Unit 3:

Client side Script: JavaScript

Introduction to JavaScript

- JavaScript was originally called LiveScript and was developed by Netscape Communications.
- JavaScript is a scripting language.
- A scripting language is a light weight programming language.
- A JavaScript consist of lines of executable computer code.
- JavaScript is embedded in Web pages and interpreted by the browser.
- A JavaScript was designed to add interactive to HTML pages.

Introduction to JavaScript

- Client-side Programs were developed to run programs and scripts on the client side of a Web browser
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- HTML and CSS concentrate on a static rendering of a page. Things do not change on the page at run time.
- JavaScript is Case Sensitive.
- Object based programming language
  - very limited object creation

Comparing Java and JavaScript

Java	JavaScript
A compiled language	An interpreted language
Requires the JDK (Java Developer's Kit) to create the applet	Requires a text editor
Requires a Java virtual machine or interpreter to run the applet	Requires a browser that can interpret JavaScript code
Applet files are distinct from the HTML and XHTML code	JavaScript programs are integrated and can be placed within HTML and XHTML code
Source code is hidden from the user	Source code is made accessible to the user
Powerful, requires programming knowledge and experience	Simpler, requiring less programming knowledge and experience

JavaScript Can Do…

- JavaScript can react to events
- JavaScript can read and write HTML elements
- JavaScript can be used to validate input data
- JavaScript can be used to detect the visitor's browser
- JavaScript can be used to create cookies

JavaScript Types

- **Two types of JavaScripts:**
  - **Client Side JavaScript:**
    - » Client Side Scripting generally refers to the class of computer programs on the web that are executed at client side by the user's web browser, instead of server-side (on the web server).
    - » This type of computer programming is an important part of the Dynamic HTML concept, enabling web pages to be scripted, that is to have different and changing content depending on user input, environmental conditions such as the time of the day, or other variables.

JavaScript Types

- **Two types of JavaScripts:**
  - **Client Side JavaScript:**
    - » Advantages:
      - The Web browser uses it's own resources, and remove the burden on the server.
    - » Disadvantages:
      - Code is usually visible
      - Code is Probably modifiable.
      - No access to database file.

JavaScript Types

- **Two types of JavaScripts:**
  - **Server Side JavaScript:**
    - » Normally when a browser requests an HTML file, the server returns the file , but if the file contains a server side script, the script inside the HTML file is executed by the server before the file is returned to the browser as a plain HTML.
    - » **What can server scripts do?**
      - Respond to user queries of data submitted from HTML forms
      - Access any data or databases and return the result for individual users.
      - Provide security since your server code cannot be viewed from the browser.

Inserting JavaScript into a Web Page

- A JavaScript program can either be placed directly in a Web page file or saved in an external text file
- Use the **<script>** tag (also use the **type** attribute to define the scripting language).

```
<html>
    <head>
        <script type="text/javascript">
        </script>
    </head>
    <body>
    </body>
</html>
```

Inserting JavaScript into a Web Page

- A JavaScript program can either be placed directly in a web page file or saved in an external text file.
- Use the **<script>** tag (also use the **type** attribute to define the scripting language). **Example: Include JS program directly in a web page.**

```
<html>
    <head>
        <script type="text/javascript">
        </script>
    </head>
    <body>
    </body>
</html>
```

Inserting JavaScript into a Web Page

- Scripts can be provided locally or remotely accessible JavaScript file using **src** attribute.
- **Example: Include external JS program file in a web page.**

```
<html>
    <head>
        <script type="text/javascript“ src=“JS/myfirstjs.js”>
        </script>
    </head>
    <body>
    </body>
</html>
```

Working with Variables and Data

- A variable is a named item in a program that stores information.
- Variable names are **case sensitive**.
- Variable names must begin with a **letter or the underscore** character.
- A variable declared within a JavaScript function becomes **LOCAL** and can only be accessed within that function. (the variable has local scope).
- Local variables are destroyed when you exit the function.
- Variables declared outside a function become **GLOBAL**, and all functions on the web page can access it.
- Global variables are destroyed when you close the page.

Working with Variables and Data

- We can declare variables with any of the following JavaScript commands:
  - **var variable;**
  - **var variable = value;**
  - **variable = value;**
- **Numeric or floating variable**- any number, such as 13, 22.5, etc
- **Boolean variable**- accepts only true and false values
- **Null variable**- has no value at all
- **String variable**- any group of text characters, such as “Hello” or “Happy Holidays!”
  - Must be enclosed within either double or single quotations

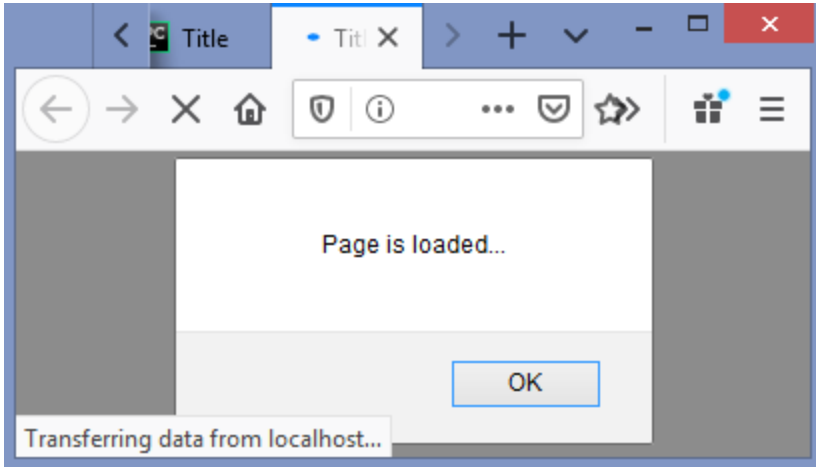
Working with Expressions and Operators

- **Expressions** are JavaScript commands that assign values and variables.

