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

        document.write("Hello World!")


        for (i=1; i<=5; i++)
        {
        document.write(i + "<br/>")
        }

        </script>
</body>
```

# Variable declaration

Variable

```
var x = 10;
var y = 20;
var z=x+y;
document.write(z);
```

# Input from user

```
var person = prompt("Please enter your name", "Harry Potter");

if (person != null) {
      document.getElementById("demo").innerHTML =
    "Hello " + person + "! How are you today?";
}
```

Global varibale\

```
1. var value=50;//global variable
2. function a(){
3. alert(value);
4. }
5. function b(){
6. alert(value);
7. }
```

For Loop

```
1. for (i=1; i<=5; i++)
2. {
```

3. document.write(i + "<br/>")
4. }

While LOOP

5. var i=11;
6. while (i<=15)
7. {
8. document.write(i + "<br/>");
9. i++;
10. }

Do while

1. var i=21;
2. do{
3. document.write(i + "<br/>");
4. i++;
5. }while (i<=25);
6.

DATA tYPE

Data TypeDescription

String represents sequence of characters e.g. "hello"

Number represents numeric values e.g. 100

Boolean represents boolean value either false or true

Undefined represents undefined value ]

Null represents null i.e. no value at all

## JavaScript in External File :

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.......
</body>
</html>
```

To use JavaScript from an external file source, you need to write your all JavaScript source code in a simple text file with extension ".js" and then include that file as shown above.

For example, you can keep following content in filename.js file and then you can use *sayHello* function in your HTML file after including filename.js file:

```
function sayHello() {
   alert("Hello World")
}
```

# JavaScript Variable Scope:

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

- **Global Variables:** A global variable has global scope which means it is defined everywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable
function checkscope( ) {
   var myVar = "local";  // Declare a local variable
   document.write(myVar);
}
//-->
</script>
```

# if...else statement:

```
if (expression){
   Statement(s) to be executed if expression is true
}else{
Statement(s) to be executed if expression is false
}
```

```
<script type="text/javascript">
<!--
var age = 15;
if( age > 18 ){
   document.write("<b>Qualifies for driving</b>");
}else{
   document.write("<b>Does not qualify for driving</b>");
}
//-->
</script>
```

# if...else if... statement:

```
<script type="text/javascript">
<!--
var book = "maths";
```

```
if( book == "history" ){
   document.write("<b>History Book</b>");
}else if( book == "maths" ){
   document.write("<b>Maths Book</b>");
}else if( book == "economics" ){
   document.write("<b>Economics Book</b>");
}else{
  document.write("<b>Unknown Book</b>");
}
//-->
</script>
```

Switch Case:

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
  case 'A': document.write("Good job<br />");
            break;
  case 'B': document.write("Pretty good<br />");
            break;
  case 'C': document.write("Passed<br />");
            break;
  case 'D': document.write("Not so good<br />");
            break;
  case 'F': document.write("Failed<br />");
            break;
  default:  document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
```

# Window Object

## Window alert() Method

```
alert("Hello! I am an alert box!!");
alert(location.hostname);
```

## Window confirm() Method

```
confirm("Press a button!");
```

e.g:

```
var txt;
var r = confirm("Press a button!");
if (r == true) {
      txt = "You pressed OK!";
} else {
      txt = "You pressed Cancel!";
}
document.write(txt);
```

# Function

```
<script type="text/javascript">
   <!--
      function functionname(parameter-list)
      {
         statements
      }
   //-->
</script>
```

## Example

```
<script type="text/javascript">
   <!--
      function sayHello()
      {
         alert("Hello there");
      }
   //-->
</script>
```

# Calling a Function

```
<html>
   <head>

      <script type="text/javascript">
         function sayHello()
         {
            document.write ("Hello there!");
         }
      </script>

   </head>
   <body>
      <p>Click the following button to call the function</p>

      <form>
         <input type="button" onclick="sayHello()" value="Say Hello">
      </form>

      <p>Use different text in write method and then try...</p>
   </body>
</html>
```

# Function Parameters

```
<html>
   <head>

      <script type="text/javascript">
         function sayHello(name, age)
         {
            document.write (name + " is " + age + " years old.");
         }
      </script>

   </head>
   <body>
      <p>Click the following button to call the function</p>

      <form>
         <input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
      </form>

      <p>Use different parameters inside the function and then try...</p>
```

