JavaScript : Introduction To Client Side Scripting

Working with 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.
- They can execute on the client side or server side.

Introduction to JavaScript

- Server-side Programs pose problems
- 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.

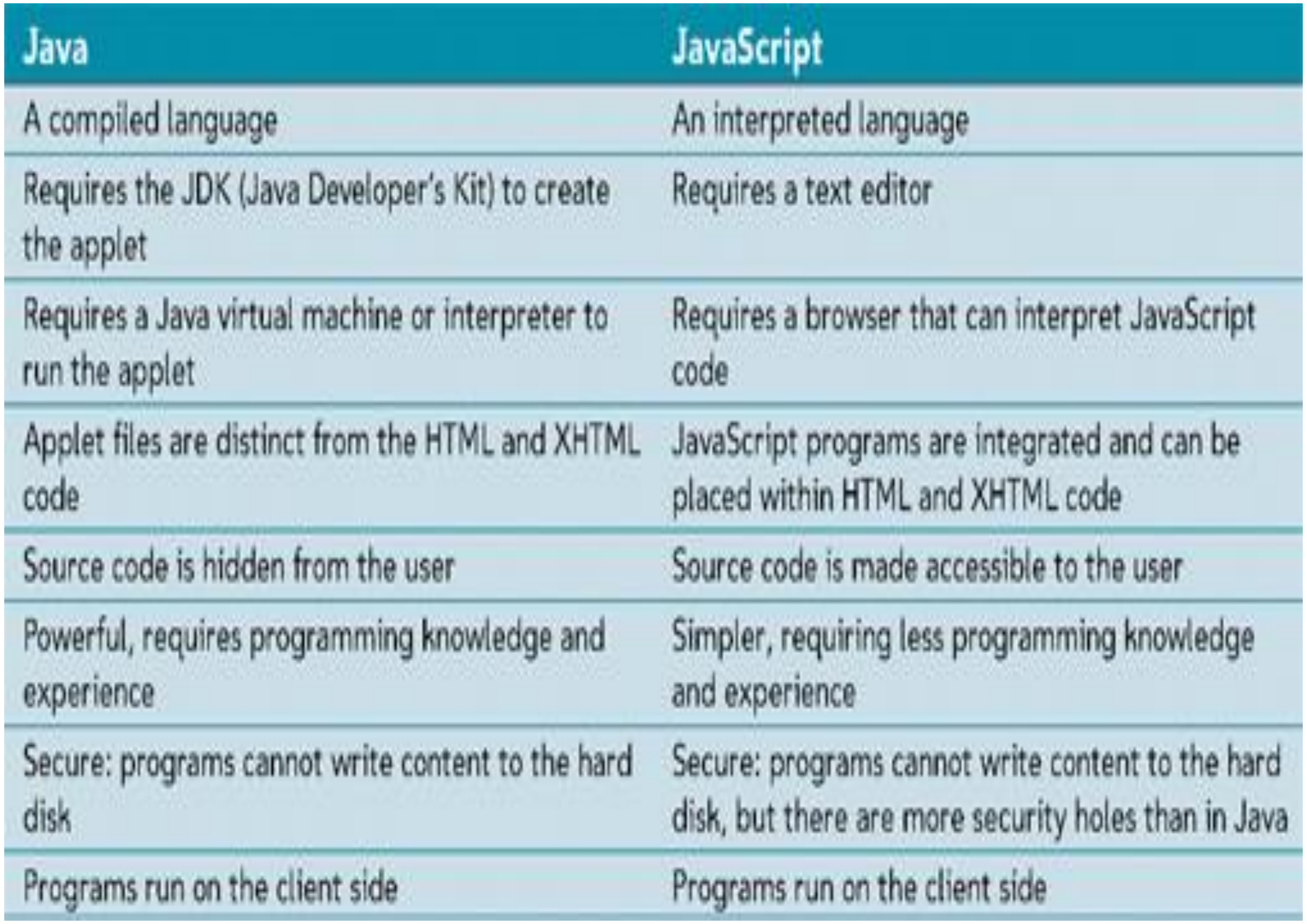
Introduction to JavaScript

- For that we use scripting languages which allows content to change dynamically.
- JavaScript is Case Sensitive.
- Object based programming language
  - very limited object creation

Comparing Java and JavaScript

- Java and JavaScript are two completely different languages in both concept and design
- Java (developed by Sun Microsystems) is a powerful and much more complex programming language.
- Java is a **compiled language**
- JavaScript is a subset of Java
- JavaScript can put dynamic text into an HTML page
- JavaScript is an **interpreted language**

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
Secure: programs cannot write content to the hard disk	Secure: programs cannot write content to the hard disk, but there are more security holes than in Java
Programs run on the client side	Programs run on the client side

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:

1. Client Side JavaScript

2. Server 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.

Advantages of ClientSide Scripting

- The Web browser uses it’s own resources, and erase the burden on the server.
- it has fewer feature than server side language.