- **Operators** are elements that perform actions within expressions.
  - **Arithmetic operators**: perform simple mathematical calculations
  - **Binary operators**: work on two elements in an expression
  - **Unary operators**: work on only one variable
  - **Increment operators**: can be used to increase the value of a variable
  - **Assignment operators**: used to assign values in expressions

Creating JavaScript

```
<!DOCTYPE html>
<head>
        <title>Title</title>
</head>
<body>
    <script type="text/javascript">
        alert("Page is loaded...");
   </script>
</body>
</html>
```



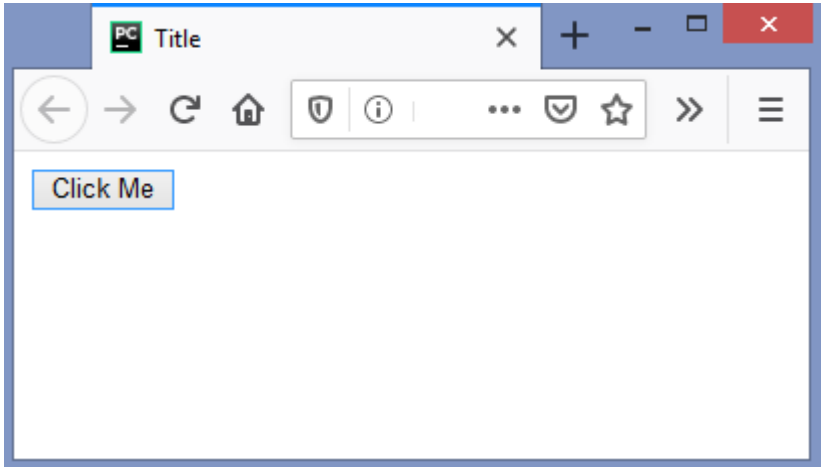
Creating JavaScript Functions

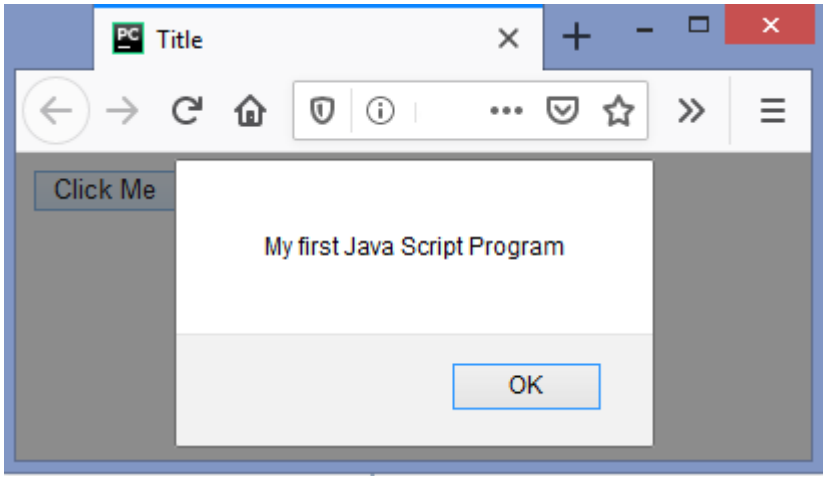
- A function contains code that will be executed by an **event** or by a **call** to the function.
- Parameters are values used by the function
- Functions can be defined both in the <head> and in the <body> section of a document.
- A group of commands set off by curly braces is called a command block. Command blocks exist for other JavaScript structures in addition to functions.
- A function with no parameters must include the parentheses () after the function name.

Creating JavaScript Functions Example

```
<head>
    <script type="text/javascript">
        function displaymessage()
            {
                alert("My first Java Script Program");
            }
    </script>
</head>
<body>
<form>
    <input type="button" value="Click me!" onclick="displaymessage()" />
</form>   </body>
```

Creating JavaScript Functions Example





Working with Conditional Statements

- Conditional statements are commands that run only when specific condition(s) are met. **(If then else then)** - Example
- Conditional statements require a Boolean value(s).
- Following operators we can use in JavaScript:
  - **Comparison operator**
  - **Logical operator (&& , || , !)**
  - **Ternary operator**

variablename=(condition)?value1:value2

Working with Array

- An array is an ordered collection of values referenced by a single variable name

  var variable = new Array (size);

  var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];

  - Where **variable** is the name of the array variable and
  - **size** is the number of elements in the array. - Example