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

# The return Statement

```
<html>
   <head>

      <script type="text/javascript">
         function concatenate(first, last)
         {
            var full;
            full = first + last;
            return full;
         }

         function secondFunction()
         {
            var result;
            result = concatenate('Zara', 'Ali');
            document.write (result );
         }
      </script>

   </head>

   <body>
      <p>Click the following button to call the function</p>

      <form>
         <input type="button" onclick="secondFunction()" value="Call Function">
      </form>

      <p>Use different parameters inside the function and then try...</p>

  </body>
</html>
```

# Nested Functions

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function hypotenuse(a, b) {
               function square(x) { return x*x; }
```

```
               return Math.sqrt(square(a) + square(b));
            }

            function secondFunction(){
               var result;
               result = hypotenuse(1,2);
               document.write ( result );
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following button to call the function</p>

      <form>
         <input type="button" onclick="secondFunction()" value="Call Function">
      </form>

      <p>Use different parameters inside the function and then try...</p>

   </body>
</html>
```

# The Function() Constructor

The *function* statement is not the only way to define a new function; you can define your function dynamically using Function() constructor along with the new operator.

The Function() constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons.

The Function() constructor expects any number of string arguments. The last argument is the body of the function – it can contain arbitrary JavaScript statements, separated from each other by semicolons.

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            var func = new Function("x", "y", "return x*y;");
            function secondFunction(){
               var result;
               result = func(10,20);
               document.write ( result );
            }
```

```
         //-->
      </script>

   </head>
   <body>
      <p>Click the following button to call the function</p>

      <form>
         <input type="button" onclick="secondFunction()" value="Call Function">
      </form>

      <p>Use different parameters inside the function and then try...</p>

   </body>
</html>
```

# Function Literals

JavaScript 1.2 introduces the concept of function literals which is another new way of defining functions. A function literal is an expression that defines an unnamed function.

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            var func = function(x,y){ return x*y };

            function secondFunction(){
               var result;
               result = func(10,20);
               document.write ( result );
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following button to call the function</p>

      <form>
         <input type="button" onclick="secondFunction()" value="Call Function">
      </form>

      <p>Use different parameters inside the function and then try...</p>

   </body>
</html>
```

# Events

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

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

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

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

# onclick Event Type

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function sayHello() {
               alert("Hello World")
            }
         //-->
      </script>

   </head>

   <body>
      <p>Click the following button and see result</p>

      <form>
         <input type="button" onclick="sayHello()" value="Say Hello" />
      </form>

   </body>
</html>
```

# onsubmit Event type

```
<html>
   <head>

      <script>

     function validateForm() {

    var x = document.forms["myForm"]["fname"].value;

    if (x == "") {

        alert("Name must be filled out");

        return false;

    }

}


      </script>


   </head>
   <body>


     <form name="myForm" action="login.html"q

onsubmit="return validateForm()" method="post"📯>

Name: <input type="text" name="fname">

<input type="submit" value="Submit">

</form>


   </body>


   </body>
```

```
</html>
```

# onmouseover and onmouseout

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function over() {
               document.write ("Mouse Over");
            }

            function out() {
               document.write ("Mouse Out");
            }

         //-->
      </script>

   </head>
   <body>
      <p>Bring your mouse inside the division to see the result:</p>

      <div onmouseover="over()" onmouseout="out()">
         <h2> This is inside the division </h2>
      </div>