DisAdvantages of ClientSide Scripting

- Code is usually visible
- Code is Probably modifiable
- Local files and database can’t be accessed

JavaScript Types:

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 File

- 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 File

- Scripts can be provided locally or remotely accessible JavaScript file using *src attribute.*

```
<html>
        <head>
        <script language="JavaScript"
        type="text/javascript"
        src="http://somesite/myOwnJavaScript.js">
        </script>
        </head>
</html>
```

Writing Output to the Web Page

- An **object-oriented** programming language writes the output by manipulating tasks.
- An action you perform on an object is called a **method**
- To write text on Web page, use following JavaScript commands:

  document.write("*text*");   Or   document.writeln("*text*")'

  Where *text* is the content to be written to the page. The doucment.write() and document.writeln() methods are identical, except that the document.writeln() method preserves any line breaks in the text string.

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 scripts and functions on the web page can access it.

Working with Variables and Data

- Global variables are destroyed when you close the page.
- If you declare a variable, without using "**var**", the variable always becomes **GLOBAL**.
- **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

Declaring a JavaScript Variable

- You can declare variables with any of the following JavaScript commands:

  var *variable*;

  var *variable* = *value*;

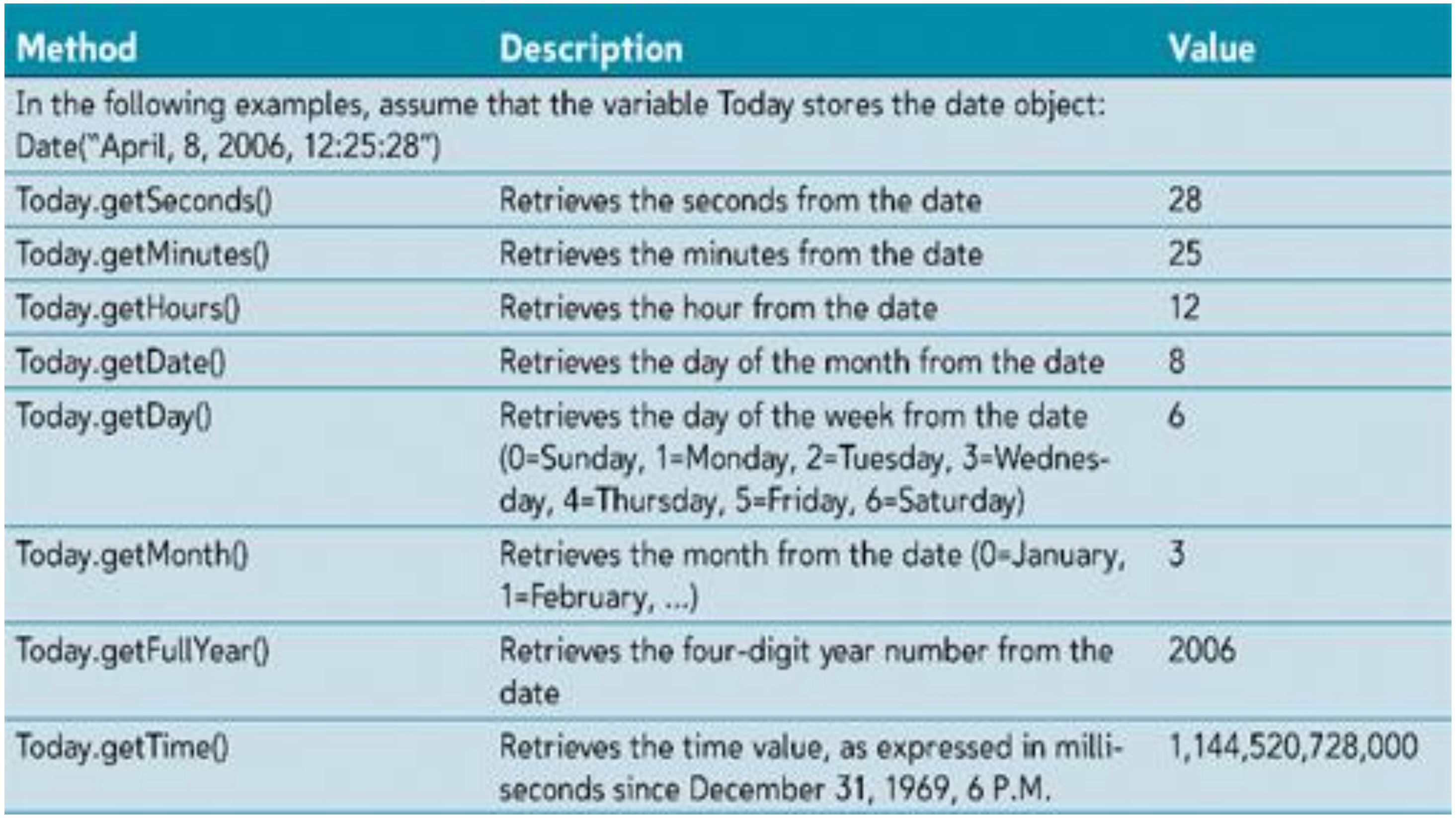
  ***variable* = *value*;**

  Where *variable* is the name of the variable and *value* is the initial value of the variable. The first command creates the variable without assigning it a value; the second and third commands both create the variable and assign it a value.

