JavaScript and DOM

Topic 4

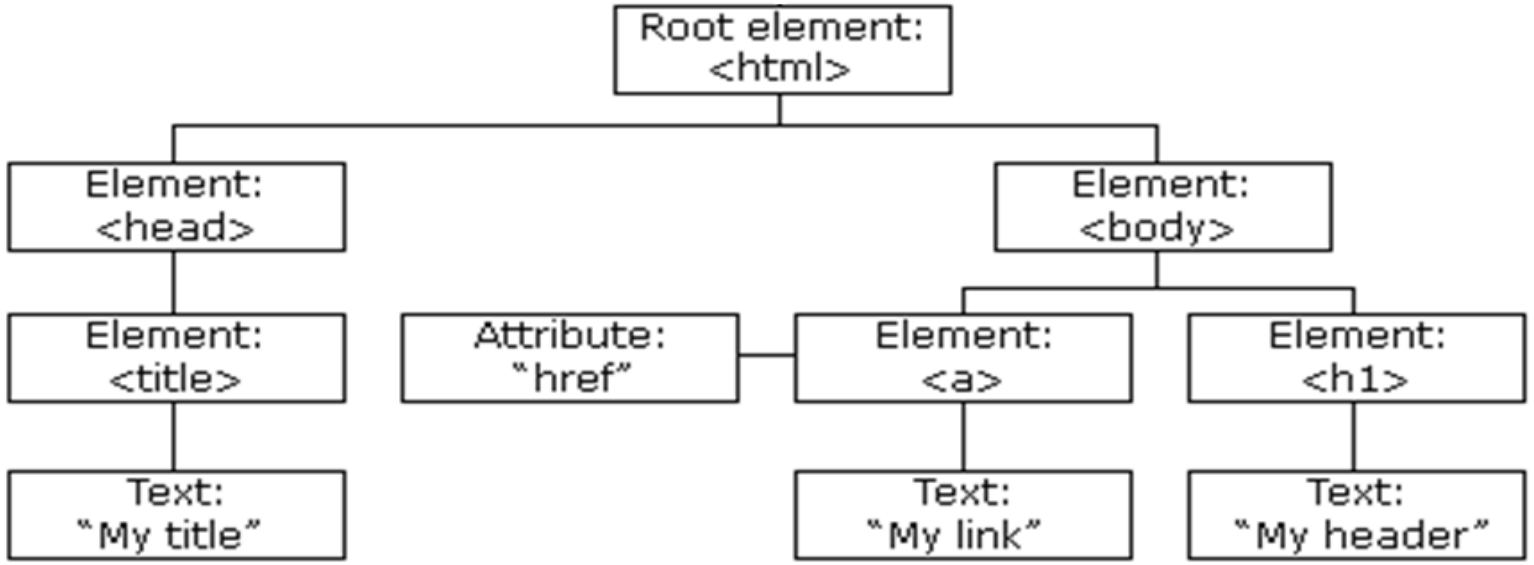
Document Object Model (**DOM**)

- The DOM defines a standard for accessing documents.
- Elements on a HTML are accessible in JavaScript through the DOM.
- Each tag in a html corresponds to a JavaScript DOM object.
- The HTML DOM is a standard that defines how JavaScript can get, change, add, or delete HTML elements.