   </body>
</html>
```

# HTML 5 Standard Events

| Attribute | Value | Description |
|---|---|---|
| Offline | script | Triggers when the document goes offline |
| Onabort | script | Triggers on an abort event |

| | | |
|---|---|---|
| onafterprint | script | Triggers after the document is printed |
| onbeforeonload | script | Triggers before the document loads |
| onbeforeprint | script | Triggers before the document is printed |
| onblur | script | Triggers when the window loses focus |
| oncanplay | script | Triggers when media can start play, but might has to stop for buffering |
| oncanplaythrough | script | Triggers when media can be played to the end, without stopping for buffering |
| onchange | script | Triggers when an element changes |
| onclick | script | Triggers on a mouse click |
| oncontextmenu | script | Triggers when a context menu is triggered |
| ondblclick | script | Triggers on a mouse double-click |
| ondrag | script | Triggers when an element is dragged |
| ondragend | script | Triggers at the end of a drag operation |
| ondragenter | script | Triggers when an element has been dragged to a |

| | | |
|---|---|---|
| | | valid drop target |
| ondragleave | script | Triggers when an element is being dragged over a valid drop target |
| ondragover | script | Triggers at the start of a drag operation |
| ondragstart | script | Triggers at the start of a drag operation |
| ondrop | script | Triggers when dragged element is being dropped |
| ondurationchange | script | Triggers when the length of the media is changed |
| onemptied | script | Triggers when a media resource element suddenly becomes empty. |
| onended | script | Triggers when media has reach the end |
| onerror | script | Triggers when an error occur |
| onfocus | script | Triggers when the window gets focus |
| onformchange | script | Triggers when a form changes |
| onforminput | script | Triggers when a form gets user input |
| onhaschange | script | Triggers when the document has change |

| | | |
|---|---|---|
| oninput | script | Triggers when an element gets user input |
| oninvalid | script | Triggers when an element is invalid |
| onkeydown | script | Triggers when a key is pressed |
| onkeypress | script | Triggers when a key is pressed and released |
| onkeyup | script | Triggers when a key is released |
| onload | script | Triggers when the document loads |
| onloadeddata | script | Triggers when media data is loaded |
| onloadedmetadata | script | Triggers when the duration and other media data of a media element is loaded |
| onloadstart | script | Triggers when the browser starts to load the media data |
| onmessage | script | Triggers when the message is triggered |
| onmousedown | script | Triggers when a mouse button is pressed |
| onmousemove | script | Triggers when the mouse pointer moves |
| onmouseout | script | Triggers when the mouse pointer moves out of an |

| | | |
|---|---|---|
| | | element |
| onmouseover | script | Triggers when the mouse pointer moves over an element |
| onmouseup | script | Triggers when a mouse button is released |
| onmousewheel | script | Triggers when the mouse wheel is being rotated |
| onoffline | script | Triggers when the document goes offline |
| onoine | script | Triggers when the document comes online |
| ononline | script | Triggers when the document comes online |
| onpagehide | script | Triggers when the window is hidden |
| onpageshow | script | Triggers when the window becomes visible |
| onpause | script | Triggers when media data is paused |
| onplay | script | Triggers when media data is going to start playing |
| onplaying | script | Triggers when media data has start playing |
| onpopstate | script | Triggers when the window's history changes |

| | | |
|---|---|---|
| onprogress | script | Triggers when the browser is fetching the media data |
| onratechange | script | Triggers when the media data's playing rate has changed |
| onreadystatechange | script | Triggers when the ready-state changes |
| onredo | script | Triggers when the document performs a redo |
| onresize | script | Triggers when the window is resized |
| onscroll | script | Triggers when an element's scrollbar is being scrolled |
| onseeked | script | Triggers when a media element's seeking attribute is no longer true, and the seeking has ended |
| onseeking | script | Triggers when a media element's seeking attribute is true, and the seeking has begun |
| onselect | script | Triggers when an element is selected |
| onstalled | script | Triggers when there is an error in fetching media data |
| onstorage | script | Triggers when a document loads |

| | | |
|---|---|---|
| onsubmit | script | Triggers when a form is submitted |
| onsuspend | script | Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched |
| ontimeupdate | script | Triggers when media changes its playing position |
| onundo | script | Triggers when a document performs an undo |
| onunload | script | Triggers when the user leaves the document |
| onvolumechange | script | Triggers when media changes the volume, also when volume is set to "mute" |
| onwaiting | script | Triggers when media has stopped playing, but is expected to resume |

# JavaScript - Objects

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers –

- Encapsulation – the capability to store related information, whether data or methods, together in an object.
- Aggregation – the capability to store one object inside another object.
- Inheritance – the capability of a class to rely upon another class (or number of

classes) for some of its properties and methods.

- Polymorphism – the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

# Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is –

```
objectName.objectProperty = propertyValue;
```

For example – The following code gets the document title using the "title"property of the document object.

```
var str = document.title;
```

# Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the this keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local

properties and parameters.

For example – Following is a simple example to show how to use the write() method of document object to write any content on the document.