Working with Dates

- Create a **date object** to store date information

Date Methods



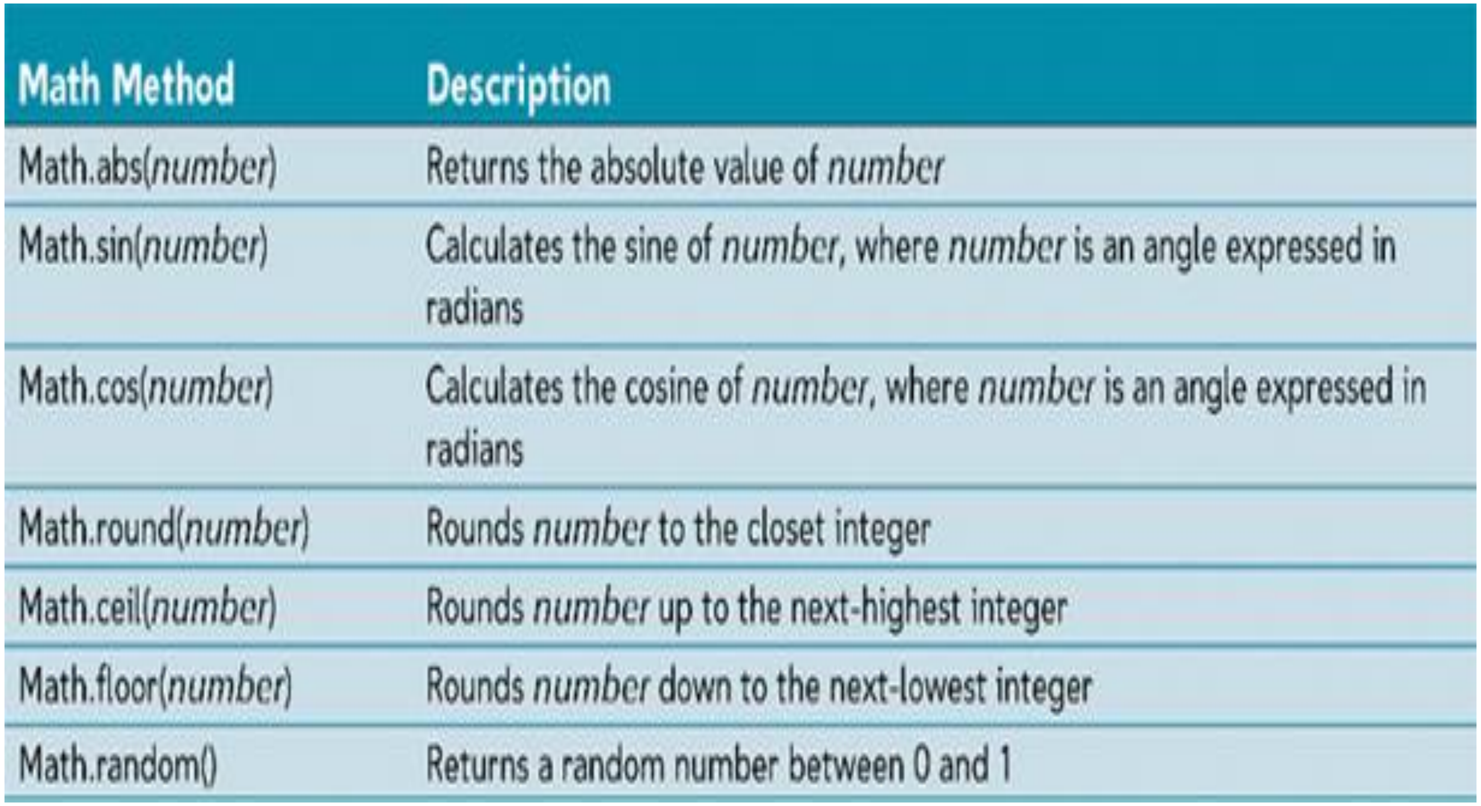
Method	Description	Value
In the following examples, assume that the variable Today stores the date object: Date("April, 8, 2006, 12:25:28")		
Today.getSeconds()	Retrieves the seconds from the date	28
Today.getMinutes()	Retrieves the minutes from the date	25
Today.getHours()	Retrieves the hour from the 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

