Events

Introduction:

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Ex:01

```
<button onclick="handleClick()">Click Here</button>
<script type="text/javascript">
    function handleClick() {
       console.log('You clicked')
   }
</script>
```

Ex: 02

```
<button onclick="handleClick()">Click Here</button>

<script type="text/javascript">
    var clickedCount = 0
    function handleClick() {
        clickedCount++
        console.log('You clicked' + clickedCount)
    }
```

standard events:

Some of the HTML events and their event handlers are:

- Mouse Events
  - onclick – when mouse click on an element
  - onmouseover – when the cursor of the mouse comes over the element
  - onmouseout – when the cursor of the mouse leaves an element
  - onmousedown – when the mouse button is pressed over the element
  - onmouseup – when the mouse button is released over the element
  - onmousemove – when the mouse movement takes place
- Keyboard Events
  - onkeydown & onkeyup – when the user press and then release the key
- Form Events
  - onfocus – when the user focuses on an element
  - onsubmit – when the user submits the form
  - onblur – when the focus is away from the form element
  - onchange – when the user modifies or changes the value of a form element
- Window/Document Events
  - onload – when the browser finishes the loading of the page
  - onunload – when the visitor leaves the current webpage, the browser unloads it
  - onresize – when the visitor resizes the window of the browser

https://www.javatpoint.com/javascript-events

DOM Manipulation

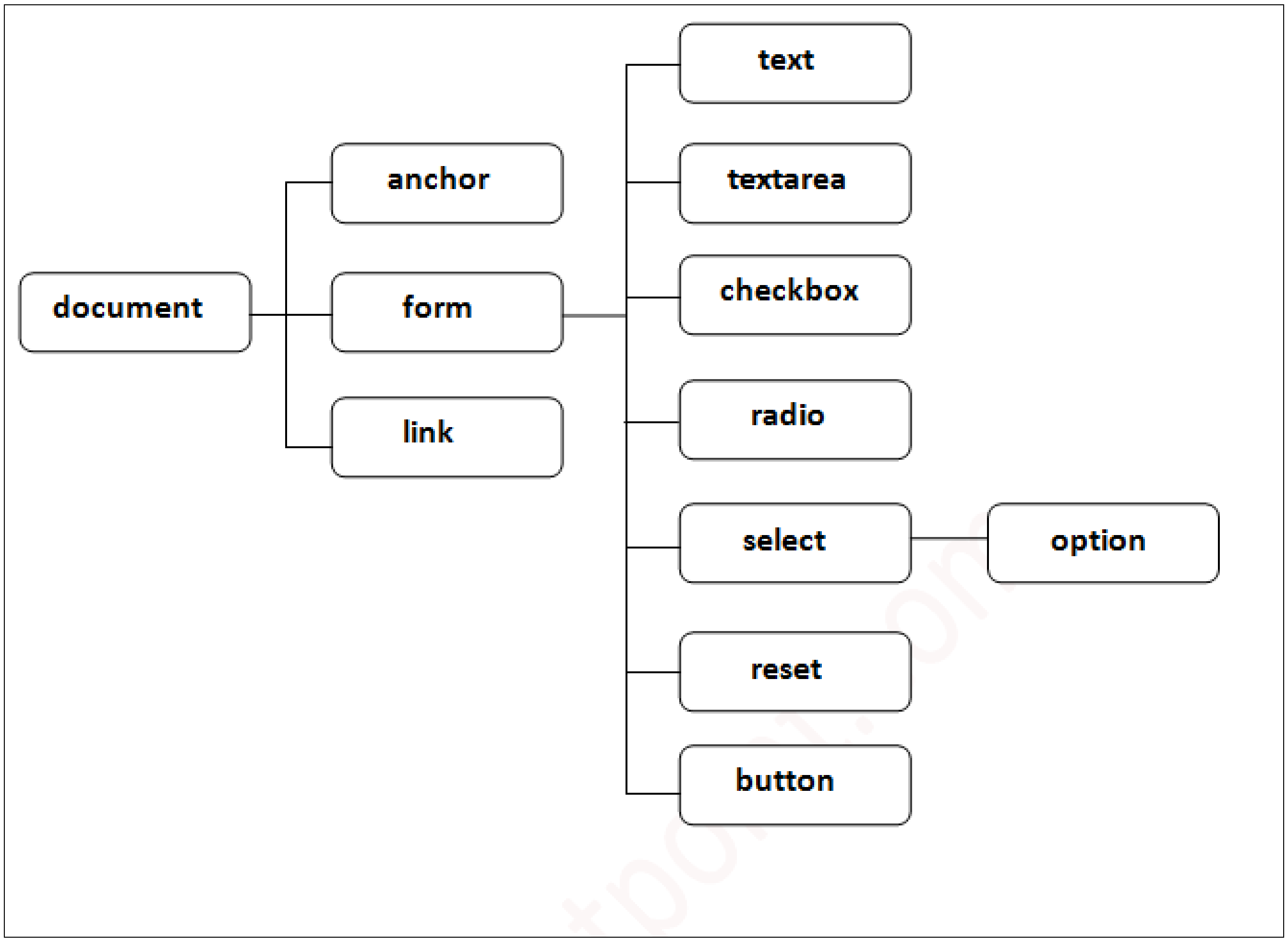
Document Object Model

The **document object** represents the whole HTML document.

When HTML document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

Properties of document object:

Let’s see the properties of document object that can be accessed and modified by the document object.



Methods of Document Object:

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucment with newline characte end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

JavaScript – *`document.getElementById()`* Method:

The ***`document.getElementById()`*** method returns the element of specified ‘id’

We will have two div elements, with a button at the buttom. Our goal is to create another box in the container element. Using the getElementById(), we will give the button element an **`id="addBox"`**. The same id is mentioned in the **`getElementById('addBox')`** such that every time we click on the button, we can see some change happening to the document. The function will return a html node which we will save in the **`buttonElement`**. The moment we click on the button, the console.log() will display **`<button id="addBox">Add Box</button>`.**

Ex:01

```
<div class="container">
    <div class="box"></div>
</div>

<button id="addBox">Add Box</button>
<script>
    var buttonElement = document.getElementById('addBox')
    console.log(buttonElement)

</script>
```

buttonElement is a html node or a wrapper object.

addEventListner:

The **addEventListner()** method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A webpage responds according to the even that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript procedure that waits for the occurrence of an event.

The **addEventListner() method** is an inbuilt function. We can add multiple event handlers to a particular element without overriding the existing event handlers.