```
document.write("This is test");
```

# User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called Object.

## The new Operator

The new operator is used to create an instance of an object. To create an object, the new operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

## The Object() Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called Object() to build the object. The return value of the Object() constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the var keyword.

## Example 1

Try the following example; it demonstrates how to create an Object.

```
<html>
   <head>
      <title>User-defined objects</title>

      <script type="text/javascript">
         var book = new Object();   // Create the object
         book.subject = "Perl"; // Assign properties to the object
         book.author  = "Mohtashim";
      </script>

   </head>

   <body>

      <script type="text/javascript">
         document.write("Book name is : " + book.subject + "<br>");
         document.write("Book author is : " + book.author + "<br>");
      </script>

   </body>
</html>
```

## Output

```
Book name is : Perl
Book author is : Mohtashim
```

## Example 2

This example demonstrates how to create an object with a User-Defined Function. Here this keyword is used to refer to the object that has been passed to a function.

```
<html>
   <head>

   <title>User-defined objects</title>

      <script type="text/javascript">
         function book(title, author){
            this.title = title;
            this.author  = author;
         }
      </script>

   </head>
```

```
   <body>

      <script type="text/javascript">
         var myBook = new book("Perl", "Mohtashim");
         document.write("Book title is : " + myBook.title + "<br>");
         document.write("Book author is : " + myBook.author + "<br>");
      </script>

   </body>
</html>
```

## Output

```
Book title is : Perl
Book author is : Mohtashim
```

# Defining Methods for an Object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

## Example

Try the following example; it shows how to add a function along with an object.

```
<html>
   <head>
   <title>User-defined objects</title>

      <script type="text/javascript">
         // Define a function which will work as a method
         function addPrice(amount){
            this.price = amount;
         }

         function book(title, author){
            this.title = title;
            this.author  = author;
            this.addPrice = addPrice; // Assign that method as property.
         }
      </script>

   </head>
```

```
   <body>

      <script type="text/javascript">
         var myBook = new book("Perl", "Mohtashim");
         myBook.addPrice(100);

         document.write("Book title is : " + myBook.title + "<br>");
         document.write("Book author is : " + myBook.author + "<br>");
         document.write("Book price is : " + myBook.price + "<br>");
      </script>

   </body>
</html>
```

## Output

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```

# The 'with' Keyword

The ‘with’ keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to with becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

## Syntax

The syntax for with object is as follows –

```
with (object){
   properties used without the object name and dot
}
```

## Example

Try the following example.

```
<html>
```

```
   <head>
   <title>User-defined objects</title>

      <script type="text/javascript">
         // Define a function which will work as a method
         function addPrice(amount){
            with(this){
               price = amount;
            }
         }

         function book(title, author){
            this.title = title;
            this.author  = author;
            this.price = 0;
            this.addPrice = addPrice; // Assign that method as property.
         }
      </script>

   </head>
   <body>

      <script type="text/javascript">
         var myBook = new book("Perl", "Mohtashim");
         myBook.addPrice(100);

         document.write("Book title is : " + myBook.title + "<br>");
         document.write("Book author is : " + myBook.author + "<br>");
         document.write("Book price is : " + myBook.price + "<br>");
      </script>

   </body>
</html>
```

Output

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```

# JavaScript Native Objects

## The Number Object

The Number object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about Number objects because the browser automatically converts number literals to instances of the number class.

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns NaN (Not-a-Number).

# Number Properties

**MAX_VALUE**

```
function showValue()
            {
               var val = Number.MAX_VALUE;

               document.write ("Value of Number.MAX_VALUE : " + val );
            }

<form>
         <input type="button" value="Click Me" onclick="showValue();" />
      </form>
```

MIN_VALUE

```
var val = Number.MIN_VALUE;

<script type="text/javascript">
         <!--
            function showValue()
            {
               var val = Number.MIN_VALUE;
               alert("Value of Number.MIN_VALUE : " + val );
            }
         //-->
      </script>

<form>
         <input type="button" value="Click Me" onclick="showValue();" />
      </form>
```

NaN

```
var val = Number.NaN;

<script type="text/javascript">
         <!--
            function showValue()
            {
```

```
               var dayOfMonth = 50;

               if (dayOfMonth < 1 || dayOfMonth > 31)
               {
                  dayOfMonth = Number.NaN
                  alert("Day of Month must be between 1 and 31.")
               }
               Document.write("Value of dayOfMonth : " + dayOfMonth );
            }
         //-->
      </script>