Working with Program Loops

- A program loop is a set of instructions that is executed repeatedly.
- Loops execute a block of code a specified number of times, or while a specified condition is true.
  - **The for Loop:**

```
var i=0;
for (i=0;i<=5;i++)
{
document.write("The number is " + i);
document.write("<br />");
}
```

Working with Program Loops

- **The while Loop**

```
while (variable<=endvalue)
 {
 code to be executed
 }
```

- **The do.. While loop**

```
do
 {
 code to be executed
 }
while (variable<=endvalue);
```

- **The break / continue Statement**

```
for (i=0;i<=10;i++)
{
        if (i==3)
        {
         break ;    or    Continue;
        }

  document.write("The  number is " +
     i);
   document.write("<br />");
}
```

JavaScript Popup Boxes

- **Alert Box:**
  - An alert box is often used if you want to make sure information comes through to the user.

```
<head>
        <script type="text/javascript">
            function show_alert()
            {
                        alert("I am an alert box!");
            }
        </script>
</head>
<body>
<input type="button" onclick="show_alert()" value="Show alert box" />
</body>
```

JavaScript Popup Boxes

- **Confirm Box:**
  - A confirm box is often used if you want the user to verify or accept something.
  - When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
  - If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

JavaScript Popup Boxes

– **Confirm Box Example:**

```
<html>
    <head>
    <script type="text/javascript">
    function show_confirm()
    {
    var r=confirm("Are want to Cont.");
    if (r==true)
     {
     alert("You pressed OK!");
     }
else
 {
      alert("You pressed Cancel!");
 }
}
    </script>
    </head>
    <body>

    <input type="button"
    onclick="show_confirm()"
    value="Show confirm box" />

    </body>
</html>
```

JavaScript Popup Boxes

- **Prompt Box:**
  - A prompt box is often used if you want the user to input a value before entering a page.
  - When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
  - If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

JavaScript Popup Boxes

– **Prompt Box Example:**

```
<html>
    <head>
    <script type="text/javascript">
    function show_prompt()
    {
    var name=prompt("Please enter
    your name");
     if (name!=null && name!="")
       {
         document.write("Hello " +
         name + "! How are you today?");
       }
    }
    </script>
    </head>
    <body>

    <input type="button"
    onclick="show_prompt()"
    value="Show prompt box" />

    </body>
</html>
```

Common Mistakes during writing JS codes

- **Common mistakes include**:
  - Misspelling a variable name
  - Mismatched parentheses or braces
  - Mismatched quotes
  - Missing quotes
  - Using (instead of [
  - Using = in place of ==

JavaScript Events

Events	Event Attributes	Meaning	Associated Tags
Blur	**onblur**	Losing the focus	**<button> <input> <textarea> <select>**
Change	**onchange**	On occurrence of some change	**<input> <textarea> <select>**
Click	**onclick**	When user click the mouse button	**<a> <input>**
dbclick	**ondbclick**	When user double click the mouse button	**<a> <input> <button>**
Keyup	**onkeyup**	When user releases the key from the keyboard	**Form Element**

JavaScript Events

Events	Event Attributes	Meaning	Associated Tags
Focus	**onfocus**	When user acquires the input focus	**<input> <select> <textarea>**
Keydown	**onkeydown**	When user presses the key down	**Form Element**
keypress	**onkeypress**	When user presses the key	**Form Element**
mousedown	**onmousedown**	When user clicks the left mouse button	**Form Element**
Mouseup	**onmouseup**	When user releases the left mouse button	**Form Element**
Mousemove	**onmousemove**	When user move the mouse	**Form Elements**

JavaScript Events

Events	Event Attributes	Meaning	Associated Tags
Mouse out	**onmouseout**	when user moves the mouse away from some element	**Form Elements**
Mouse over	**onmouseover**	when the user moves the mouse over some element	**Form Elements**
Load	**onload**	After getting the document loaded	**<body>**
Reset	**onreset**	when the reset button is clicked	**<form>**
Submit	**onsubmit**	when the submit button is clicked	**<form>**
Select	**onselect**	On selection	**<input> <textarea>**
Unload	**onunload**	When user exits the document	**<body>**

JavaScript Objects

- **Document object:**
  - When an HTML document is loaded into a web browser, it becomes a document object.
  - The document object is the root node of the HTML document and the "owner" of all other nodes: (element nodes, text nodes, attribute nodes, and comment nodes).
  - The document object provides properties and methods to access all node objects, from within JavaScript.
  - **Tip: The document is a part of the Window object and can be accessed as window.document.**

JavaScript Objects

- **Document Object Properties:**
  - document.bgcolor = “colorname”;
  - document.write(“Statement”);
  - document.writeln(“Statement”);
  - document.cookie = “key=value”;
  - document.formname
  - document.getElementById(“idname”)
  - document.title: to access the title of document
  - document.linkcolor = “colorname”;

JavaScript Objects

- **String Object Properties:**
  - strvar.charAt(index);
  - strvar.length;
  - strvar.endsWith(“string or char”); or strvar.startsWith(“string or char”);
  - strvar.includes(“string or char”);
  - strvar.indexOf(“char”);
  - strvar.lastIndexOf(“char”);
  - strvar.match(RegularExpression);
  - strvar.toLowerCase();
  - strvar.toUpperCase();

