Document Object Model

Dr.G.Malathi
Associate Professor, SCSE
VIT University, Chennai Campus

Courtesy: w3schools.com

DOM Concept

- DOM makes all components of a web page accessible
  - HTML elements
  - their attributes
  - Text/values
- They can be created, modified and removed using JavaScript.
- It allows programs and scripts to navigate their structure, add, modify or delete elements and content

DOM

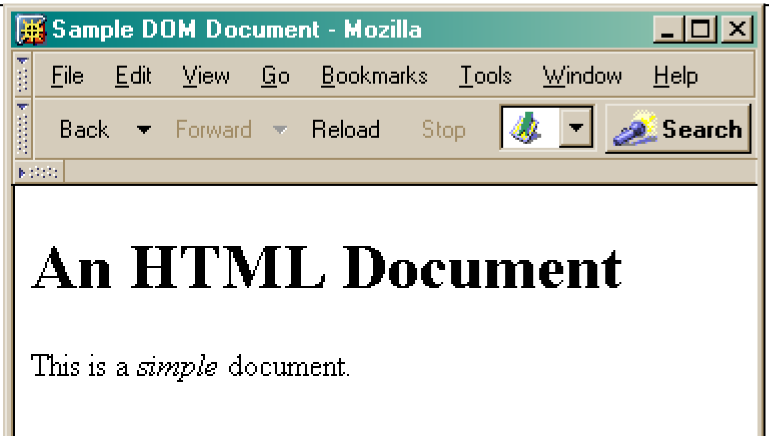
- The Document Object Model (DOM) allows JavaScript (and other scripting languages) to

  access the structure of the document in the browser.
- Each document is made up of structured nodes
- for example, the body tag would be a node, and any elements within the body element would be child nodes of the body element

This is what the browser reads

```
<html>
  <head>
    <title>Sample DOM Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.
  </body>
</html>
```

This is what the browser displays on screen.


Sample DOM Document - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop Search

An HTML Document

This is a *simple* document.