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 by 1
  - Assignment operators: used to assign values in expressions

Working with Expressions and Operators

- 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

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

-
  ```
  <html>
  <head>
  <script type="text/javascript">
  function displaymessage()
  {
  alert("Hello World!");
  }
  </script>
  </head>
  <body>    <form>
  <input type="button" value="Click me!"
  onclick="displaymessage()" />
  </form>
  </body>
  </html>
  ```

Working with Conditional Statements

- **Conditional statements** are commands that run only when specific conditions are met
- Conditional statements require a **Boolean expression**
  – you need one of the following operators to create a Boolean expression:
    - Comparison operator
    - Logical operator (&& , || , !)
    - Conditional operator

      variablename=(condition)?value1:value2

Using Arrays

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

  var *variable* = new Array (*size*);

Where ***variable*** is the name of the array variable and

size is the number of elements in the array

Working with Program Loops

- A **program loop** is a set of instructions that is executed repeatedly
  - The loop uses a **counter** to track the number of times the command block has been run
  - 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.

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.

JavaScript Popup Boxes

- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

  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

Alert Box Example

```
<html>
    <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>
</html>
```

JavaScript Popup Boxes

Confirm Box Example

```
<html>
<head>
<script
type="text/javascript">
function show_confirm()
{
var r=confirm("Press a
button");
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 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>
```

Debugging Your JavaScript Programs

- Three types of errors:
  - Load-time errors (occurs when the script is loading)
  - Run-time errors (occurs when the being executed)
  - Logical errors (free from syntax and structural mistakes, but result in incorrect results)

Common Mistakes

- You need to **debug** your program to fix the mistakes
- Common mistakes include:
  - Misspelling a variable name
  - Mismatched parentheses or braces
  - Mismatched quotes
  - Missing quotes
  - Using (instead of [
  - Using = in place of ==

JavaScript Events

- Every element on a web page has certain events which can trigger invocation of event handlers
- Attributes are inserted into HTML tags to define events and event handlers

Examples of events

- A mouse click
- A web page or an image loading
- Mousing over a hot spot on the web page
- Selecting an input box in an HTML form
- Submitting an HTML form
- A keystroke

JavaScript Events

- onabort - Loading of an image is interrupted
- onblur - An element loses focus
- onchange - The content of a field changes
- onclick - Mouse clicks an object
- ondblclick - Mouse double-clicks an object
- onerror - An error occurs when loading a document or an image
- onfocus - An element gets focus
- onkeydown - A keyboard key is pressed

JavaScript Events

- onkeypress - A keyboard key is pressed
- onkeyup - A keyboard key is released
- onload - A page or an image is finished loading
- onmousedown - A mouse button is pressed
- onmousemove - The mouse is moved
- onmouseout - The mouse is moved off an element
- onmouseover - The mouse is moved over an element
- onmouseup - A mouse button is released
- onreset - The reset button is clicked

JavaScript Events

- onresize - A window or frame is resized
- onselect - Text is selected
- onsubmit - The submit button is clicked
- onunload - The user exits the page

JavaScript Events

onload & onUnload Events

- The *onload and onUnload events are triggered when* the user enters or leaves the page
- Both the onload and onUnload eThe onload event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information
- vents are also often used to deal with cookies that should be set when a user enters or leaves a page.

JavaScript Events

onFocus, onBlur and onChange

- The onFocus, onBlur and onChange events are often used in combination with validation of form fields.

Example: The *checkEmail() function will be called* whenever the user changes the content of the field:

<input type="text" size="30"

id="email" onchange="*checkEmail()"*>*;*

JavaScript Events

Example : onblur

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

JavaScript Events

onSubmit

- The *onSubmit event is used to validate all form* fields before submitting it.

  Example: The *checkForm() function will be called* when the user clicks the submit button in the form. If the field values are not accepted, the submit should be canceled.

  The function *checkForm() returns* either true or false. If it returns true the form will be submitted, otherwise the submit will be cancelled:

<form method="post" action="xxx.html“onsubmit=“c*heckForm()"*>

JavaScript Events

```
<html>   <head>
<script type="text/javascript">
function validate()
{
// return true or false based on validation logic
}
</script>  </head>
<body>
<form name=“f1” action="tryjs_submitpage.htm" onsubmit="validate()">

Name (max 10 chararcters): <input type="text" id="fname" size="20"><br />
Age (from 1 to 100): <input type="text" id="age" size="20"><br />
E-mail: <input type="text" id="email" size="20"><br /> <br />
<input type="submit" value="Submit">
</form>   </body>  </html>
```

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 Object

A JavaScript object has properties and methods

Example: *String JavaScript object has* ***length*** *property and* ***toUpperCase()*** *method*

```
<script type="text/javascript">
        var txt="Hello World!“
        document.write(txt.length)
        document.write(txt.toUpperCase())
</script>
```

String , Date , Array , Boolean and Math are Built in javascript objects.

