(2 == '2');    //true, although left on is Number  
                           //and right one is string
```

```
console.log(2 === '2');   //false, checks equality between  
                           //same type of variable
```

Read more:

<https://codeahoy.com/javascript/2019/10/12/==-vs-===-in-javascript/>