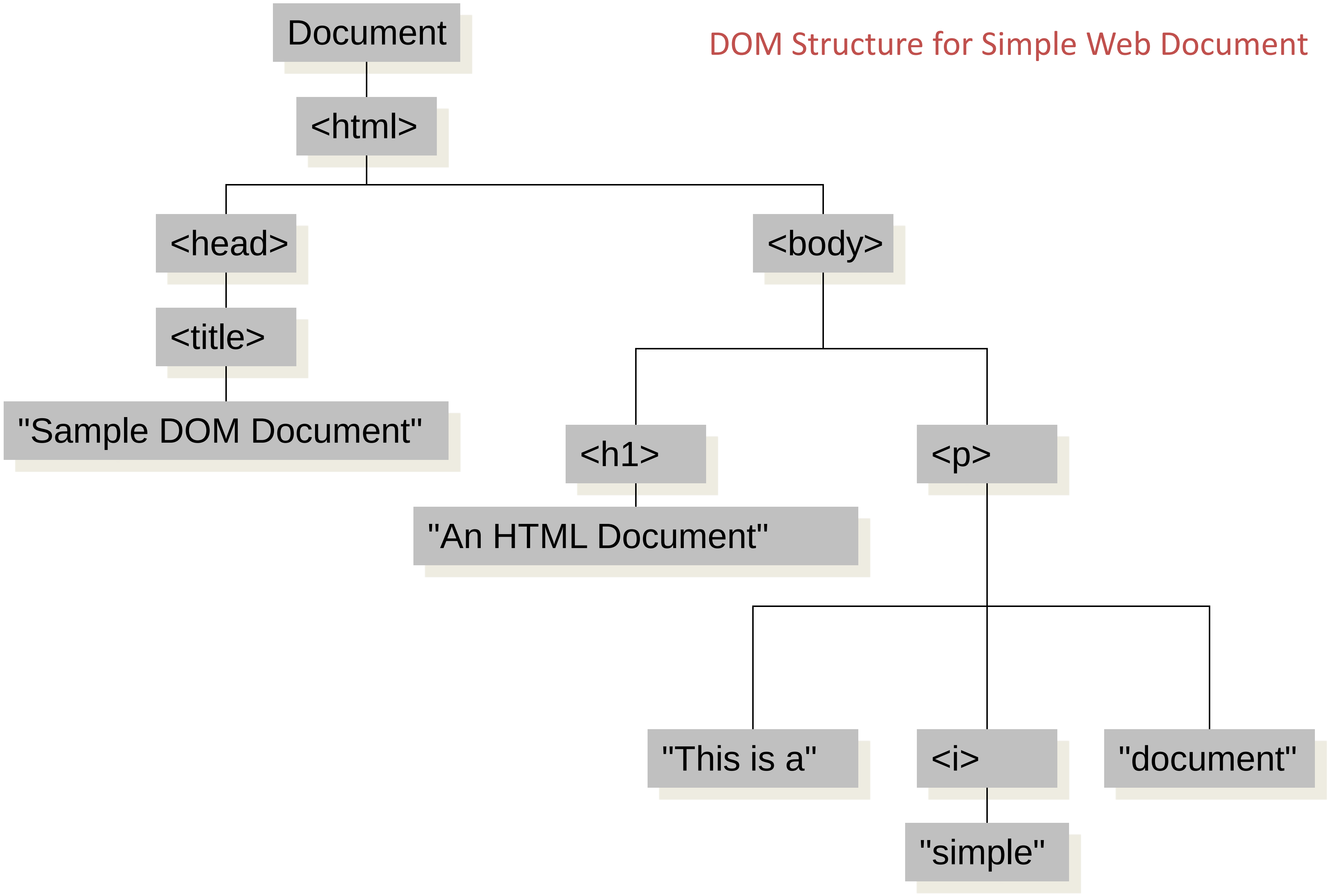
DOM Tree

- The documents in DOM are represented using a tree like structure in which every element is represented as a node.
- Hence the tree structure is also referred as DOM Tree.

DOM Tree

- Every element in the DOM tree is called node.
- The topmost single node in the DOM tree is called the root.
- Every child node must have a parent node.
- The bottommost node that have no children are called leaf nodes.
- The nodes that have the common parent are called siblings.

DOM Structure for Simple Web Document



DOM object Methods

Method	Description
getElementByID	Used to obtain the specific element which is specified by id within the script
createElement	This method is used to create an element node
CreateTextNode	Useful for creating a text node
CreateAttribute	Useful for creating attribute
appendChild	For adding a new child to specified node, this method is used.
RemoveChild	For removing a child node of a specific node this method is used
getAttribute	For returning the specified attribute value
setAttribute	For setting or changing the specified attribute to the specified value.

DOM object Properties

Method	Description
Attributes	This property is used to get the attribute of the nodes
parentNode	This is useful for obtaining the parent nde of the specific node
childNodes	Useful for obtaining the child nodes of the specific node
innerHTML	It is useful for getting the text value of a node.

Accessing Nodes by `id`

- Access to elements by their `id`
  - `document.getElementById(<id>)`
    - returns the element with `id` `<id>`
  - `id` attribute can be defined in each start tag

Other Access Methods

- Access by elements' tag
  - there are typically several elements with the same tag
  - **`document.getElementsByTagName(<tag>)`**
    - returns the collection of all elements whose tag is **`<tag>`**
    - the collection has a **`length`** attribute
    - an item in the collection can be reached by its index
  - e.g.
    - **`var html = document.getElementsByTagName(“h1") [0];`**
- Access by elements' **`name`** attribute
  - several elements can have the same name
  - **`document.getElementsByName(<name>)`**
    - returns the collection of elements with **`name`** **`<name>`**

Traversing DOM tree

- Traversal through node properties
  - `childNodes` property
    - the value is a collection of nodes
      - has a `length` attribute
      - an item can be reached by its index
    - e.g. `var body = html.childNodes[1];`
  - `firstChild`, `lastChild` properties
  - `nextSibling`, `previousSibling` properties
  - `parentNode` property

Text Nodes

- Text node
  - can only be as a leaf in DOM tree
  - it's `nodeValue` property holds the text
  - `innerHTML` can be used to access the text