<form>
         <input type="button" value="Click Me" onclick="showValue();" />
      </form>
```

# Cookies

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.

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.

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-value pairs

Cookies were originally designed for CGI programming. The data contained in a cookie is automatically transmitted between the web browser and the web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the cookie property of the Document object. JavaScript can read, create, modify, and delete the cookies that apply to the current web page.

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

document.cookie = "key1=value1;key2=value2;expires=date";
Here the expires attribute is optional. If you provide this attribute with a valid date or time, then the cookie will expire on a given date or time and thereafter, the cookies' value will not be accessible.

Note – Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript escape() function to encode the value before storing it in the cookie. If you do this, you will also have to use the corresponding unescape() function when you read the cookie value.

Example
Try the following. It sets a customer name in an input cookie.

```
<html>
  <head>

    <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;
          document.write ("Setting Cookies : " + "name=" + cookievalue );
        }
      //-->
    </script>

  </head>

  <body>

    <form name="myform" action="">
      Enter name: <input type="text" name="customer"/>
      <input type="button" value="Set Cookie" onclick="WriteCookie();"/>
    </form>

  </body>
</html>
```

# Reading Cookies

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function ReadCookie()
            {
               var allcookies = document.cookie;
               document.write ("All Cookies : " + allcookies );

               // Get all the cookies pairs in an array
               cookiearray = allcookies.split(';');

               // Now take key value pair out of this array
               for(var i=0; i<cookiearray.length; i++){
                  name = cookiearray[i].split('=')[0];
                  value = cookiearray[i].split('=')[1];
                  document.write ("Key is : " + name + " and Value is : " + value);
               }
            }
         //-->
      </script>

   </head>
   <body>

      <form name="myform" action="">
         <p> click the following button and see the result:</p>
         <input type="button" value="Get Cookie" onclick="ReadCookie()"/>
      </form>

   </body>
</html>
```

# Setting Cookies Expiry Date

```
 <head>

      <script type="text/javascript">
         <!--
            function WriteCookie()
            {
               var now = new Date();
               now.setMonth( now.getMonth() + 1 );
               cookievalue = escape(document.myform.customer.value) + ";"
```

```
                document.cookie="name=" + cookievalue;
                document.cookie = "expires=" + now.toUTCString() + ";"
                document.write ("Setting Cookies : " + "name=" + cookievalue );
            }
         //-->
      </script>

   </head>
   <body>

      <form name="myform" action="">
         Enter name: <input type="text" name="customer"/>
         <input type="button" value="Set Cookie" onclick="WriteCookie()"/>
      </form>

   </body>
</html>
```

# Deleting a Cookie

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function WriteCookie()
            {
               var now = new Date();
               now.setMonth( now.getMonth() - 1 );
               cookievalue = escape(document.myform.customer.value) + ";"

               document.cookie="name=" + cookievalue;
               document.cookie = "expires=" + now.toUTCString() + ";"
               document.write("Setting Cookies : " + "name=" + cookievalue );
            }
          //-->
      </script>

   </head>
   <body>

      <form name="myform" action="">
         Enter name: <input type="text" name="customer"/>
         <input type="button" value="Set Cookie" onclick="WriteCookie()"/>
      </form>

   </body>
```

```
</html>
```

Event

# onclick Event Type

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function sayHello() {
               alert("Hello World")
            }
         //-->
      </script>

   </head>

   <body>
      <p>Click the following button and see result</p>

      <form>
         <input type="button" onclick="sayHello()" value="Say Hello" />
      </form>

   </body>
</html>
```

# onsubmit Event type

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function validation() {
               all validation goes here
               .........
               return either true or false
            }
         //-->
      </script>
```

```
   </head>
   <body>

      <form method="POST" action="t.cgi" onsubmit="return validate()">
         .......
         <input type="submit" value="Submit" />
      </form>

   </body>
</html>
```

# onmouseover and onmouseout

```
<html>
   <head>

      <script type="text/javascript">
         <!--
            function over() {
               document.write ("Mouse Over");
            }

            function out() {
               document.write ("Mouse Out");
            }

         //-->
      </script>

   </head>
   <body>
      <p>Bring your mouse inside the division to see the result:</p>