JavaScript Object

Properties

- Properties are object attributes.
- Object properties are defined by using the object's name and the property name.
  - e.g., background color is expressed by: **`document.bgcolor`** .
  - **`document`** is the object.
  - **`bgcolor`** is the property.

JavaScript Object

Method

In Javascript method is function that is invoked through object

To invoke method

```
myobject.method();

myobject.method(x,y,z);
```

Exa.

```
var Today= new Date();

Today.getDay();
```

The `document` object

- Each HTML document loaded into a browser window becomes a **Document** object.
- Many attributes of the current document are available via the **`document`** object:
- Document object represents the HTML displayed in the window

  document.bgcolor=red

  document.linkcolor=yellow

  document.write (“<h2> Hello World </h2>);

  we can access any form information in a document by using the **formname.**

The `document` object

Example

<form name=“userdetails”>

<input type=“text” name=“fname”/>

<input type=“text” name=“lname”/>

<input type=“submit” name=“submit”/>

</form>

document.formname

it can be referred by **document.userdetails**

any element within it can accessed by

document.userdetails.fname.value

Window object

Window - Built-in Properties

- ‘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

– **closed** - A boolean value that indicates whether the window is closed. **window.closed()**

– **defaultStatus** - This is the default message that is loaded into the status bar when the window loads.

window.defaultStatus = “This is the status bar”;

-

open:

window.open()

```
open("URLname","Windowname",["options"])
```

```
open(“http://google.com”, “My Google”,”height,width”);
```

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

frames

navigator

location

history

document

screen

- **frames**: The frames property returns an array of all the frames (including iframes) in the current window.

NAVIGATOR OBJECT

navigator : 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

NAVIGATOR OBJECT

```
<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>
```

- Browser Name is: Netscape
  Browser Code is: Mozilla
  Browser Version No is: 5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36
  Platform is: Win32
  User agent Header: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36
  Cookies Enabled is: true

SCREEN OBJECT

- **screen** : it will give information about size of user's display and color depth.

Property	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)
colorDepth	Returns the bit depth of the color palette for displaying images
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

Location Object

- The location object contains information about the current URL.
- The location object is part of the window object and is accessed through the **window.location** property.
- This differs from the document object because the document is the real content; the location is simply the URL.
- URL consists of many components that define the address and method of data transfer for a file.
- Pieces of a URL include the protocol (such as http:) and the hostname (such as www.giantco.com).
- Setting the **location.href** property is the primary way your scripts navigate to other pages: **location.href = “http://www.dannyg.com”**

Location Object

Property	Description
host	Returns the hostname and port of a URL
hostname	Returns the hostname of a URL
href	Returns the entire URL
pathname	Returns the path name of a URL
port	Returns the port number the server uses for a URL
protocol	Returns the protocol of a URL
search	Returns the query portion of a URL

History Object

- The history object contains the URLs visited by the user (within a browser window).
- Each window maintains a list of recent pages that the browser has visited.
- While the history object’s list contains the URLs of recently visited pages, those URLs are not generally accessible by script due to privacy and security limits imposed by browsers.
- Methods of the history object allow for navigating backward and forward through the history relative to the currently loaded page.

History Object

Property	Description
length	Returns the number of URLs in the history list
Method	**Description**
back()	Loads the previous URL in the history list
forward()	Loads the next URL in the history list
go()	Loads a specific URL from the history list

History Object

```
<html>
<head>
<script>
function goBack()
  {
  window.history.go(-2);
  }
</script>
</head>
<body>
<input type="button" value="Go back 2 pages" onclick="goBack()">
</body>
</html>
```

The Link Object

- A link object is the object model equivalent of an <A> tag when the tag includes an HREF attribute.
- When you want to click a link to execute a script rather than navigate directly to another URL, you can redirect the HREF attribute to call a script function.
- The technique called as **javascript:** URL.
- If you place the name of a function after the **javascript:** URL, then a scriptable browser runs that function.

The Link Object

- The function should probably perform some navigation in the end.

  The syntax for this construction in a link is as follows:
- **<A HREF=”javascript:void”**

  onClick=“*functionName([parameter1]..[parameterN])”*>   <*/A*>
- The void keyword prevents the link from trying to display any value that the function may return.
- javascript: URL technique for all tags that include HREF. If an attribute accepts a URL, it can accept this javascript: URL .

DOM & Web Browser Environment

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.

DOM History and Levels

A simple DOM was implemented in Netscape 2.0 browser. **write()** method is used.

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.