Syntax:

```
element.addEventListener(event, function, useCapture);
```

event – specifies the event's name

Note: Do not use any prefix such as "on" with the parameter value. For example, Use "click" instead of using "onclick".

function – responds to the event occur

useCapture – a Boolean type value

Q) How to attach this event listener to the html document?

A) **buttonElement.addEventListner()**

Ex:02

```
<div class="container">
    <div class="box"></div>
</div>

<button id="addBox">Add Box</button>
<script>
    var buttonElement = document.getElementById('addBox')
    console.log(buttonElement)
    function handleClick() {
        console.log('Clicked')
    }
    buttonElement.addEventListner('click', handleClick)
</script>
```

Now when we click on the button, we get 'Clicked' in the console.

Now, to change the text content, since we already have reference to the buttonElement, so using

`buttonElement.innerText = 'Custom Text'`

the moment we click on it, it changes to Custom Text

This is what basically DOM Manipulation refers to.

innerHTML :

```
<button id="addBox">Add Box</button>
<script>

  var buttonElement = document.getElementById('addBox')

  console.log(buttonElement)

    function handleClick() {
      console.log('Clicked')

      var mySource = 'https://via.placeholder.com/30'
      buttonElement.innerHTML = `<img src="${mySource}" /> Cust
om Text`
      // buttonElement.innerText = `Custom Text`
    }
  buttonElement.addEventListener('Click', handleClick)
</script>
```

Simplest Way:

In order to do it, we need to get the reference of the container, because we are going to append into it. We will use the ***`document.getElementsByClassName()`*** method.

getElementByClassName()

The ***`getElementsByClassName()`*** method is used for selecting or getting the elements through their class name value. This DOM method returns an array-like object that consists of all the elements having the specified class name. On calling the

`getElementsByClassName()` method on any particular element, it will search the whole document and will return only those elements which match the specified or given class name.

```
Syntax:

Element.getElementsByClassName(name)
```

NOTE: ‘Id’ is unique in entire HTML document. If we have multiple elements with same id then we would get reference only to the first element that has required id. But for class, it can be applied to multiple elements.

```
<div class="container">
    <div class="box"></div>
</div>

<button id="addBox">Add Box</button>

<script>
  var buttonElement = document.getElementById('addBox')
  var container = document.getElementsByClass('container')
  console.log(container)

    function handleClick() {
      console.log('Clicked')
      // var mySource = 'https://via.placeholder.com/30'
      // buttonElement.innerHTML = `<img src="${mySource}" /> C
ustom Text`
      // buttonElement.innerText = `Custom Text`
      // buttonElement.addEventListener('Click', handleClick)
        }
</script>
```

```
Output:

HTMLCollection [div.container]

0: div.container
length: 1
```

So, if we add another element which has the same class (container) applied to it, then we will get multiple elements in an array-like object. It is not technically an array, but it symbolizes HTML nodes into a list.

