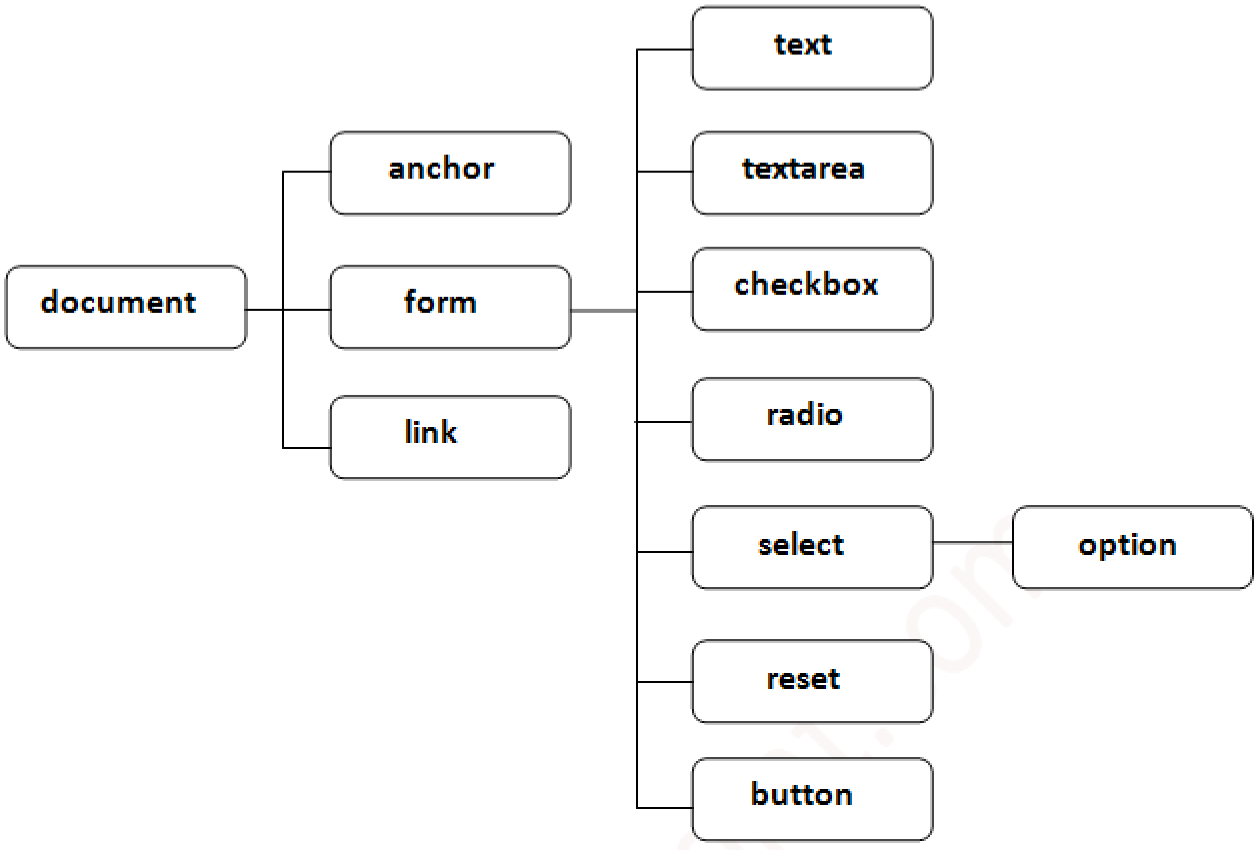
4. Introduction to Java Script

4.1. Basics of Java Script and Document Object Model

JavaScript is a programming language used to make web pages interactive. It runs on your visitor's computer and doesn't require constant downloads from your website. JavaScript support is built right into all the major web browsers, including Internet Explorer, Firefox and Safari. Provided that the visitors to your site are using web browsers that support JavaScript (most do) and have JavaScript enabled (it is by default), then your JavaScript will run when they visit the page.

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.

According to W3C - "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."



4.2. Element Access in Java Script

Java Script provides the ability for getting the value of an element on a web page as well as dynamically changing the content within an element. There are different methods of accessing elements of a web document using java script.