Modifying DOM Structure

- **`document.createElement(<tag>)`**
  - creates a new DOM element node, with **`<tag>`** tag.
  - the node still needs to be inserted into the DOM tree
- **`document.createTextNode(<text>)`**
  - creates a new DOM text with **`<text>`**
  - the node still needs to be inserted into the DOM tree
- **`<parent>.appendChild(<child>)`**
  - inserts `<child>` node behind all existing children of `<parent>` node
- **`<parent>.insertBefore(<child>,<before>)`**
  - inserts **`<child>`** node before **`<before>`** child within `<parent>` node
- **`<parent>.replaceChild(<child>,<instead>)`**
  - replaces **`<instead>`** child by **`<child>`** node within `<parent>` node
- **`<parent>.removeChild(<child>)`**
  - removes **`<child>`** node from within `<parent>` node

Element Access in JavaScript

```
<html>
<head>
   <title> Login </title>
</head>
<body>
      <form name = “form1”>
           <input type=“text”
 name=“iptext”>
      </form>
 </body>
</html>
```

Element Access in Javascript

- Method1:
- Every document element is associated with some address.
- This address is called DOM address.
- The document has the collection of forms and elements.
- So the text box element can be refered as

```
var a=document.forms[0].elements[0];
```

Disadvantage with Method1

- If new controls are placed, then the index of the text control gets changed.

```
<form name = “form1”>
 <input type=“button” name=“button1”>
<input type=“text” name=“iptext”>
</form>
```

Now

```
var a=document.forms[0].elements[0];
```

No longer points to the text control

Element Access in JavaScript

- Method 2:
- Using the name attribute inorder to access the desired element.
- `var a=document.forms1.iptext;`

Disadvantage in Method 2

- Accessing through the name attribute is not supported in few standards.
- `var a=document.forms1.iptext;`

Element Access in JavaScript

- Method 3:
- The desired element from the web document can be access using getElementById.
- `<input type="text" name="iptext">`

- `var a=document.getElementById("iptext");`
-

Group objects

- If there are certain elements on the form such as radio buttons or check boxes then they normally appear in the groups.
- To access these elements the index can be used.

Group Objects- Example

```
<form id=“food”>
<input type=“checkbox” name=“states”
  value=“TamilNadu”/>
<input type=“checkbox” name=“states”
  value=“Delhi”/>
<input type=“checkbox” name=“states”
  value=“Andhra Pradesh”/>
</form>
var a =document.getElementById(“food”);
for (i=0;i<a.states.length;i++)
{ document.write(a.states[i]); }
```

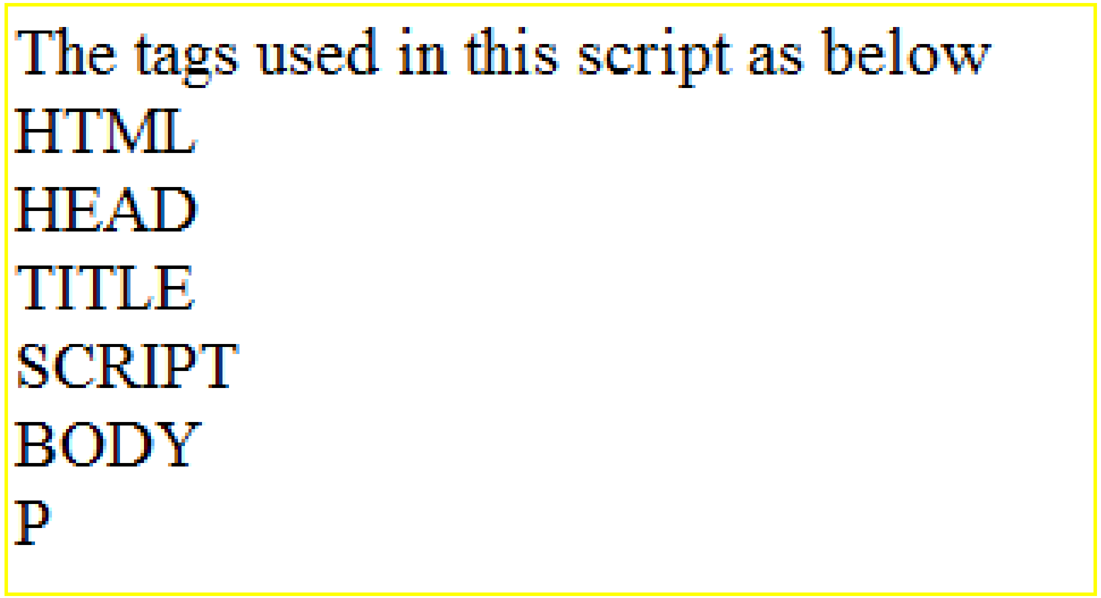
DOM Tree Traversal and Modification

- There is a special object model called *all* which is used to refer all the HTML elements.
- The order in which these HTML elements come in the program in the same order all those elements will be displayed.