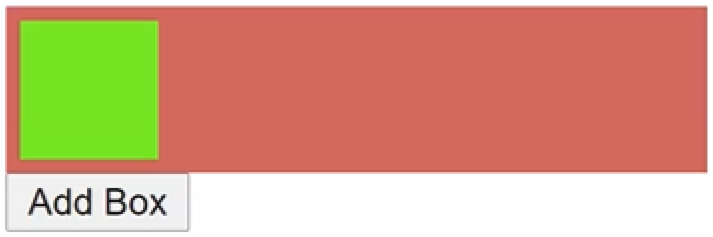
To access the element in the array-like object, we can use the indexing. So,

```
var container = document.getElementsByClass('container')[0]
```

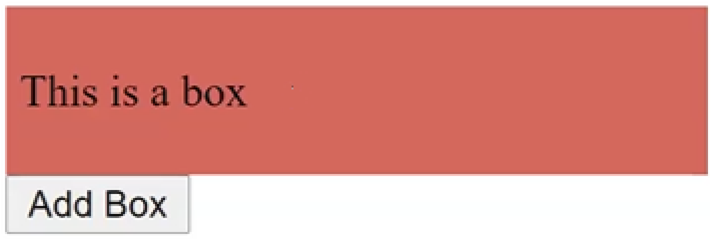
This will get the entire container. Earlier it was HTML collection, but now, it is HTML element.

```
<div class="container">
    <div class="box"></div>
</div>

<button id="addBox">Add Box</button>
<script>
    var buttonElement = document.getElementById('addBox')
    var container = document.getElementsByClassName('container')[0]
    console.log(container)

    buttonElement.addEventListener('Click', handleClick)
    function handleClick() {
        console.log('Clicked')
        container.innerHTML = '<p>This is a box</p>'
    }
</script>
```

Output:



After clicking



This is a box

Now, we can replace the

container.innerHTML = '<p>This is a box</p>'

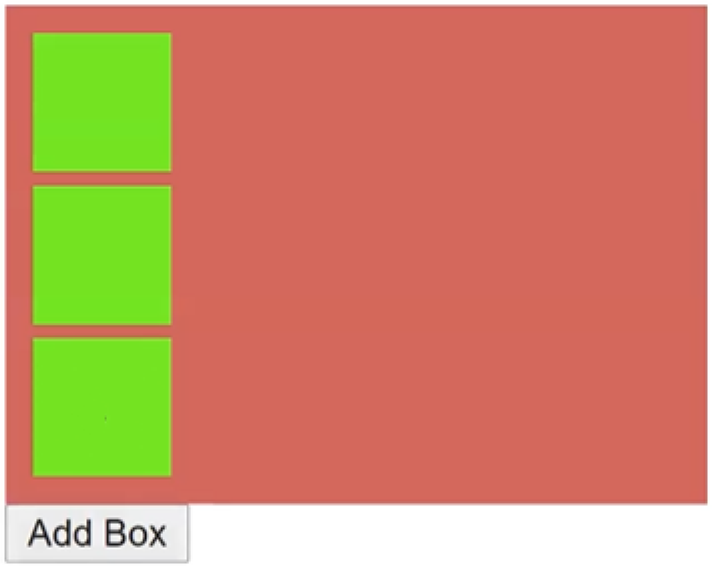
with

container.innerHTML = '<div class="box"></div>'

this will not make any changes as container element already has a box element in it. So, to add another box, we need to append to what is already present in the container element.

```
<div class="container">
    <div class="box"></div>
</div>

<button id="addBox">Add Box</button>
<script>
    var buttonElement = document.getElementById('addBox')
    var container = document.getElementsByClassName('container')[0]

     console.log(container)

    buttonElement.addEventListener('Click', handleClick)
    function handleClick() {
       console.log('Clicked')
       var oldContent = container.innerHTML

       container.innerHTML = oldContent + '<div class="box"></div>'
    }
</script>
```

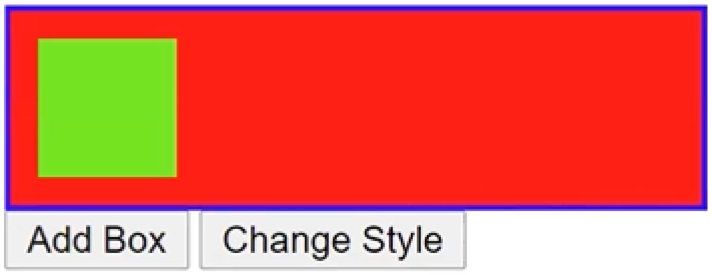
Output:



Styling:

```
<button id="styleBtn">Change Style</button>

<script>
    var styleButton = document.getElementById('styleBtn')
    var container = document.getElementsByClassName('container')[0]
    styleButton.addEventListener('Click', function(){

        container.style.backgroundColor = 'red'
        container.style.border = '2px solid blue'
    })
</script>
```

Output:



Instead of writing all the properties individually, we can create a class and define all the CSS in it and then add the class to the div element.

```
.myClass {
    background-color: black;
    border: 1px solid green;
}
container.classList.add('myClass')

container.classList.remove('myClass')
```

If we use ***`continer.className = 'test'`***

All the classes will be removed and it will be replaced by class test.

Event Bubbling

```
<div class="container">
    <div class="box"></div>
</div>

<script type="text/javascript">
    var container = document.getElementsByClassName('container')[0]
    var boxElement = document.getElementsByClassName('box')[0]

    container.addEventListener('click', function(){
        console.log('Clicked Container')
    })

    boxElement.addEventListener('click' function(){
        console.log('Clicked Box')
    })

</script>
```

Every time click on the container, in console we see, ***clicked container*** and when we click on the box, it shows ***clicked box clicked container.*** It should only speak about box and not the container. This phenomenon where, when clicked on child element, the parent element also gets fired, is called Event Bubbling.

When we attach a click event listener to the child and parent, then when you click on the child that event bubbles up and reaches to the parent which also gets executed.

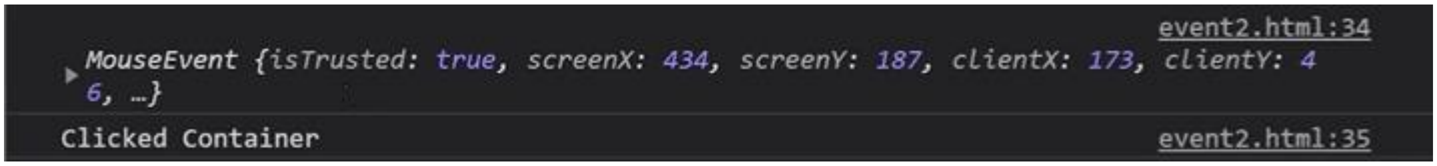
If we change the event in either the parent or the child element, event bubble can be stopped.

The browser also passes a parameter to the event, called event object. It will have information regarding the event itself. Generally people use ‘e’ or event.

```
<div class="container">
    <div class="box"></div>
</div>

<script type="text/javascript">
    var container = document.getElementsByClassName('container')[0]
    var boxElement = document.getElementsByClassName('box')[0]

    container.addEventListener('click', function(e){
        console.log(e)
        console.log('Clicked Container')
    })
    boxElement.addEventListener('click' function(){
        console.log('Clicked Box')
    })
</script>
```

Output:

In the console we get lot of information regarding the event.



```
                                                                    event2.html:34
▸ MouseEvent {isTrusted: true, screenX: 434, screenY: 187, clientX: 173, clientY: 4
  6, …}
Clicked Container                                                   event2.html:35
```

In this event there is something called as **`stopPropagation()`**. If we call this function, the bubbling will not happen. So, we will call it in the child element and it stops event bubbling that is being triggered by **`addEventListener()`**.

Prevent Default

When we enter a name in the input and click submit, it gets added to the url. If we don't want it to get attached to the url then,

```
<form action="#">
    <input type="text" name="username">
    <button type="submit">Submit</button>
</form>

<script type="text/javascript">
    var container = document.getElementsByTagName('form')[0]

    form.addEventListener('submit')

</script>
```

When the form gets submitted we want some function to be executed. We can add alert() function.

```
<form action="#">
    <input type="text" name="username">
    <button type="submit">Submit</button>
</form>

<script type="text/javascript">
    var container = document.getElementsByTagName('form')[0]

    form.addEventListener('submit', function(){
        alert('Form Submitted')
    })
</script>
```

So, in order to avoid “?username=****#” we can use the prevent default function.

```
<form action="#">
    <input type="text" name="username">
    <button type="submit">Submit</button>
</form>

<script type="text/javascript">
    var form = document.getElementsByTagName('form')[0]

    form.addEventListener('submit', function(e){
        e.preventDefault()
        console.log('Submitted')
    })
</script>
```

We could any logic in the addEventListener(). The idea is to stop the default behavior of the browser. There are some SEO related tasks for example, to see how many people are clicking on the link. So if we write a code and put e.preventDefault() before the code, then we can manually redirect the user to another page.