Method to read and write HTML Elements

- **There are two Methods:**
  - **Using Form name:** with this method, we can read the HTML Element.
    - **Syntax:** document.formname.elementname.value;
    - **Example:** document.f1.fn.value;
  - **Using ID:** with this method we can read and write HTML Element.
    - **Syntax(read):** document.getElementById("ElementID").value;
    - **Example:** document.getElementById("fn1").value;
    - **Syntax(write):** document.getElementById("ElementID").innerHTML = "Content to Display";
    - **Example:** document.getElementById("s1").innerHTML = "Enter First Name";

Forms and Validation

- **Required Fields:**
  - The function below checks if a field has been left empty. If the field is blank, an alert box alerts a message, the function returns false, and the form will not be submitted.

```
function validateForm()
    {
    var x=document.f1.fn.value;
      if (x==null || x=="")
                {
                        alert("First name must be filled out");
                        document.f1.fn.focus();
                        return false;
                 }
    }
```

Forms and Validation

- **Validating User:**

```
function validateForm()
{
    var myform = document.f1;
            if(myform.pass.value==“letmein”)
                    {
                            document.write(“Welcome”);
                            return true;
                    }
            else
                    {
                            alert(“Wrong Password”);
                            myform.pass.focus();
                    }
}
```

Forms and Validation

- **Validating Checkbox or Radio button:**

```
function validateForm()
{
    if (document.f1.checkThis.checked==true)
    {
            alert("The box is checked.")
    }
    else
    {
            alert("The box is not checked at the moment.")
    }
}
```

Forms and Validation

- **Minimum & Maximum Characters for Password field :**

```
function validateForm()
{
    if(document.f1.pass.value.length <=“6” || document.f1.pass.value.length
    >=“12”)
    {
        alert('Your Password Should have Min 6 & Max 12 Char');
        document.form1.pass.focus();
        return false;
    }
}
```

Forms and Validation

- **Mobile number (10 digit):**

```
 function validateForm()
{
      var mobile = document.getElementById("mob").value;
       var pattern = “/^\d{10}$/”;                //    /^[0-9]{6,10}$/
           if (pattern.test(mobile))
             {
                        alert("Your mobile number : "+mobile);
                        return true;
              }
            else
             {
                        alert("It is not valid mobile number.input 10 digits number!");
                        return false;
              }
}
```

Forms and Validation

- **Validating AlphaNumeric Field:**

```
 function validateForm()
{
    var alphaExp = /^[a-zA-Z0-9]+$/;
    if(document.f1.fn.value.match(alphaExp))
    {
        return true;
    }
    else
    {
        alert("Letters only please!!!!!");
        return false;
    }
}
```

Forms and Validation

- **Field should contains capital, small letter and special character:**

```
function validateForm()
{
    var x=document.f1.pass.value;
    var RegExpr= /^[a-z]+[A-Z]+[@#$%^&*]+$/;
   if (RegExpr.test(x))
        {
            return true;
         }
        else
         {
            alert("Invalid Input!");
            return false;
         }
}
```

Forms and Validation

- **Validating Email Field:**
  - This means that the input data must contain an @ sign and at least one dot (.). Also, the @ must not be the first character of the email address, and the last dot must be present after the @ sign, and minimum 2 characters before the end.

```
function validateForm()
{                                        abc_123@gmail.com
    var x=document.forms[“f1”][“email”].value;
    var atpos=x.indexOf("@");
    var dotpos=x.lastIndexOf(".");
    if (atpos<1  || dotpos+2>=x.length)
     {
          alert("Not a valid e-mail address");
         return false;
    }
}
```

Forms and Validation

- **Validating Email with Regular Expression:**

```
function validateForm()
{
    var x=document.f1.email.value;
    var RegExpr= /^[a-z0-9]+[@][a-z]+[.][a-z]{2,3}([.][a-z]{2})*$/;
   if (RegExpr.test(x))
        {
            return true;
        }
       else
        {
            alert("Invalid Email Address!");
            return false;
        }
}
```

Forms and Validation

- **Validating Radio Button:**

```
function validateForm()
{
if (document.f1.gender[0].checked == false  &&  document.f1.gender[1].checked == false)
{
alert ( "Please choose your Gender: Male or Female" );
return false;
}
}
```

Forms and Validation

- **Validating Radio Button:**

```
<html>
<head>
   <script language="javascript">
            function f1(f_o){
                     alert("You have selected "+f_o.value); }
   </script>
</head>
<body>
            <form name="form1">
            <input type="radio" name="group1" value="Red" onclick=f1(this)> Red
            <input type="radio" name="group1" value="Blue" onclick=f1(this)> Blue
            </form>
</body>
</html>
```

JavaScript Objects

- **Window object:**
  - **window** object is JavaScript representation of a browser window.
  - **window** object represents the window or frame that displays the document and it is the global object in client side programming
  - **Properties:**
    - window.open("URL", "Windowname", "Options like height and width");
    - window.close();
    - window.defaultStatus = "Statement to display";

JavaScript Objects

- **Window object Example:**

```
<script>
function openWin()
{
myWin=window.open(“www.google.com”, “Google”, “width=200,height=100”);
myWin.document.write(“<p>This is 'myWindow'</p>”);
}
function closeWin()
{
window.close();
}
</script> <body>
<input type="button"  value="Open 'myWindow'" onclick="openWin()" />
<input type="button" value="Close 'myWindow'" onclick="closeWin()" />
</body>
```

JavaScript Objects

- **Navigator object:**
  - The navigator object contains information about the browser. Read-only!