      <div onmouseover="over()" onmouseout="out()">
         <h2> This is inside the division </h2>
      </div>

   </body>
</html>
```

**JavaScript Form Validation Example**

In this example, we are going to validate the name and password. The name can't be empty and password can't be less than 6 characters long.

Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

```
<script>
function validateform(){
var name=document.myform.name.value;
var password=document.myform.password.value;

if (name==null || name==""){
  alert("Name can't be blank");
  return false;
}else if(password.length<6){
  alert("Password must be at least 6 characters long.");
  return false;
  }
}
</script>
<body>
<form name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/>
<input type="submit" value="register">
</form>
```

**JavaScript Retype Password Validation**

```
<script type="text/javascript">
function matchpass(){
var firstpassword=document.f1.password.value;
var secondpassword=document.f1.password2.value;

if(firstpassword==secondpassword){
return true;
}
else{
alert("password must be same!");
return false;
}
}
</script>

<form name="f1" action="register.jsp" onsubmit="return matchpass()">
Password:<input type="password" name="password" /><br/>
```

```
Re-enter Password:<input type="password" name="password2"/><br/>
<input type="submit"
```

**JavaScript Number Validation**

Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

```
<script>
function validate(){
var num=document.myform.num.value;
if (isNaN(num)){
  document.getElementById("numloc").innerHTML="Enter Numeric value only";
  return false;
}else{
  return true;
  }
}
</script>
<form name="myform" onsubmit="return validate()" >
Number: <input type="text" name="num"><span id="numloc"></span><br/>
<input type="submit" value="submit">
</form>
```

**JavaScript validation with image**

Let’s see an interactive JavaScript form validation example that displays correct and incorrect image if input is correct or incorrect.

```
<script>
function validate(){
var name=document.f1.name.value;
var password=document.f1.password.value;
var status=false;

if(name.length<1){
document.getElementById("nameloc").innerHTML=
" <img src='unchecked.gif'/> Please enter your name";
status=false;
}else{
document.getElementById("nameloc").innerHTML=" <img src='checked.gif'/>";
status=true;
}
if(password.length<6){
document.getElementById("passwordloc").innerHTML=
" <img src='unchecked.gif'/> Password must be at least 6 char long";
```

```
status=false;
}else{
document.getElementById("passwordloc").innerHTML=" <img src='checked.gif'/>";
}
return status;
}
</script>

<form name="f1" action="#" onsubmit="return validate()">
<table>
<tr><td>Enter Name:</td><td><input type="text" name="name"/>
<span id="nameloc"></span></td></tr>
<tr><td>Enter Password:</td><td><input type="password" name="password"/>
<span id="passwordloc"></span></td></tr>
<tr><td colspan="2"><input type="submit" value="register"/></td></tr>
</table>
</form>
```

**JavaScript email validation**

```
<script>
function validateemail()
{
var x=document.myform.email.value;
var atposition=x.indexOf("@");
var dotposition=x.lastIndexOf(".");
if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
  alert("Please enter a valid e-mail address \n atpostion:"+atposition+"\n
dotposition:"+dotposition);
  return false;
  }
}
</script>
<body>
<form name="myform"  method="post" action="#" onsubmit="return validateemail();">
Email: <input type="text" name="email"><br/>

<input type="submit" value="register">
</form>
```

# METHOD 1) SESSION STORAGE

```
<script>
function store () {
  // (A) VARIABLES TO PASS
  var first = "Foo Bar",
      second = ["Hello", "World"];

  // (B) SAVE TO SESSION STORAGE
  // (B1) FLAT STRING OR NUMBER
  // SESSIONSTORAGE.SETITEM("KEY", "VALUE");
  sessionStorage.setItem("first", first);

  // (B2) ARRAY OR OBJECT
  // SESSION STORAGE CANNOT STORE ARRAY AND OBJECTS
  // JSON ENCODE BEFORE STORING, CONVERT TO STRING
  sessionStorage.setItem("second", JSON.stringify(second));

  // (C) REDIRECT
  location.href = "1b-session.html";
  // window.open("1b-session.html");
}
</script>

<input type="button" value="Store To Session Storage" onclick="store()"/>
```

1b-session.html

```
<script>
function get () {
  // (A) GET FROM SESSION
  var first = sessionStorage.getItem("first"),
      second = JSON.parse(sessionStorage.getItem("second"));