Let us consider a XHTML document as below:

XHTML Document

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>This  is my Web Page</title>
</head>
<html>
<body>
<form name="form1">
<input type="text" name="myinput"/>
</form>
</body>
</html>
```

Method 1

Every XHTML document element is associated with some address. This address. This address called DOM address. The document has the collection of forms and elements. Hence we can refer the text box element as-

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

But this is not the suitable method of addressing the elements. Because if we change the above script as:

```
....
<form name="form1">
<input type="button" name="mybutton"/>
<input type="text" name="myinput"/>
</form>
....
```

then index reference gets changed. Hence another approach of accessing the elements is developed.

Method 2

In this method we can make use of the name attribute in order to access the desired element.

```
var Dom_Obj=document.form1.myinput;
```

But this may cause a validation problem because the XHTML 1.1 standard does not support the name attribute to the form element. Hence another way of accessing is introduced.

Method 3

We can access the desired element from the web document using JavaScript method getElementById. This method is defined in DOM1. The element does access can be made as follows -

```
var Dom_Obj=document.getElementById("myinput")
```

But if the element is in particular group, that means if there are certain elements on the form such as radio buttons or check boxes then they normally appear in the groups. Hence to access these elements we make use of its index. Consider the following code sample:

```
<form name="form1">
<input type="checkbox" name="vegetables" value"Spinach" />Spinach
<input type="checkbox" name="vegetables" value"FenuGreek" />FenuGreek
<input type="checkbox" name="vegetables" value"Cabbage" />Cabbage
</form>
```

For getting the values of these checkboxes we can write following code

```
var Dom_Obj=document.getElementById("Food");
for(i=0;i<Dom_Obj.vegetables.length;i++)
document.write(Dom_Obj.vegetables[i]+"<br/>");
```

4.3. Events and Events Handling

Events In JavaScript enable your script to fire when certain events occur within the document, such as when mousing over an image, or setting focus on a form element.

To allow you to run your bits of code when these events occur, JavaScript provides us with event handlers. All the event handlers in JavaScript start with the word on, and each event handler deals with a certain type of event. Here's a list of all the event handlers in JavaScript, along with the objects they apply to and the events that trigger them:

Event handler	Applies to:	Triggered when:
onAbort	Image	The loading of the image is cancelled.
onBlur	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question loses focus (e.g. by clicking outside it or pressing the TABkey).
onChange	FileUpload, Select, Text, TextArea	The data in the form element is changed by the user.
onClick	Button, Document, Checkbox, Link, Radio, Reset, Submit	The object is clicked on.
onDblClick	Document, Link	The object is double-clicked on.
onDragDrop	Window	An icon is dragged and dropped into the browser.
onError	Image, Window	A JavaScript error occurs.
onFocus	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question gains focus (e.g. by clicking on it or pressing the TAB key).
onKeyDown	Document, Image, Link, TextArea	The user presses a key.
onKeyPress	Document, Image, Link, TextArea	The user presses or holds down a key.
onKeyUp	Document, Image, Link, TextArea	The user releases a key.
onLoad	Image, Window	The whole page has finished loading.
onMouseDown	Button, Document, Link	The user presses a mouse button.
onMouseMove	None	The user moves the mouse.
onMouseOut	Image, Link	The user moves the mouse away from the object.
onMouseOver	Image, Link	The user moves the mouse over the object.
onMouseUp	Button, Document, Link	The user releases a mouse button.
onMove	Window	The user moves the browser window or frame.
onReset	Form	The user clicks the form's Reset button.
onResize	Window	The user resizes the browser window or frame.
onSelect	Text, Textarea	The user selects text within the field.
onSubmit	Form	The user clicks the form's Submit button.
onUnload	Window	The user leaves the page.

To use an event handler, you usually place the event handler name within the HTML tag of the object you want to work with, followed by ="SomeJavaScriptCode", where SomeJavaScriptCode is the JavaScript you would like to execute when the event occurs.

For example:

```
<input type="submit" name="clickme"

value="Click Me!" onclick="alert('Thank You!')"/>
```

4.3.1. Some common event handlers

In this section, we'll look at a few of the more commonly used event handlers, and examine how they can be used.

onChange

onChange is commonly used to validate form fields or to otherwise perform an action when a form field's value has been altered by the user. The event handler is triggered when the user changes the field then clicks outside the field or uses the TAB key (i.e. the object loses focus).

This example code ensures that you type in both your first and your last names. It will bring up an alert box and refocus the text box field if you only type one word into the text box.