Property	Description
appCodeName	Returns the code name of the browser
appName	Returns the name of the browser
appVersion	Returns the version information of the browser
cookieEnabled	Determines whether cookies are enabled in the browser
onLine	Boolean, returns true if the browser is on line, otherwise false.
platform	Returns for which platform the browser is compiled
userAgent	Returns the user-agent header sent by the browser to the server

JavaScript Objects

- **Navigator object Example:**

```
<html><body>
<script type="text/javascript">
document.write("Browser Name is: " + navigator.appName);
document.write("<br>Browser Code is: " + navigator.appCodeName);
document.write("<br>Browser Version No is: " + navigator.appVersion);
document.write("<br>Platform is: " + navigator.platform);
document.write("<br>User agent Header: " + navigator.userAgent);
document.write("<br>Cookies Enabled is: " + navigator.cookieEnabled);
</script>
</body><html>
```

JavaScript Objects

- **Screen object:**
  - The screen object contains information about the visitor's screen.

Property	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)
height	Returns the total height of the screen
pixelDepth	Returns the color resolution (in bits per pixel) of the screen
width	Returns the total width of the screen
colorDepth	specifies the depth of color palette, in bits, to display images

JavaScript Objects

- **Screen object Example:**

```
<html>
<body>
<script>
document.writeln("<br/>Screen Width: "+screen.width);
document.writeln("<br/>Screen Height: "+screen.height);
document.writeln("<br/>Screen availWidth: "+screen.availWidth);
document.writeln("<br/>Screen availHeight: "+screen.availHeight);
document.writeln("<br/>Screen pixelDepth: "+screen.pixelDepth);
document.writeln("Screen Color Depth : " +screen.colorDepth + "<br/>");
</script>
</body>
</html>
```

JavaScript Objects

- **Location object:**
  - The location object contains information about the current URL.

Property	Description
host	Returns the hostname and port number of a URL
hostname	Returns the hostname of a URL
href	Returns the entire URL
origin	Returns the protocol, hostname and port number of a URL
pathname	Returns the path name of a URL
port	Returns the port number of a URL
protocol	Returns the protocol of a URL
search	Returns the querystring part of a URL

JavaScript Objects

- **Location object Example:**

```
<head>
<script>
function myFunction()
{
   var x = location.host;
   document.getElementById("demo").innerHTML = x;
}
</script></head><body>
<p>Click the button to display the hostname and port number of the current
   URL.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>.
</body>
```

JavaScript Objects

- **Date object:**
  - Create a date object to store date information. **var Today = new Date();**

Method	Description	Value
In the following examples, assume that the variable Today stores the date object: Date("April, 8, 2006, 12:25:28")		
Today.getSeconds()	Retrieves the seconds from the date	28
Today.getMinutes()	Retrieves the minutes from the date	25
Today.getHours()	Retrieves the hour from the date	12
Today.getDate()	Retrieves the day of the month from the date	8
Today.getDay()	Retrieves the day of the week from the date (0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday)	6
Today.getMonth()	Retrieves the month from the date (0=January, 1=February, ...)	3
Today.getFullYear()	Retrieves the four-digit year number from the date	2006
Today.getTime()	Retrieves the time value, as expressed in milliseconds since December 31, 1969, 6 P.M.	1,144,520,728,000

JavaScript Objects

- **Math object:**
  - The Math object is a JavaScript object used for calculations other than simple math.

Math Method	Description
Math.abs(*number*)	Returns the absolute value of *number*
Math.sin(*number*)	Calculates the sine of *number*, where *number* is an angle expressed in radians
Math.cos(*number*)	Calculates the cosine of *number*, where *number* is an angle expressed in radians
Math.round(*number*)	Rounds *number* to the closet integer
Math.ceil(*number*)	Rounds *number* up to the next-highest integer
Math.floor(*number*)	Rounds *number* down to the next-lowest integer
Math.random()	Returns a random number between 0 and 1

Document Object Model (DOM)

- Document Object Model (DOM) is a set of platform independent and language neutral application programming interface which describes how to access and manipulate the information stored in XML, XHTML and JavaScript documents.
- DOM is an API that defines the interface between XHTML and application program.
- That means, suppose application program is written in JAVA and this java program wants to access the elements of XHTML web document then it is possible by using a set of API which belongs to DOM.
- A simple DOM was implemented in Netscape 2.0 browser.