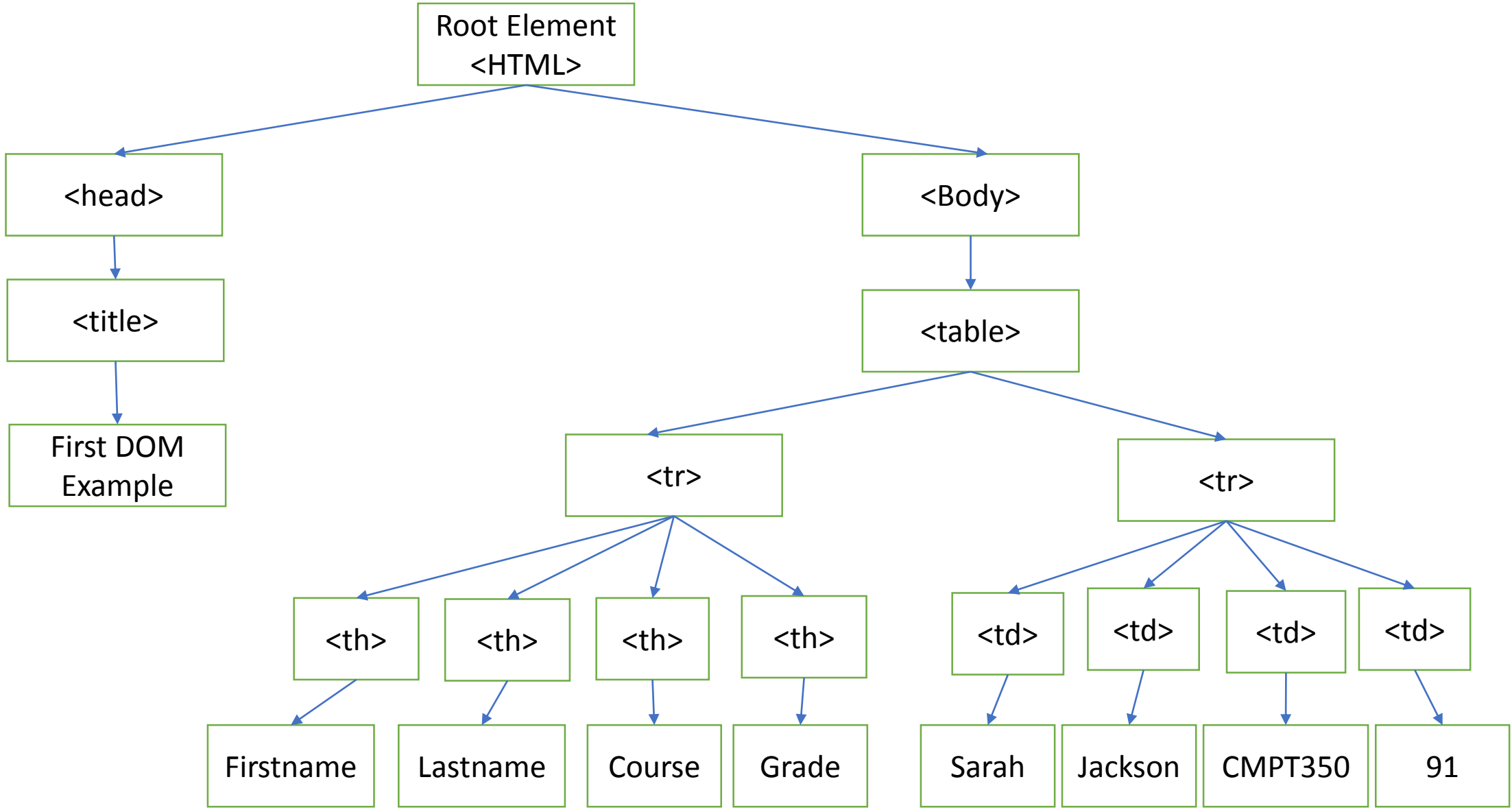
DOM

- In DOM The Objects are organized in a hierarchy.
- The DOM is tree of node objects corresponding to the HTML elements on a page.

The HTML DOM Tree of Objects



```html
<!DOCTYPE html>
<html>
 <head>
 <title>First DOM Example</title>
 </head>
 <body>
        <table>
                <tr>
                  <th>Firstname</th>
                  <th>Lastname</th>
                  <th>Course</th>
                  <th>Grade</th>
                </tr>
                <tr>
                  <td>Sarah</td>
                  <td>Jackson</td>
                  <td>CMPT350</td>
                  <td>91</td>
                </tr>
 </body>
</html>
```


Root Element
<HTML>
<head>
<Body>
<title>
<table>
First DOM
Example
<tr>
<tr>
<th>
<th>
<th>
<th>
<td>
<td>
<td>
<td>
Firstname
Lastname
Course
Grade
Sarah
Jackson
CMPT350
91

Node relationships in DOM tree

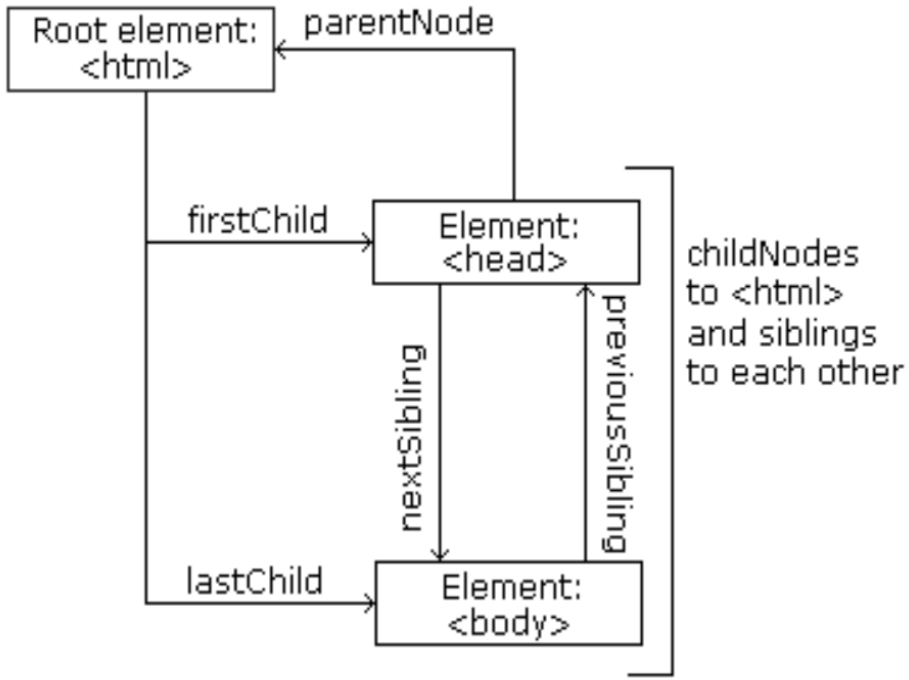
- The nodes have a hierarchical relationship to each other
- There are three relationships: parent, child, sibilings
- In a node tree, the top node is called the root (or root node)
- Every node has exactly one parent, except the root (which has no parent)
- A node can have a number of children
- Siblings are nodes with the same parent