DOM Tree

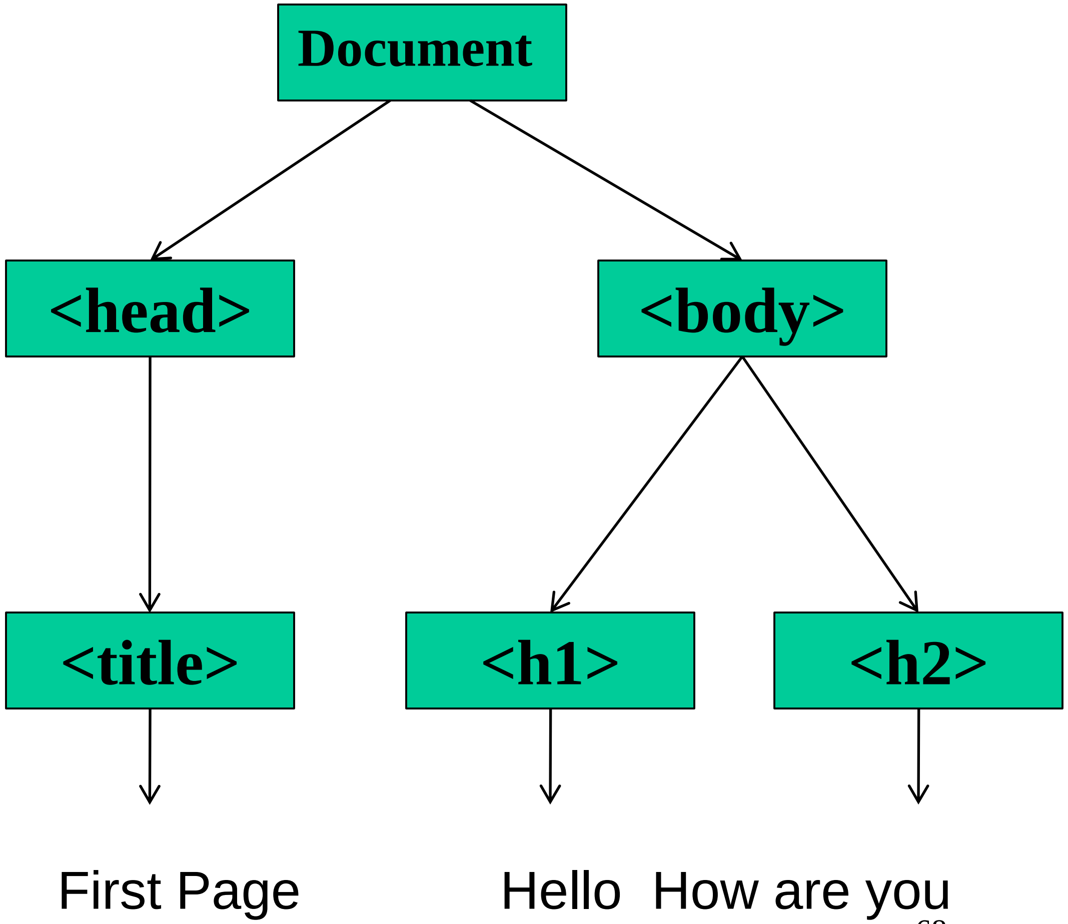
The document in DOM are represented using a tree like structure in which every element is represented as a node.

Basic terminologies used in DOM tree as follows:

1. Every element in the DOM tree is called node.
2. The topmost single node in the DOM tree is called root.
3. Every child node must have parent node.
4. The bottommost nodes that have no children are called leaf nodes.
5. The nodes that have the common parent are called siblings.

DOM Tree

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



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.form1.fname.value;
if (x==null || x=="")
  {
  alert("First name must be filled out");
   document.form1.fname.focus();
  return false;
  }
}
```

Forms And Validation

- **Validating User:**

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

Forms And Validation

- **Validating Password & Confirm password field**

```
 function f1()
{
if(document.form1.pass.value != document.form1.cpass.value)
    {
            alert('Confirm Password Not Match');
            document.form1.cpass.focus();
            document.form1.cpass.select;
            return false;
    }
}
```

Forms And Validation

E-mail Validation

- The function below checks if the content has the general syntax of an email.
- 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

abc_123@yahoo.co.in

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

Forms And Validation

Email Validation using Regular Expression:

```
 function f1(){
     var check=document.form1.email.value;
     Document.write(testmail(check));
}
 function testmail(chkmail){
     var emailpattern=“^[\\w-_\.]*[\\w-_\.]\@[\\w]\.+[\\w]+[\\w]$”;
     var regex = new RegExp(emailpattern); // Regular Expression is created
     return regex.test(chkmail); // tested against incoming parameters
}
```

Forms And Validation

The first section is: ^[\\w-_\.]

This sequence begin with ^. Means check the first character is a word represented by **\\w**. it can be also _, - which normally not used but are legal.

The second section is: *[\\w-_\.]

The * means that the next series of characters described can be repeated many times of not at all.

The third section is: \@[\\w]\.+

This section begin with @ character. Followed by word character and then at least one dot (.).

The last part is: [\\w]+[\\w]$

There are some character after last dot (.).

Forms And Validation

Text-related objects

- Four text-related HTML form elements—text, password and TEXTAREA—is an element in the document object hierarchy.
- The single most used property of a text-related element is the **value** property. This property represents the current contents of the text element.
- A script can retrieve and set its content at any time.