DOM Tree Traversal – To display all the tags used in this document.

```
<html>
<head><title>Display HTML tags</title>
<script>
 var a="";
function display(){
for (i=0;i<document.all.length;i++)
{
a+="<br>"+document.all[i].tagName;
}
pmsg.innerHTML+=a;}
</script></head>
<body onload="display()">
<p id="pmsg"> The tags used in this script
  as below</p> </body> </html>
```

DOM Tree Traversal – To display all the tags used in this document.



```
The tags used in this script as below
HTML
HEAD
TITLE
SCRIPT
BODY
P
```

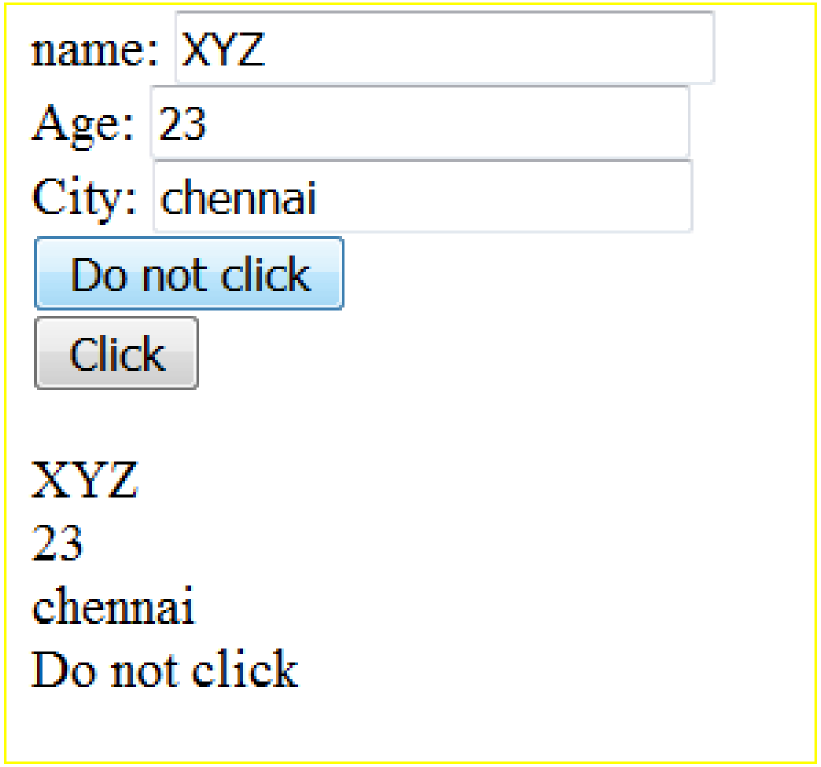
DOM Tree Traversal – using Element

- The element object has various properties using which the document tree can be traversed.
- To check what are the elements in the web document and its associated value/type

DOM Tree Traversal – using Element

```
<html>  <body> <form id="form1">
  name: <input type="text" name="name" value="XYZ"><br>
  Age: <input type="text" name="age" value="23"><br>
  City: <input type="text" name="city" value="chennai"><br>
  <input type="Button" value="Do not click">
</form>
<button onclick="myFunction()">Click</button>
<p id="demo"></p>
<script>
function myFunction() {
    var x = document.forms[0];      var txt = "";      var i;
    for (i = 0; i < x.length; i++) {
        txt = txt + x.elements[i].value + "<br>";
    }
    document.getElementById("demo").innerHTML = txt;  }
</script>   </body>  </html>
```

DOM Tree Traversal – using Element



```
XYZ
23
chennai
Do not click
```

To obtain active element in a document

The activeElement property returns the currently focused element in the document.

```
var x =
  document.activeElement.tagName;
Alert (x);
```