```
Please enter your name: <input type="text" name="your_name"
onchange="validateField(this)"/>

<script type="text/javascript">

function validateField ( fieldname )

{

  if ( ( fieldname.value ) &&

  ( ! fieldname.value.match ( " " ) ) )

  {

    alert ( "Please enter your first and last names!" );

    fieldname.focus ( );

  }

}

</script>
```

onClick

The onClick handler is executed when the user clicks with the mouse on the object in question. Because you can use it on many types of objects, from buttons through to checkboxes through to links, it's a great way to create interactive Web pages based on JavaScript.

In this example, an alert box is displayed when you click on the link.

```
<a href="#" onclick="alert('Thanks!')">Click Me!</a>
```

onFocus

onFocus is executed whenever the specified object gains focus. This usually happens when the user clicks on the object with the mouse button, or moves to the object using the TAB key. onFocus can be used on most form elements.

This example text box contains the prompt "Please enter your email address" that is cleared once the text box has focus.

```
<input type="text" name="email_address"

size="40" value="Please enter your email address"

onfocus="this.value=''"/>
```

onKeyPress

You can use onKeyPress to determine when a key on the keyboard has been pressed. This is useful for allowing keyboard shortcuts in your forms and for providing interactivity and games.

This example uses the onKeyPress event handler for the Window object to determine when a key was pressed. In addition, it uses the which property of the Event object to determine the ASCII code of the key that was pressed, and then displays the pressed key in a text box. If event.which is undefined it uses event.keyCode instead (Internet Explorer uses event.keyCode instead of event.which).

```
<html>

<body onkeypress = "show_key(event.which)">

<form method="post" name="my_form">

The key you pressed was:

<input type="text" name="key_display" size="2"/>

</form>
```

```
<script type="text/javascript">
function show_key ( the_key )
{
   if ( ! the_key )
   {
      the_key = event.keyCode;
   }
document.my_form.key_display.value  = String.fromCharCode ( the_key );
}
</script>
</body>
</html>
```

onLoad

The onLoad event handler is triggered when the page has finished loading. Common uses of onLoad include the dreaded pop-up advertisement windows, and to start other actions such as animations or timers once the whole page has finished loading.

This simple example displays an alert box when the page has finished loading:

```
<html>
<body onload = "alert('Thanks for visiting my page!')">
My Page
</body>
</html>
```

onMouseOut, onMouseOver

The classic use of these two event handlers is for JavaScript rollover images (images, such as buttons, that change when you move your mouse over them).

Here's a simple example that alters the value of a text box depending on whether the mouse pointer is over a link or not.

```
<form>
<input type="text" id="status" value="Not over the link"/>
<br>
<a href="" onmouseover="document.getElementById('status').value='Over the link'"
onmouseout="document.getElementById('status').value='Not over the link'">Move
the mouse over me!</a>
</form>
```

onSubmit

The onSubmit event handler, which works only with the Form object, is commonly used to validate the form before it's sent to the server. In fact we have a whole tutorial on this topic, called Form validation with JavaScript.

This example asks you to confirm whether you want to submit the form or not when you click on the button. It returns true to the event handler if the form is to be submiited, and false if the submission is to be cancelled.

```
<form onsubmit="return confirm('Are You Sure?')" action="">
<input type="submit" name="submit" value="Submit"/>
</form>
```

onblur

Fires when an element loses input focus. In most non IE browsers, this event only works on form elements such as INPUT and TEXTAREA, whereas in IE, it also fires for additional elements as well.