Document Object Model (DOM)

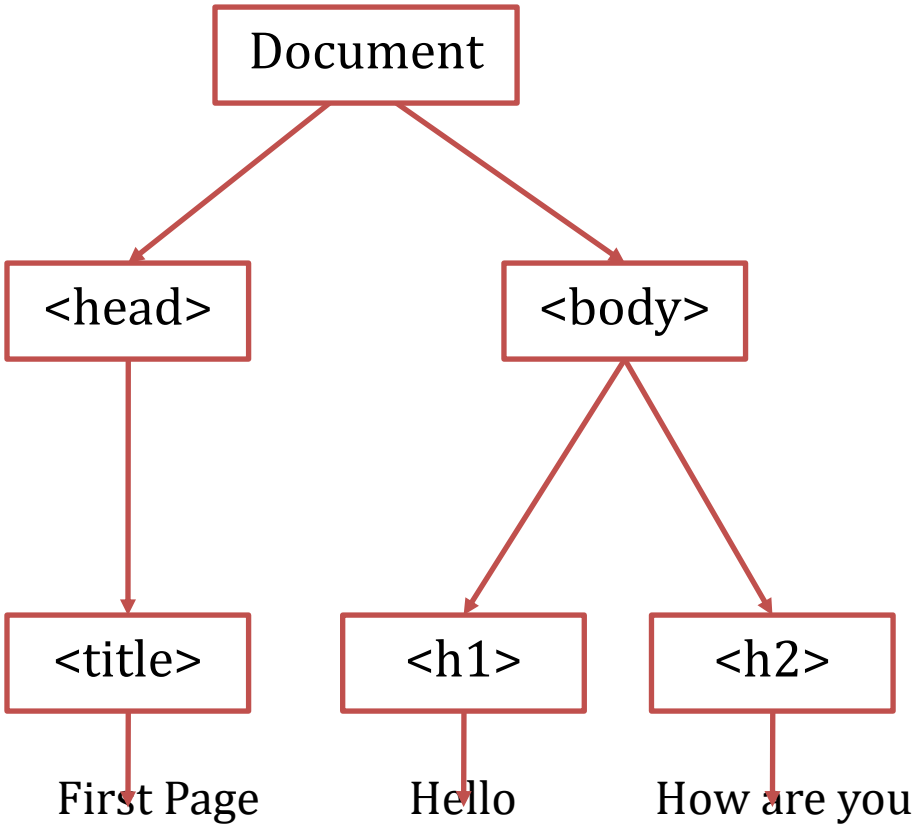
Level	Description
DOM 0	This model is supported by early browsers. This level could support JavaScript. This version was implemented in Netscape 3.0 and IE 3.0 browser.
DOM 1	Issued in 1998 which was focused on XHTML & XML.
DOM 2	Issued in 2000 that could specify the style sheet. It also supports the event model with in the document.
DOM 3	Current release of DOM specification published in 2004. this version could deal with XML with DTD and schema, document validations, document views and formatting.

Document Object Model (DOM)

- The document in DOM are represented using a tree like structure in which every element is represented as a node.
- **Basic terminologies used in DOM tree as follows:**
  - Every element in the DOM tree is called node.
  - The topmost single node in the DOM tree is called root.
  - Every child node must have parent node.
  - The bottommost nodes that have no children are called leaf nodes.
  - The nodes that have the common parent are called siblings.

Document Object Model (DOM)

```
<html>
<head>
<title> First Page </title>
</head>
<body>
<h1> Hello </h1>
<h2> How are you </h2>
</body>
</html>
```



JavaScript Timing Events

- It is possible to execute some code after a specified time-interval. This is called timing events.
- **The two key methods that are used:**
  - **setTimeout()** - executes a code some time in the future

    var t=setTimeout("javascript statement",milliseconds);

    The setTimeout() method returns a value.
  - **clearTimeout()** – stop/cancels the setTimeout()

    Argument is setTimeout variable (i.e **t**)
- To get a timer to work in infinite loop, write a function that calls itself.

JavaScript Timing Events

- **setTimeout() Example:**

```
<script type="text/javascript">
function timeMsg()
{
        var t=setTimeout("alertMsg()",3000);
}
function alertMsg()
{
        alert("Hello");
}
</script>
<body>
<form>
        <input type="button" value="Box in 3 seconds" onclick="timeMsg()" />
</form>
</body>
```

JavaScript Timing Events

- **Counter Start Example:**

```
<head>
<script type="text/javascript">
var c=0;   var t;
function doTimer()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("doTimer()",1000);
}
</script>
</head>
<body>
<form>
<input type="text" id="txt" />
<input type="button" value="Start count!" onclick="doTimer()">
</form>
</body>
```

JavaScript Timing Events

- **Counter Start and Pause/Stop Exa:**

```
<head>
<script type="text/javascript">
var c=0;   var t;
function doTimer()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("doTimer()",1000);
}
```


```
function stopCount()
{
clearTimeout(t);

document.getElementById('txt').value=""
;  c=0;  //if needed
 }
</script>
</head>
<body>
<input type="button" value="Start
 count!" onclick="doTimer()">
<input type="text" id="txt">
 <input type="button" value="Stop
 count!" onclick="stopCount()">
 </body>
```

Dynamic HTML

- "DHTML" is a term used by some vendors to describe the combination of HTML, style sheets and JavaScript that allows documents to be animated.
- DHTML is the combination of 3 browser technologies to render a dynamic browser experience.
  - HTML for content
  - Cascading Style Sheets for general presentation
  - JavaScript to enable dynamic presentation
- With Dynamic HTML the emphasis is on making the browser provide a richer viewing experience through effects like Animation, Filters, Transitions etc.