Traversing the DOM tree: node relationship

- firstChild, lastChild: start/end of this node's list of children
- childNodes: array of all this node's children
- nextSibling, previousSibling: neighboring nodes with the same parent
- parentNode: the element that contains this node

DOM tree traversal

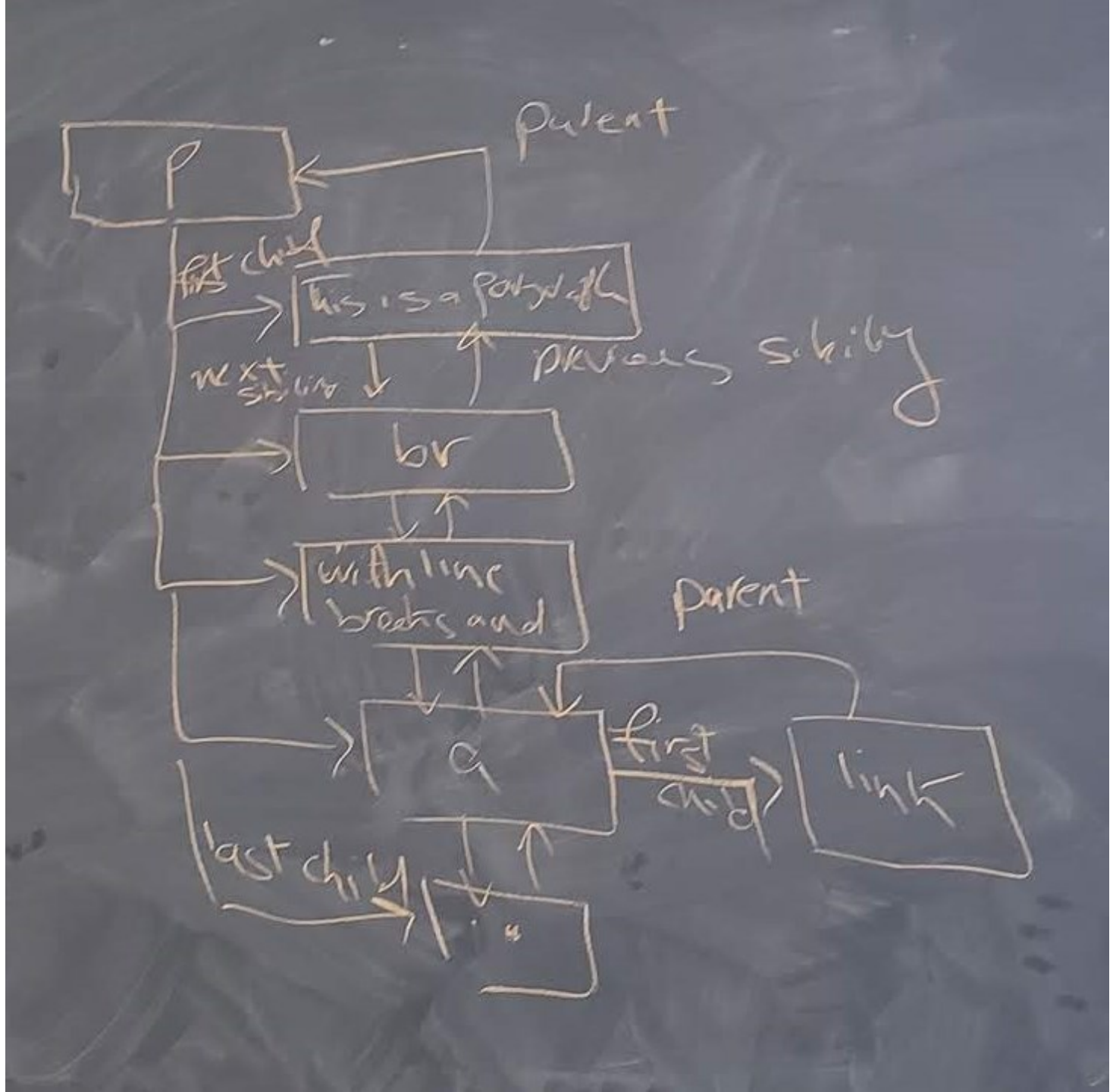
```
<!DOCTYPE html>
<html>
        <head>
        </head>
        <body>
        </body>
</html>
```



https://www.w3schools.com/js/js_htmldom_navigation.asp

DOM tree traversal example 2

This is a paragraph
with line breaks and link. `


parent
p
first child
this is a paragraph
next sibling
previous sibling
br
with line breaks and
parent
a
first child
link
last child

DOM methods and properties

- We can access and change the contents of document by its methods.
- HTML DOM methods are **actions** you can perform (on HTML Elements).
- HTML DOM properties are **values** (of HTML Elements) that you can set or change.
- In the DOM, all HTML elements are defined as **objects**.