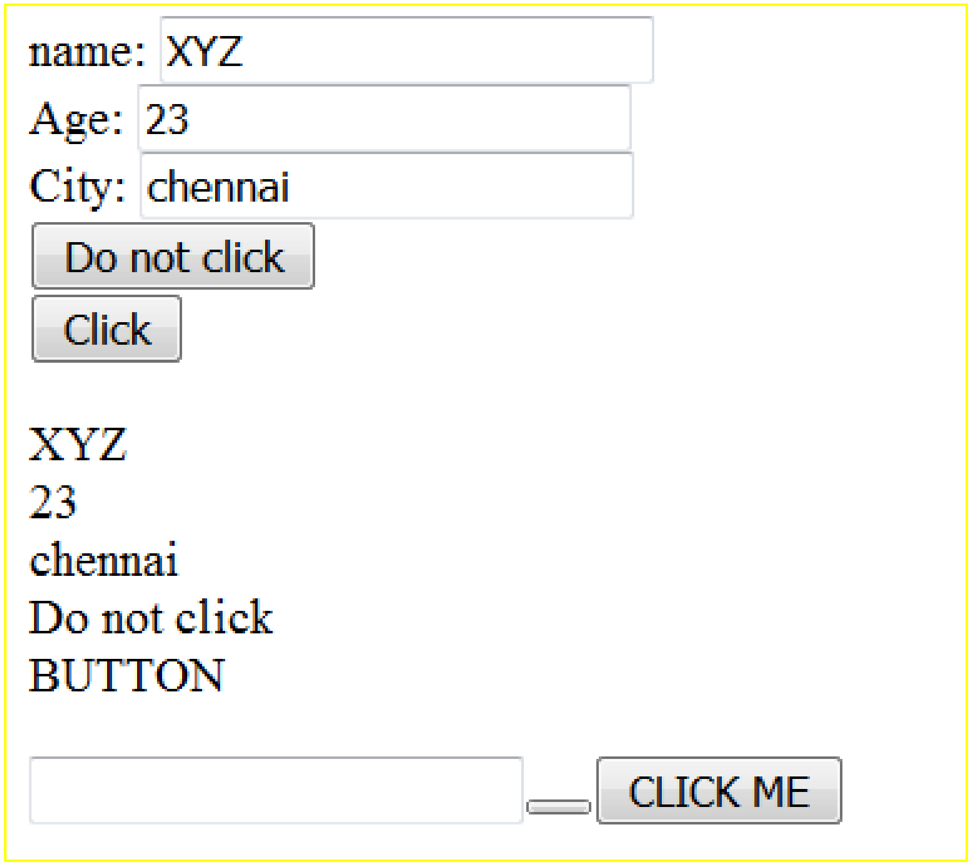
To createElement

```
var b=document.createElement("input");
document.body.appendChild(btn);
```

```
var btn = document.createElement("BUTTON");
document.body.appendChild(btn);
```

```
var btn = document.createElement("BUTTON");
var t = document.createTextNode("CLICK ME");
btn.appendChild(t);
document.body.appendChild(btn);
```