Dynamic HTML Advantages/Disadvantages

- **Advantages:**
  - Small file sizes (faster than Flash)
  - No plug-ins required
  - Easy to learn (learn HTML, JavaScript)
  - Faster web experience (change the page content without load new pages)
- **Disadvantages:**
  - Browser and OS incompatibilities
  - The implementation of CSS, DOM varies from browser to browser

The Role of each component in DHTML

- With CSS, we can change the style of any HTML elements.
- With DOM, we can have a map on every elements in the HTML page.
- With JavaScript, we can access and have operations on the elements in the DOM tree.
- With Event Handler, we can execute the predefined scripts at any time.

Dynamic HTML Example (Image Viewer)

First Previous Next Last

Dynamic HTML Example (Image Viewer)

```
<html>
<body>
<img id="imageviewer" title="abc" src="anim0.jpg" width="200" height="100“ />
<form name="form1">
<input name="First" type="button" value="First" onclick="first()">
<input name="Previous" type="button" value="Previous" onclick="previous()">
<input name="Next" type="button" value="Next" onclick="next()">
<input name="Last" type="button" value="Last" onclick="last()">
</br>
<input type="checkbox" name="automatic" onclick="automaticly()">
</form>
</body>
</html>
```

Dynamic HTML Example (Image Viewer)

```
<script type="text/javascript">
var myImages=new Array();
myImages[0]="anim0.jpg";
myImages[1]="anim1.jpg";
myImages[2]="anim2.jpg";
myImages[3]="anim3.jpg";
myImages[4]="anim4.jpg";
myImages[5]="anim5.jpg";
myImages[6]="anim6.jpg";
myImages[7]="anim7.jpg";
myImages[8]="anim8.jpg";

var imagecounter=myImages.length-1;
var i=0;
```

Dynamic HTML Example (Image Viewer)

```
function first()
{
document.getElementById('imageviewer').src=myImages[0];
i=0;
}
function last()
{
document.getElementById('imageviewer').src=myImages[imagecounter];
i=imagecounter;
}
```

Dynamic HTML Example (Image Viewer)

```
function next()
{
if (i<imagecounter)
      {
      i++;
      document.getElementById('imageviewer').src=myImages[i];
      }}
function previous()
{
if (i>0)
      {
      i--;
      document.getElementById('imageviewer').src=myImages[i];
      }}
```

Dynamic HTML Example (Image Viewer)

```
function automaticly()
{
if (document.form1.automatic.checked)
{
if (i<myImages.length)
{
document.getElementById('imageviewer').src=myImages[i];
i++;
var delay = setTimeout("automaticly()",1500);
}
}
}
</script>
```

JavaScript: Introduction to Cookies

- Web Browser and Server use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website it is required to maintain session information among different pages. How to maintain user's session information across all the web pages?
- In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases and other information required for better visitor experience or site statistics.
- A cookie is a variable that is stored on the visitor's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With JavaScript, you can both create and retrieve cookie values.

JavaScript: Introduction to Cookies

- Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive.
- Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval.
- Once retrieved, your server knows/remembers what was stored earlier.

JavaScript: Cookies Variable

- **Cookies are a plain text data record of 5 variable-length fields:**
  - **Expires**: The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
  - **Domain**: The domain name of your site. Specifies the domain for which the cookie is valid.
  - **Path**: The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page. commonly specified to /, the root directory.
  - **Secure**: If this field contains the word "secure" then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
  - **Name=Value**: Cookies are set and retrieved in the form of key and value pairs. Value is an information we wish to save, in reference to a particular cookie. name is an identifier by which we reference a particular cookie.

JavaScript: Cookies as Objects

- JavaScript deals with cookies as objects.
- Specifically, JavaScript works with cookies using the **document.cookie.**
- We can read information from cookies by examining the **document.cookie** object.

JavaScript: Create Cookies

With JavaScript, a cookie can be created like this:

document.cookie = "username=GCET";

We can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

document.cookie = "username=GCET; expires=Tue, 3 Feb 2020 12:00:00 UTC";

With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

document.cookie = "username=GCET; expires=Tue, 3 Feb 2020 12:00:00 UTC; path=/";

JavaScript: Read Cookies

With JavaScript, cookies can be read like this:

var x = document.cookie;

document.cookie will return all cookies in one string much like: cookie1=value; cookie2=value; cookie3=value;

Change a Cookie with JavaScript

With JavaScript, you can change a cookie the same way as you create it:

document.cookie = "username=GCET CVM; expires=Tue, 3 Feb 2020 12:00:00 UTC; path=/";

The old cookie is overwritten.

JavaScript: Delete Cookies

we don't have to specify a cookie value when you delete a cookie.

Just set the expires parameter to a passed date:

document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";

We should define the cookie path to ensure that you delete the right cookie.

Some browsers will not let you delete a cookie if you don't specify the path.

JavaScript: Cookie String

The **document.cookie** property looks like a normal text string. But it is not.