Properties and Methods

- The programming interface is the properties and methods of each object.

```
document.getElementById("p1").innerHTML = "Hello World!";
```

innerHTML is an example of DOM property

getElementById is an example of DOM method

Access element

- The document object represents the web page.
- To access any element in an HTML page, you should access the document object.

DOM access element methods

- document.getElementById(*id*)

  Find an element by element id
- document.getElementsByTagName(*name*)

  Find elements by tag name
- document.getElementsByClassName(*name*)

  Find elements by class name

getElementById

- If the element is found, the method will return the element as an object (in myElement). If the element is not found, the method will contain null.
- access/modify the attributes of a DOM object with dot notation objectName.attributeName

```
<h1 id="id01">Old Heading</h1>
```

Old Heading

Add JS

```
var element = document.getElementById("id01");
element.innerHTML = "New Heading";
element.style.color= "green";
```

New Heading

innerHTML

- HTML DOM property
- It sets or returns the HTML content (inner HTML) of an element.

DOM Style property

- The style property returns a CSS StyleDeclaration object, which represents an element's style attribute.
- *Syntax: element*.style.*property*
- Set style properties: *element*.style.*property* = *value*

```
element.style.backgroundColor = "red";
element.style.borderBottom = "3px solid blue";
```

getElementsByTagName()

- This method returns a collection of an element's child elements with the specified **tag name**, as a **NodeList** object.
- An HTMLCollection object is an array-like list (collection) of HTML elements.

```
<div id="myDIV">
  <p>First p element in div.</p>
  <p>Second p element in div.</p>
  <p>Third p element in div.</p>
</div>
```

Add JS

```
var x = document.getElementById("myDIV");
 var y = x.getElementsByTagName("P");
 var i;
 for (i = 0; i < y.length; i++) {
                y[i].style.backgroundColor = "yellow";
                y[i].style.color = "red";
 }
```

First p element in div.

Second p element in div.

Third p element in div.

First p element in div.

Second p element in div.

Third p element in div.

getElementsByClassName()

- This method returns a **collection** of an element's child elements with the specified **class name**, as a **NodeList object.**

```
<div id="myDIV">
  <p class="child">First p element with class="child" in a div (index 0).</p>
  <p class="child">Second p element with class="child" in a div (index 1).</p>
  <p class="child">Third p element with class="child" in a div (index 2).</p>
</div>
```

First p element with class="child" in a div (index 0).

Second p element with class="child" in a div (index 1).

Third p element with class="child" in a div (index 2).

Add JS

```
var x = document.getElementById("myDIV");
  x.getElementsByClassName("child")[1].style.backgroundColor = "green";
```

First p element with class="child" in a div (index 0).

Second p element with class="child" in a div (index 1).

Third p element with class="child" in a div (index 2).

Two more methods to access HTML elements

- You can find elements with rules that cannot be expressed with getElementById and getElementsByClassName
- document.querySelector('css selector');

  Returns the first element that matches the given CSS selector(s) in the document

  document.querySelector('#button');

  document.querySelector("p.example");

  document.querySelector("div > p");
- Document.querySelectorAll('css selector');

  Returns all elements that match the given CSS selector.

  document.querySelectorAll('.classA, .classB');

Adding and Deleting Elements

Method	**Description**
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

createElement(*element*)

Add a new element to the HTML DOM

1. Create the element (element node)
2. Append it to an existing element using appendChild() method

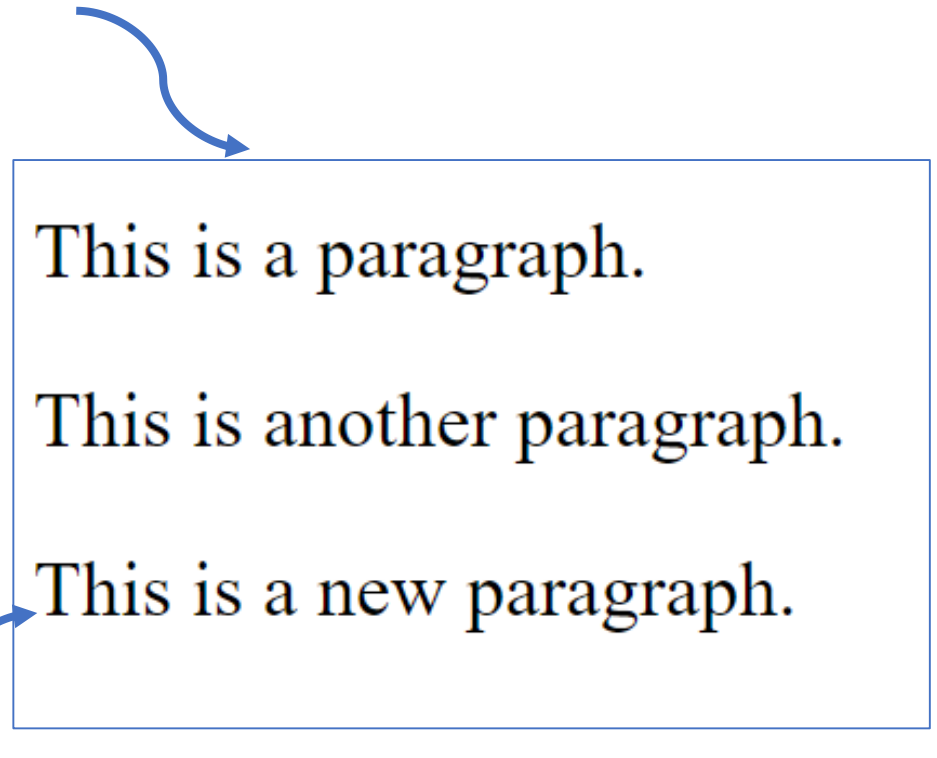
```
<body>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is a new paragraph.");
para.appendChild(node);
var element = document.getElementById("div1");
element.appendChild(para);
</script>

</body>
```