```
<input type="text" name="email_address"
size="40" value="Please enter your email address"
onblur="this.value=''try again"/>
```

4.4. DOM Element Positioning

Sometimes (especially in AJAX projects), it is necessary to get the position of some DOM element in "absolute" coordinates within the current document. For example, such an "absolute" position is needed if you would like to show some hidden DIV object exactly on the position (or with some offset) of another element. We use this function in one of our projects to show a popup menu under a text label.

Such properties as **style.left**, **style.top**, or **offsetLeft**, **offsetTop** can be used to get (or set) the position of an element within its parent. So, to get the element's absolute position within the document, we should move upwards on the element's tree and add the position of all the element's parents (except the latest document element).

However, it is not quite easy. There are still some problems:

1. First, we need to take into account possible scrolling in the element's parents and decrease our result accordingly.
2. Second, there are some distinctions in the behavior of different browsers. For Internet Explorer, we always just subtract the scrolling position of the object stored in the element's offsetParent property. But for FireFox, we also need to take into consideration all the parents accessible by the parentNode properties.
3. Finally, we should take into account the border width for some parent elements. Unfortunately, this task is not so easy as it can be supposed, especially for Internet Explorer.

NOTE: Mention the moving div example we did in lab for this.

4.5. Element Visibility

You can show or hide the div element dynamically using Javascript DOM getElementById method. visibility property of style object can be accessed using getElementById method of document object. visibility property can get or set the value that provides the functionality to toggle the show hide state of HTML <div> tag dynamically. When you use visibility property to hide the div element, it just hides the div layer along with its child HTML elements and inner text but it does not release the space block captured by it on the HTML document.

Basically visibility property belongs to the CSS styles that help to hide or show the HTML elements. Javascript DOM style object can access this CSS property that can modify its value dynamically.

visibility: visibility property accepts two types of values hidden or visible. hidden value for visibility property hides the target HTML element where as visible value for the property shows the target HTML element.

You can set the hidden or visible value dynamically using Javascript to show or hide the div element's visibility.

```
<html>
<head>
  <title>Javascript Show Hide Div Visibility</title>
  <style type="text/css">
    .divStyle {
      width: 200px;
      height: 100px;
      margin: 0px auto;
    }
  </style>
  <script language="javascript" type="text/javascript">
 function showHideDiv() {
      var divstyle = new String();
      divstyle = document.getElementById("div1").style.visibility;
      if (divstyle.toLowerCase() == "visible" || divstyle == "") {
        document.getElementById("div1").style.visibility = "hidden";
      }
      else {
        document.getElementById("div1").style.visibility = "visible";
      }
    }
  </script>
```

```
</head>
<body>
  <div id="div1" class="divStyle">
    Show Hide Div visibility using Javascript DOM.
  </div>
  <center>
    <input type="button"
        value="show hide div"
        onclick="showHideDiv()" />
  </center>
</body>
</html>
```

4.6. Changing Colors and Fonts

JavaScript can change colors and fonts of an element or text using some events. All we need to do is change the element's style property and modify the attributes of colors and fonts underlying it. Here is an example:

```
<html>
<head>
<title>DHTML and DOM</title>
<script language="javascript">
function turnRed() {
var myPara = document.getElementById("changeText");
myPara.style.color = "red";
}
</script>
</head>
<body>
```

```
<p id="changeText">This is my normal text.</p>


<p1><button onclick='turnRed()'>Turn Red</button></p1>
</body>
</html>
```

4.7. The alert, confirm, and prompt boxes

The three "commands" involved in creating alert, confirm, and prompt boxes are:

- window.alert()
- window.confirm()
- window.prompt()

Lets look at them in detail. The first one is:

window.alert()

This command pops up a message box displaying whatever you put in it. For example:

```
<body>
<script type="text/javascript">
window.alert("My name is George. Welcome!")
</script>
</body>
```

window.confirm()

Confirm is used to confirm a user about certain action, and decide between two choices depending on what the user chooses.

```
<script type="text/javascript">
var x=window.confirm("Are you sure you are ok?")
if (x) { window.alert("Good!") }
else { window.alert("Too bad") }
</script>
```

There are several concepts that are new here, and I'll go over them. First of all, "var x=" is a variable declaration; it declares a variable ("x" in this case) that will store the result of the confirm box. All variables are created this way. x will get the result, namely, "true" or "false". Then we use a "if else" statement to give the script the ability to choose between two paths, depending on this result. If the result is true (the user clicked "ok"), "good" is alerted. If the result is false (the user clicked "cancel"), "Too bad" is alerted instead. (For all those interested, variable x is called a Boolean variable, since it can only contain either a value of "true" or "false").

window.prompt()

Prompt is used to allow a user to enter something, and do something with that info:

```
<script type="text/javascript">

var y=window.prompt("please enter your name")

window.alert(y)

</script>
```

The concept here is very similar. Store whatever the user typed in the prompt box using y. Then display it.