To createElement

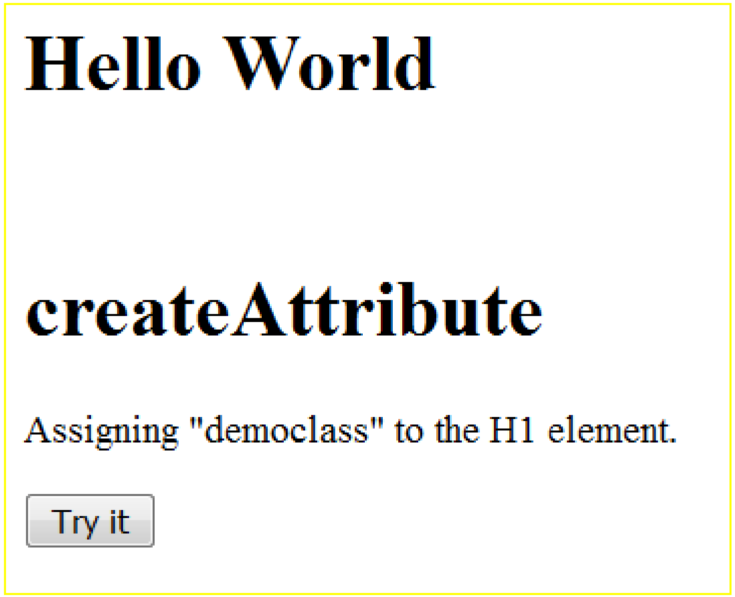

name: XYZ
Age: 23
City: chennai

XYZ
23
chennai
Do not click
BUTTON

Create and set attribute

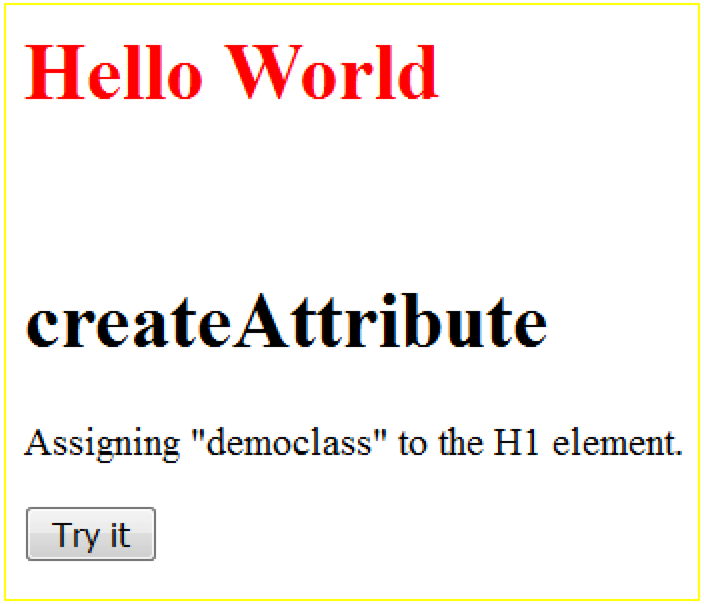
```
<html>  <head>
<style>
.democlass {  color: red; } </style> </head>
<body> <h1>Hello World</h1><br>
  <h1>createAttribute</h1>
<p>Assigning "democlass" to the H1
  element.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {      var h1 =
  document.getElementsByTagName("H1")[0];
  var att = document.createAttribute("class");
 att.value = "democlass“;
  h1.setAttributeNode(att);
}</script> </body> </html>
```

Create and set attribute


Hello World

createAttribute

Assigning "democlass" to the H1 element.

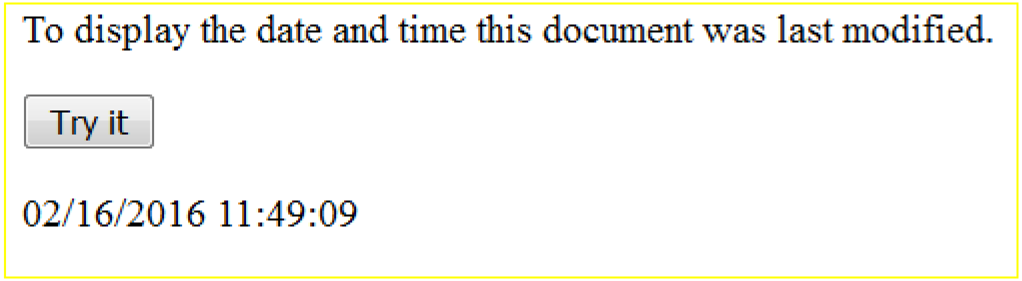

Hello World

createAttribute

Assigning "democlass" to the H1 element.

lastModified

- returns the date and time the current document was last modified.
- `var x = document.lastModified;`
- `document.getElementById("demo").innerHTML = x;`



To display the date and time this document was last modified.

```
02/16/2016 11:49:09
```

lastModified

```
<html>
<body>
<p>To display the date and time this
  document was last modified.</p>

<button onclick="myFunction()">Try
  it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var x = document.lastModified;
   document.getElementById("demo").innerHTML
  = x;
}</script> </body>  </html>
```

Insert value to list

- insertBefore(): The insertBefore() method inserts a node as a child, right before an existing child, which you specify.
- *node*.insertBefore(*newnode,existingnode*)

Insert value to list

```
<html>
<body>
<ul id="myList">
  <li>Coffee</li>
  <li>Tea</li>
</ul>
<p>Click to insert an item to the list.</p>
<button onclick="myFunction()">Try
  it</button>
```