Webpage Output



Appended at the end.

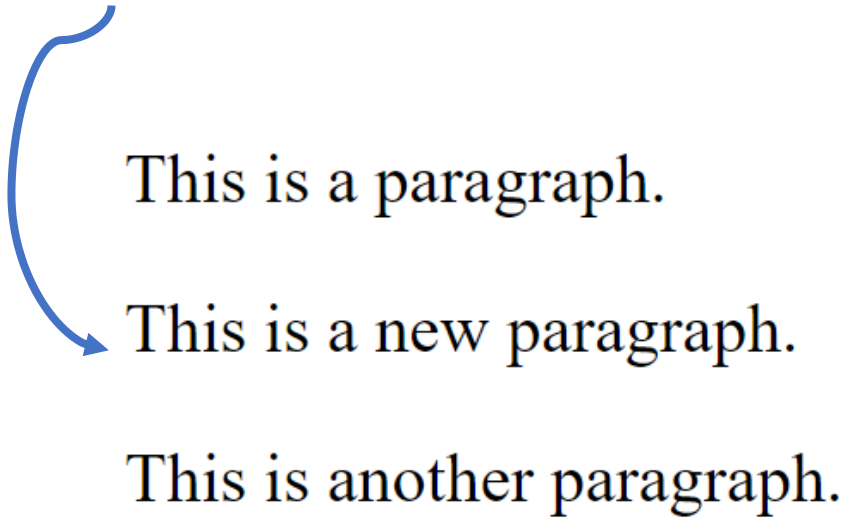
```
<body>

<div id="div1">
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>

<script>
var para = document.createElement("p");
var node = document.createTextNode("This is a new
paragraph.");
para.appendChild(node);
var element = document.getElementById("div1");
var y = element.getElementsByTagName("P");
y[0].appendChild(para);
</script>

</body>
```

Appended after the first child



removeChild(*element*)

- Remove an HTML element.

```
<body>
<div>
<p id="p1">This is a paragraph.</p>
<p id="p2">This is another paragraph.</p>
</div>


<script>
  var element = document.getElementById("p1");
  element.remove();
</script>
</body>
```

Webpage Output

This is another paragraph.

replaceChild

- Replace an element to the HTML DOM
- replaceChild(newChild, oldChild);

```
<body>

<div id="div1">
<p id="p1">This is a paragraph.</p>
</div>

<script>
var parent = document.getElementById("div1");
var child = document.getElementById("p1");
var para = document.createElement("p");
var node = document.createTextNode("This is new.");
para.appendChild(node);
parent.replaceChild(para,child);
</script>

</body>
```

Webpage Output

This is new.

Changing HTML Elements

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element
Method	Description
element.setAttribute(attribute, value)	Change the attribute value of an HTML element

Changing the Value of an Attribute

Change width attribute of an image

```
<body>

<img id="image" src="landscape.jpg" width="160" height="120">

<script>
document.getElementById("image").width = "720";
</script>

</body>
```

setAttribute()

```
<head>
<style>
.democlass {
  color: red;
}
</style>
</head>
<body>
<h1>Hello World!</h1>
<script>
  document.getElementsByTagName("H1")[0].setAttribute("class",
"democlass");
</script>
</body>
```

Webpage Output

Hello World!

document.write()

- Write directly to the HTML output stream.

```
<!DOCTYPE html>
<html>
<body>

<script>
document.write("this is directly written by JavaScript");
</script>

</body>
</html>
```

Webpage Output

this is directly written by JavaScript

DOM node properties

- Navigation between nodes
- DOM root nodes
- nodeName
- nodeValue
- nodeType

Navigation between nodes

parentNode

childNodes[nodenumber]

firstChild

lastChild

nextSibling

previousSibling

document.getElementById(“div1").firstChild.nodeValue;

document.getElementById(“div1").childNodes[1].nodeValue;

nodeValue

- nodeValue for element nodes is null
- nodeValue for text nodes is the text itself
- nodeValue for attribute nodes is the attribute value