Even if you write a whole cookie string to **document.cookie**, when you read it out again, you can only see the name-value pair of it.

If you set a new cookie, older cookies are not overwritten. The new cookie is added to document.cookie, so if you read document.cookie again you will get something like:

cookie1 = value; cookie2 = value;

If you want to find the value of one specified cookie, you must write a JavaScript function that searches for the cookie value in the cookie string.

JavaScript: Setting or Storing a Cookie

- The simplest way to create a cookie is to assign a string value to the document.cookie object, which looks like this:

  document.cookie="yourname=" + prompt("What is your name?");

- Setting a Cookie – General Form:

  window.document.cookie = “cookieName = cookieValue; expires = expireDate; path = pathName; domain = domainName; secure”;

JavaScript: Setting or Storing a Cookie

- **Example:**

```
<script type="text/javascript">
        function writeCookie()
             {
             if( document.f1.un.value == "" )
                          {
                          alert("Enter some value!");
                          }
             cookievalue= escape(document.f1.un.value);
             document.cookie="name=" + cookievalue;
             alert("Setting Cookies : " + "name=" + cookievalue );
             }
</script>
```

JavaScript: Setting or Storing a Cookie

- Cookie values may not include semicolons, commas, or whitespace.

- For this reason, we have to use the JavaScript **escape()** function to encode the value before storing it in the cookie.

Callbacks in Javascript

- A callback is a function passed as an argument to another function
- This technique allows a function to call another function
- A callback function can run after another function has finished

Function Sequence

- JavaScript functions are executed in the sequence they are called. Not in the sequence they are defined. – Example (Javascript3_callback_fun_sequence)

Sequence Control

- Sometimes you would like to have better control over when to execute a function.
- Suppose you want to do a calculation, and then display the result. – Example (Javascript3_callback_fun_sequence_control)
- The problem with the example, is that you have to call two functions to display the result. The problem with the second part of example, is that you cannot prevent the calculator function from displaying the result. Now it is time to bring in a callback.

Callbacks in Javascript

- A callback is a function passed as an argument to another function.
- Using a callback, you could call the calculator function (myCalculator) with a callback, and let the calculator function run the callback after the calculation is finished
- In the example(Javascript3_callback_callback), myDisplayer is the name of a function. It is passed to myCalculator() as an argument.
- When you pass a function as an argument, remember not to use parenthesis.

  Right: myCalculator(5, 5, myDisplayer);

  Wrong: ~~myCalculator(5, 5, myDisplayer());~~

Callbacks in Javascript

When to Use a Callback?

- The examples above are not very exciting.
- They are simplified to teach you the callback syntax.
- Where callbacks really shine are in asynchronous functions, where one function has to wait for another function (like waiting for a file to load).

Asynchronous JavaScript

- The purpose of the example (Javascript3_callback_asyncronous) was to demonstrate the syntax of callback functions: In the example, myDisplayer is the name of a function. It is passed to myCalculator() as an argument.

Callbacks in Javascript

Waiting for a Timeout

- When using the JavaScript function setTimeout(), you can specify a callback function to be executed on time-out:
- In the example (Javascript3_callback_waiting timeout), myFunction is used as a callback. The function (the function name) is passed to setTimeout() as an argument. 3000 is the number of milliseconds before time-out, so myFunction() will be called after 3 seconds.
- When you pass a function as an argument, remember not to use parenthesis.
  - Right: setTimeout(myFunction, 3000);
  - Wrong: ~~setTimeout(myFunction(), 3000);~~

Callbacks in Javascript

Waiting for a Timeout

- Instead of passing the name of a function as an argument to another function, you can always pass a whole function instead
- In the example above, function(){ myFunction(“Web Programming !!!"); } is used as a callback. It is a complete function. The complete function is passed to setTimeout() as an argument.
- 3000 is the number of milliseconds before time-out, so myFunction() will be called after 3 seconds.

Callbacks in Javascript

Waiting for Intervals:

- When using the JavaScript function setInterval(), you can specify a callback function to be executed for each interval
- In the example (Javascript3_callback_setinterval()), myFunction is used as a callback.
- The function (the function name) is passed to setInterval() as an argument.
- 1000 is the number of milliseconds between intervals, so myFunction() will be called every second.

Callbacks in Javascript

Waiting for Files

- If you create a function to load an external resource (like a script or a file), you cannot use the content before it is fully loaded.
- This is the perfect time to use a callback.
- This example (Javascript3_callback_wait for files()) loads a HTML file (mycollege.html), and displays the HTML file in a web page, after the file is fully loaded:
- In the example above, myDisplayer is used as a callback.
- The function (the function name) is passed to getFile() as an argument.

Function as arguments in JavaScript

- Passing a function as an argument to the function is quite similar to the passing variable as an argument to the function. Below examples describes to passing a function as a parameter to another function.

  Javascript_Fun_as_Args_1

- This example passes a function geeks_inner alongwith a argument ‘Geeks!’ to the function geeks_outer as a argument.

  Javascript_Fun_as_Args_2

Introduction to JSON

- JSON stands for JavaScript Object Notation
- JSON is a text format for storing and transporting data
- JSON is "self-describing" and easy to understand