Insert value to list

```
<script>
function myFunction() {
var newItem = document.createElement("LI");
var textnode =
  document.createTextNode("Water");
 newItem.appendChild(textnode);
var list =
  document.getElementById("myList");
    list.insertBefore(newItem,
  list.childNodes[0]);
}
</script>
</body>
</html>
```

Remove value from the list

- The removeChild() method removes a specified child node of the specified element.
- *node*.removeChild(*node*)

Remove value from the list

```
<html> <body>
<ul id="myList">  <li>Coffee</li>
  <li>Tea</li>  </ul>
<p>Click to remove an item to the list.</p>
<button onclick="myFunction()">Try
  it</button>
<script>
function myFunction() {
var list =
  document.getElementById("myList");
 list.removeChild(list.childNodes[0]);
} </script> </body> </html>
```

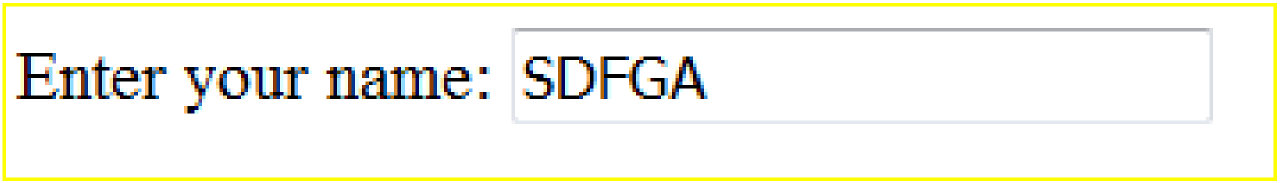
replace value from the list

```
<html> <body>
<ul id="myList">  <li>Coffee</li>
  <li>Tea</li>  </ul>
<p>Click to replace an item in the list.</p>
<button onclick="myFunction()">Try
  it</button>
<script>
function myFunction() {
var textnode =
  document.createTextNode("Water");
 var list =
  document.getElementById("myList");
 list.replaceChild(list.childNodes[0]);
} </script> </body> </html>
```

onchange

```
<html>
<head>
<script>
function myFunction() {
    var x =
  document.getElementById("fname");
    x.value = x.value.toUpperCase();
}
</script>
</head>
<body>
Enter your name: <input type="text"
  id="fname" onchange="myFunction()">
</body>
</html>
```

onchange



Enter your name: SDFGA

Mouse Events – on image

- onmousedown and onmouseup
  Change an image when a user holds down the mouse button.

onmousedown and onmouseup

```
<html>  <head> <script>
function babypuppy() {
    document.getElementById('myimage').src =
  "babypuppy.jpg";
}
function babycat() {
    document.getElementById('myimage').src =
  "babycat.jpg";
}
</script> </head> <body>
<img id="myimage" onmousedown="babypuppy()"
  onmouseup="babycat()" src="babycat.jpg"
  width="400" height="400" />
</body> </html>
```

Mouse Events – on image

- onmouseover onmouseout
- Change an image when a user moves over and out of the image
- .

Onmouseover and onmouseout

```
<html> <head> <script>
function babypuppy() {
    document.getElementById('myimage').src =
  "babypuppy.jpg"; }
function babycat() {
    document.getElementById('myimage').src =
  "babycat.jpg"; } </script> </head>
<body>
<img id="myimage" onmouseover="babypuppy()"
  onmouseout="babycat()" src="babycat.jpg"
  width="400" height="400" />
</body> </html>
```

Mouse events

- Change an image when a user holds down the mouse button.

Mouse Events

```
<html>
  <head>
<style>
#div1 { color:#000000; background-
  color:#FFFFFF; }
#div2 { border-style:solid; border-
  width:1px; border-color:#000000; }
 </style> </head>
```

Mouse Events

```
<body>
<div id="div1"> I am in div1. It
  seems like a nice place. </div>
<div id="div2"> I am in div2. It's a
  little fancier here. </div>
<script>
var d1 =
  document.getElementById("div1");
var d2 =
  document.getElementById("div2");
```

Mouse Events

```
d1.onmouseover =function myFunction()
{
d1.style.backgroundColor = "#00FF00";
d2.style.borderWidth = "7px";
}
d1.onmouseout = function
  myfunction1() {
d1.style.backgroundColor = "#FFFFFF";
d2.style.borderWidth = "1px";
} </script>  </body>  </html>
```