DOM root nodes properties

- document.body
- document.documentElement

NodeName

- Provides the name of a node.
- nodeName of an element node is the same as the tag name
- nodeName of an attribute node is the attribute name
- nodeName of a text node is always #text
- nodeName of the document node is always #document

nodeType

Node	Type
ELEMENT_NODE	1
ATTRIBUTE_NODE	2
TEXT_NODE	3
COMMENT_NODE	8
DOCUMENT_NODE	9
DOCUMENT_TYPE_NODE	10

Event-driven programming

1. User interacts with page

2. An event occurs

3. A piece of JS code runs in the response. (Execute event handler function)

```
Function eventX() {
        …
}
```

4. The pages appearance is updated

Event handlers

- HTML events are **"things"** that happen to HTML elements. (https://www.w3schools.com/js/js_events.asp)
- JavaScript functions can be set as event handlers
- The function only executes when you interact with the element
- examples
  - An HTML web page has finished loading
  - An HTML input field was changed
  - An HTML button was clicked

Registering Event Handlers

- Assigning an event handler to an event on DOM tree node.
- Previous method: set a property on the object or document element that is the event target. In this method, developer sets the corresponding attribute directly in HTML.
- Newer method: pass the handler to a method of the object or element. addEventListener().

HTML button

```
<button>Click here</button>
```

Click on it!
Do nothing!

Add an event listener to the button
The **addEventListener()** method **attaches an event handler** to the specified element.

Onclick event

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id01">The onclick Event</h1>
<button onclick="myFunction()">Click me to change color</button>
<script>
function myFunction() {
  var element= document.getElementById("id01");
  element.style.color= "red";
}
</script>
</body>
</html>
```

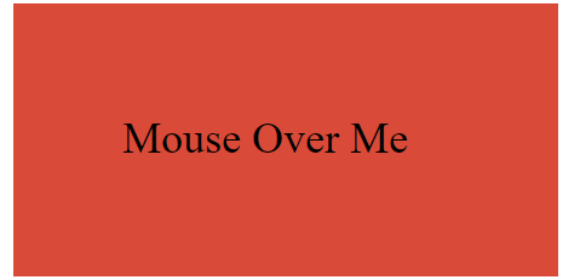
```
<!DOCTYPE html>
<html>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver(obj) {
  obj.innerHTML = "Thank You";
  obj.style.color= "yellow";
}

function mOut(obj) {
  obj.innerHTML = "Mouse Over Me";
  obj.style.color= "black";
}
</script>

</body>
</html>
```



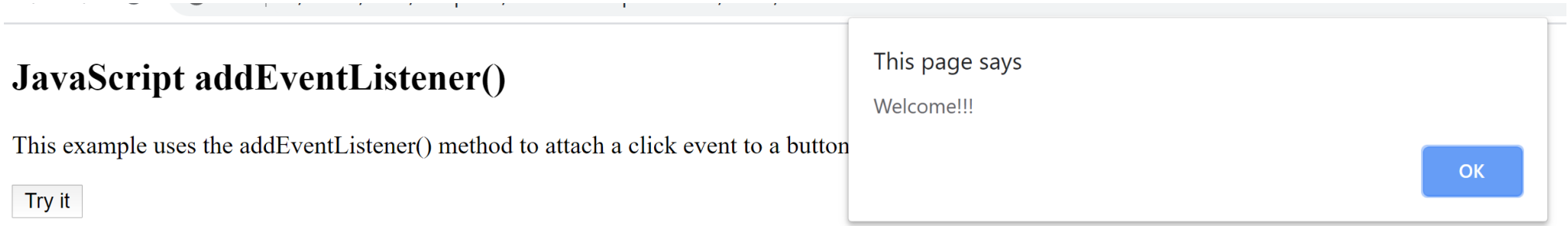
Event listener

- A listener is an object that listens, (and takes actions) upon certain events by evoking the event handler.
- The addEventListener() method attaches an event handler to the specified element.
- addEventListener(event name, function name);
  - event name is the name of the JavaScript event you want to listen to. For example: click, focus, blur, etc
  - function name is the name of the JavaScript function that will be executed when the event fires.

Anonymous functions

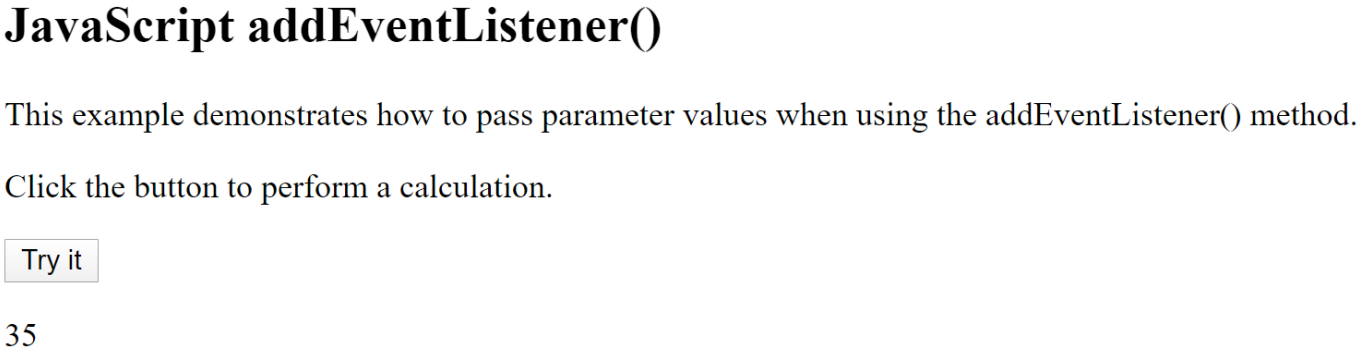
- JavaScript allows you to declare anonymous functions.
- An anonymous is a function without a name.
- Anonymous functions can be invoked directly, or they can store in a variable and they are invoked using the variable name.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<p>This example uses the addEventListener() method
to attach a click event to a button.</p>
<button id="myBtn">Try it</button>
<script>
document.getElementById("myBtn").addEventListener(
"click", function() {
  alert("Welcome!!!");
});
</script>
</body>
</html>
```