```
        var field = document.formname.converter.value
        field.value = “abc”
<INPUT TYPE=”text” NAME=”converter” VALUE=”sample”>
```

Forms And Validation

The Checkbox Object

- The key property of a checkbox object is whether or not the box is checked.
- The checked property is a Boolean value: true if the box is checked, false if not.

```
<INPUT TYPE=”checkbox” NAME=”checkThis”>Check here<BR>
if (document.form1.checkThis.checked)
{
alert(“The box is checked.”)
} else
{
alert(“The box is not checked at the moment.”)
}
```

Forms And Validation

The Radio Object

- The key property of a Radio object is whether or not the box is checked.
- The checked property is a Boolean value: true if the box is checked, false if not.

```
<FORM>
<B>Select your favorite Color:</B>
<INPUT TYPE=”radio” NAME=”Color” VALUE=”Red” CHECKED />Red
<INPUT TYPE=”radio” NAME=”Color” VALUE=”Pink”  />Pink
<INPUT TYPE=”radio” NAME=”Color” VALUE=”Yellow”  />Yellow<BR>
</FORM>
```

Accessing a radio button value

```
formname.color.checked
```

Forms And Validation

Minimum & Maximum Characters for Password field:

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

Forms And Validation

Validating the Mobile Number:

```
function validate() {
        var mobile = document.getElementById("mobile").value;
        var pattern = “/^\d{10}$/”;
        if (pattern.test(mobile)) {
          alert("Your mobile number : "+mobile);
          return true;
        }
          alert("It is not valid mobile number.input 10 digits number!");
          return false;
      }
```

Forms And Validation

Validating Alphabetic Field:

```
var alphaExp = "/^[a-zA-Z]+$/";
if(document.form1.fname.value.match(alphaExp))
{
return true;
}
else{
alert("Letters only please!!!!!");
document.form1.fname.focus();
return false;}}
```

Forms And Validation

The SELECT Object

- The SELECT object is really a compound object: an object that contains an array of OPTION objects.
- Some properties belong to the entire SELECT object; others belong to individual options inside the SELECT object.
  - document.form1.*selectName.value*

    This value is the index number of the currently selected item.
- the first item has an index of zero.

Forms And Validation

```
<script language="javascript">
 function show()
{
 var value = document.form1.s.value
 alert("You have selected "+value);
 }
 </script>
```

```
<form name="form1">
            <select name="s"
onchange="show();">
  <option value=Red> Red </option>
  <option value=Yellow> Yellow
</option>
  <option value=Blue> Blue </option
  <option value=Black> Black
</option>
  <option value=White> White
</option>
            </select>  </form>
```

Forms And Validation

Radio Button Value:

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

Forms And Validation

String Functions

Concatnation

var msg = “Four score” + “ and seven”

Changing string case

var result = string.toUpperCase()

var result = string.toLowerCase()

String searches

indexOf() function returns the position of a character in a string.
Strpos=stringname.indexOf(“@”);

which Char=stringName.charAt(index)

Example :

var stringA = “Building C”

- var bldgLetter = stringA.charAt(9) // result: bldgLetter = “C”

Forms And Validation

- Processing of submitted information on the client side is advantage in terms of resources – by not sending the data over to the server
- JavaScript can be used to validate data in HTML forms before sending off the content to a server.

  Form data that typically are checked by a JavaScript could be:

  has the user left required fields empty?

  has the user entered a valid e-mail address?

  has the user entered a valid date?

  has the user entered text in a numeric field?

Timing Event

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 are:

setTimeout() - executes a code some time in the future

clearTimeout() - cancels the setTimeout()

The setTimeout() Method

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

The setTimeout() method returns a value.

To get a timer to work in infinite loop, write a function that calls itself.

Timing Event

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

Timing Event

```
<!DOCTYPE html>
<html>
<body onload="startTime()">
<div id="txt"></div>
<script>
function startTime() {
    var today = new Date();
    var h = today.getHours();
    var m = today.getMinutes();
    var s = today.getSeconds();
    // add a zero in front of numbers<10
    m = checkTime(m);
    s = checkTime(s);
document.getElementById("txt").in
     nerHTML = h + ":" + m + ":" + s;
    var t = setTimeout(function(){
     startTime() }, 500);
}

function checkTime(i) {
    if (i < 10) {
        i = "0" + i;
    }
    return i;
}
</script>
```

Timing Event

JavaScript Timing Events

```
 <html> <head>
<script type="text/javascript">
var c=0;   var t;    var timer_is_on=0;