addEventListener: Passing parameter values

```
<body>
<h2>JavaScript addEventListener()</h2>
<p>This example demonstrates how to pass parameter values when using the addEventListener() method.</p>
<p>Click the button to perform a calculation.</p>
<button id="myBtn">Try it</button>
<p id="demo"></p>
<script>
var p1 = 5;
var p2 = 7;
document.getElementById("myBtn").addEventListener("click", function() {
  myFunction(p1, p2);
});
function myFunction(a, b) {
  var result = a * b;
  document.getElementById("demo").innerHTML = result;
}
</script>
</body>
```



Timing Events

- The window object allows execution of code at specified time intervals.
- setTimeout(function, milliseconds)

  calls given function after given delay in ms
- setInterval(function, milliseconds)

  calls given function repeatedly every given ms

```
<!DOCTYPE html>
<html>
<body>

<p id="p1">Click "Try it". Wait 3 seconds, and the page will
show a message.</p>

<button onclick="setTimeout(myFunction, 3000);">Try
it</button>

<script>
function myFunction() {
var para = document.createElement("p");
var node = document.createTextNode("This has been shown
after 3 seconds!");
para.appendChild(node);
var element = document.getElementById("p1");
element.appendChild(para);
}
</script>

</body>
</html>
```

Click "Try it". Wait 3 seconds, and the page will show a message.

This has been shown after 3 seconds!

Try it

More DOM objects

- **Window** represents an open window in a browser.
- **History** presents the list of pages the user has visited
- **Location** is the URL of the current HTML page
- **Navigator** contains information about the user’s browser.
- **Screen** contains information about the user's screen.

The window object

- The most top-level object in DOM hierarchy.
- Includes all global code and variables
- Properties:
  - document, history, location, name, navigator, screen, frames, etc.
- Methods:

  Alert(), confirm(), prompt ()
  Open(), close(), stop()
  Blur(), focus(), moveBy(), moveTo(),
  setInterval(), setTimeout()
  …

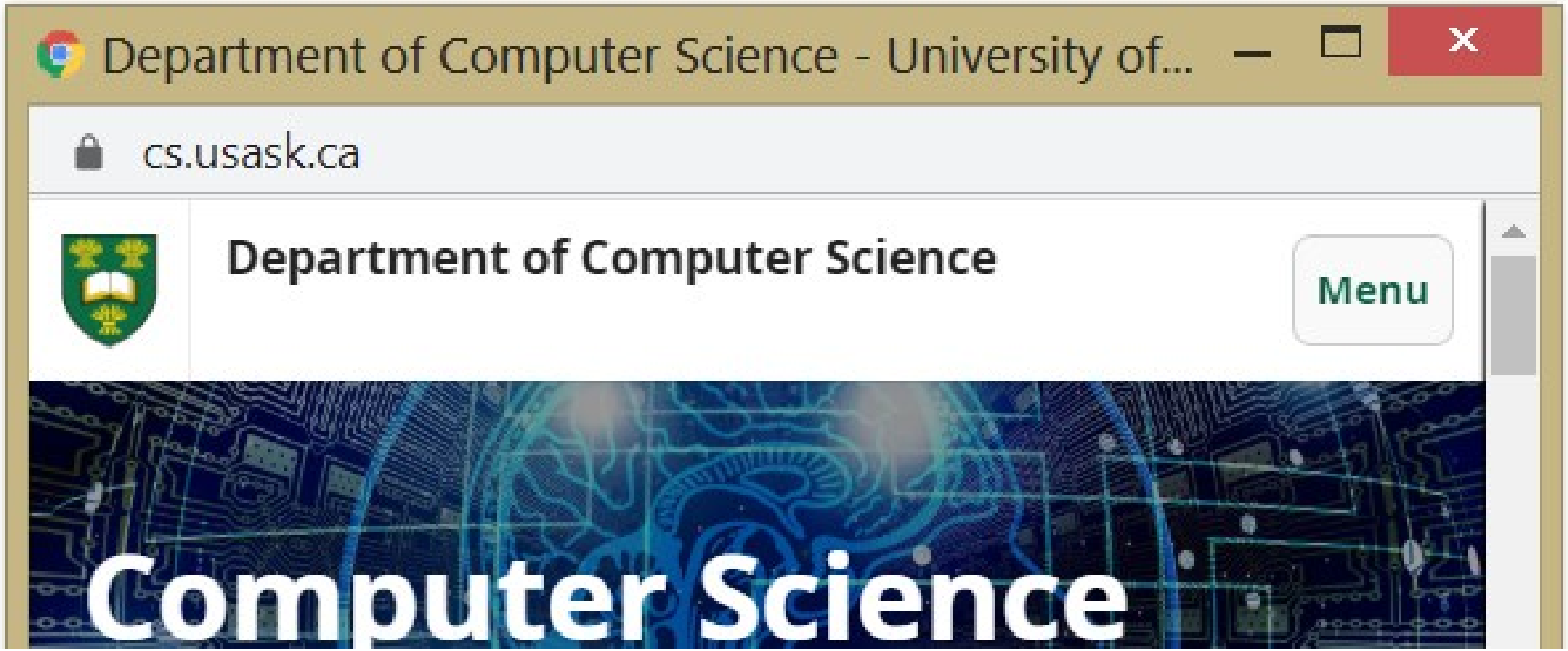
```
<!DOCTYPE html>
<html>
<body>
<p>Click the button to open the cs website in a new window.</p>
<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  var myWindow = window.open("https://www.cs.usask.ca/", "usask cs department",
"width=900,height=600,scrollbars=1");
}
</script>

</body>
</html>
```

Click the button to open the cs website in a new window.

Try it



The Screen object

- Presents information about the visitor's screen.
- Properties:

availheight, availwidth, colordepth, height, pixeldepth, width

The location object

- Properties:
  Hash, host, hostname, href, origin, etc.
- Methods:
  Assign(), reload(), replace()

```
<!DOCTYPE html>
<html>
<body>

<button onclick="myFunction()">Load new document</button>

<script>
function myFunction() {
  location.assign("https://www.cs.usask.ca/");
}
</script>

</body>
</html>
```

The Navigator object

- Provides information about the browser.

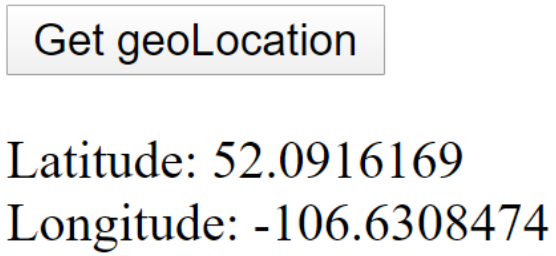
Properties:

appCodeName, appName, cookieEnabled, geolocation, etc.

Methods:

javaEnabled()

```
<!DOCTYPE html>
<html>
<body>
<button onclick="getLocation()">Get geoLocation</button>
<p id="demo"></p>
<script>
var x = document.getElementById("demo");
function getLocation() {
    navigator.geolocation.getCurrentPosition(showPosition);
}
function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
</script>
</body>
</html>
```



The history object

- Contains the URLs visited by the user.
- Properties:

  Length
- Methods:

  Back(), forward(), go()

The storage object

- The storage DOM object provides access to the session storage or local storage for a specific domain.
- Property:

  Length: Returns the number of data items stored in the Storage object
- Methods:

  key(n): Returns the name of the nth key in the storage

  getItem(keyname):Returns the value of the specified key name

  setItem(keyname, value): Adds the key to the storage, or update that key's value if it already exists

  removeItem(keyname): Removes the key from the storage

  clear():Empty all key out of the storage

```
<!DOCTYPE html>
<html>
<body>

<div id="result"></div>

<script>

if (typeof(Storage) !== "undefined") {
  localStorage.setItem("lastname", "Doe");
  document.getElementById("result").innerHTML = "This data is retrived from web storage: " +
localStorage.getItem("lastname");
} else {
  document.getElementById("result").innerHTML = "Your browser does not support Web Storage...";
}
</script>
</body>
</html>
```

This data is retrived from web storage: Doe

HTML <script> defer attribute

- A script that will not run until after the page (the DOM) has loaded

```
<script src="script.js" defer></script>
```

Alternative not recommended method:

Put the <script> tag at the bottom of the page

Listen for the "load" event on the window object

Event Propagation

- Event Propagation defines the elements order when an event occurs.
- Bubbling: first the inner most element's event is handled and then the outer element.
- Capturing: first the outer most element's event is handled and then the inner element.
- How to implement?

  Using "useCapture“ as the third parameter for addEventListener().

  addEventListener(*event*, *function*, *useCapture*);

  Default value is false and it refers to bubbling propagation.

  To use the capturing propagation, the value should be set to true.