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

Timing Event

JavaScript Timing Events

The clearTimeout() Method

clearTimeout(*setTimeout_variable*)

See Timing Clock Example

Timing Event

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

Event, Attributes & Tags

Events	Event Attributes	Meaning	Associated Tags
Blur	onblur	Losing the focus	<button> <input> <a> <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>

Event, Attributes & Tags

Events	Event Attributes	Meaning	Associated Tags
Focus	onfocus	When user acquires the input focus	<a> <input> <select> <textarea>
Keyup	onkeyup	When user releases the key from the keyboard	Form Element
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

Event, Attributes & Tags

Events	Event Attributes	Meaning	Associated Tags
Mousemove	onmousemove	when user move the mouse	Form Elements
Mouseout	onmouseout	when user moves the mouse away from some element	Form Elements
Mouseover	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>

Event, Attributes & Tags

Events	Event Attributes	Meaning	Associated Tags
Select	onselect	On selection	<input> <textarea>
Unload	onunload	When user exits the document	<body>
Onerror	onerror	An error occurred when loading a document or an image	<body>

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. For example one user registration ends after completing many pages. But 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, commissions, 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.

How It Works ?

- 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.

Types

- There are two types of cookies:
  - **Session Cookies**
    - A browser stores session cookies in memory.
    - Once a browser session ends, browser loses the contents of a session cookie.
  - **Persistent Cookies**
    - Browsers store persistent cookies to a user's hard drive.
    - We can use persistent cookies to customize information about a user that we can use when the user returns to a website at a later date.

Cookie Variables

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.

Cookies as Objects

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

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?");
```

<u>Setting a Cookie – General Form</u>

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

Setting or Storing a Cookie

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

Setting or Storing a Cookie

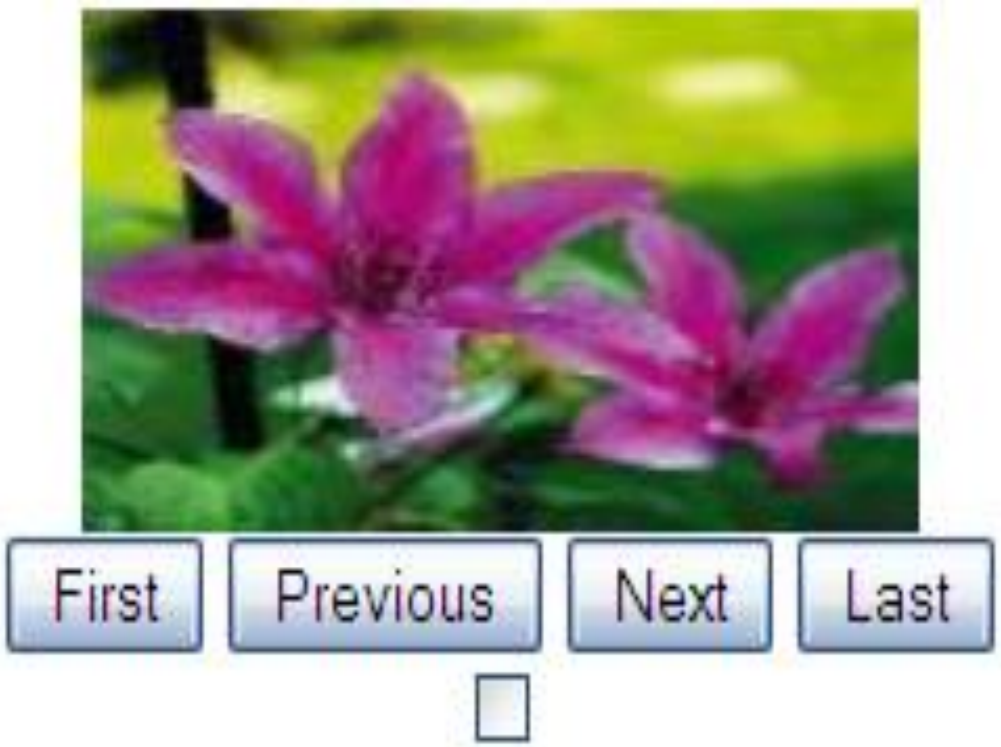
Cookie values may not include semicolons, commas, or whitespace.

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

DHTML (Image Viewer)

```
<body>
<center>
<img id="imageviewer" title="abc" src="anim0.jpg" alt="w3Scools"
    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" value="Automatic"
    onClick="automaticly()">
</form></center></body></html>
```

DHTML (Image Viewer)



DHTML (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]="Desert.jpg";
myImages[5]="Jellyfish.jpg";
myImages[6]="Koala.jpg";
myImages[7]="Penguins.jpg";
myImages[8]="Waterfall.jpg";

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

DHTML (Image Viewer)

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

DHTML (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];
    }}
```

DHTML (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>
```