  // (B) IT WORKS!
  // MANUALLY OPENING 1B-SESSION.HTML WILL NOT WORK THOUGH
  // SESSION DATA WILL PERISH ONCE TAB/WINDOW IS CLOSED
  console.log(first);  // Foo Bar
  console.log(second); // ["Hello", "World"]

  // (EXTRA) CLEAR SESSION STORAGE
```

```
  // sessionStorage.removeItem("KEY");
  // sessionStorage.clear();
}
</script>

<input type="button" value="Get From Session Storage" onclick="get()"/>
```

# COOKIE STORAGE

## 4a-cookie.html

```
<script>
function store () {
  // (A) VARIABLES TO PASS
  var first = "Foo Bar",
      second = ["Hello", "World"];

  // (B) URL PARAMETERS + STORE INTO COOKIE
  // * WORKS, BUT NOT RECOMMENDED
  var params = new URLSearchParams();
  params.append("first", first);
  params.append("second", JSON.stringify(second));
  document.cookie = "pass=" + params.toString();
```

```
  // (C) REDIRECT
  location.href = "4b-cookie.html";
  // window.open("4b-cookie.html");
}
</script>

<input type="button" value="Go"
onclick="store()"/>
```

4b-cookie.html

```
<script>
function get () {
  // (A) GET BACK PASS VARS
  var passvars = document.cookie
    .split("; ")
    .find(row => row.startsWith("pass"))
    .substring(5);
```

```
  // (B) PARSE BACK
  var params = new
URLSearchParams(passvars),
      first = params.get("first"),
      second =
JSON.parse(params.get("second"));

  // (C) IT WORKS!
  console.log(first);  // Foo Bar
  console.log(second); // ["Hello",
"World"]
}
</script>

<input type="button" value="Get"
onclick="get()"/>
```

Login

Html

```
<!DOCTYPE
html>
                <html lang="en">


                <head>
                 <meta charset="UTF-8">
                 <meta name="viewport" content="width=device-width, initial-scale=1.0">
                 <title>Login</title>
                 <link rel="stylesheet" href="login-page.css">
                 <script defer src="login-page.js"></script>
                </head>


                <body>
                 <main id="main-holder">
                   <h1 id="login-header">Login</h1>


                   <div id="login-error-msg-holder">
                     <p id="login-error-msg">Invalid username <span
                id="error-msg-second-line">and/or password</span></p>
                   </div>


                   <form id="login-form">
                     <input type="text" name="username" id="username-field"
                class="login-form-field" placeholder="Username">
```

```
                    <input type="password" name="password" id="password-field"
                class="login-form-field" placeholder="Password">
                    <input type="submit" value="Login" id="login-form-submit">
                  </form>


                 </main>
                </body>



                </html>
```

Css

```
html {
  height: 100%;
}

body {
  height: 100%;
  margin: 0;
  font-family: Arial, Helvetica, sans-serif;
  display: grid;
  justify-items: center;
  align-items: center;
  background-color: #3a3a3a;
}

#main-holder {
  width: 50%;
  height: 70%;
  display: grid;
  justify-items: center;
  align-items: center;
  background-color: white;
  border-radius: 7px;
  box-shadow: 0px 0px 5px 2px black;
}

#login-error-msg-holder {
  width: 100%;
```

```
  height: 100%;
  display: grid;
  justify-items: center;
  align-items: center;
}

#login-error-msg {
  width: 23%;
  text-align: center;
  margin: 0;
  padding: 5px;
  font-size: 12px;
  font-weight: bold;
  color: #8a0000;
  border: 1px solid #8a0000;
  background-color: #e58f8f;
  opacity: 0;
}

#error-msg-second-line {
  display: block;
}

#login-form {
  align-self: flex-start;
  display: grid;
  justify-items: center;
  align-items: center;
}

.login-form-field::placeholder {
  color: #3a3a3a;
}

.login-form-field {
  border: none;
  border-bottom: 1px solid #3a3a3a;
  margin-bottom: 10px;
  border-radius: 3px;
  outline: none;
  padding: 0px 0px 5px 5px;
}
```

```
#login-form-submit {
  width: 100%;
  padding: 7px;
  border: none;
  border-radius: 5px;
  color: white;
  font-weight: bold;
  background-color: #3a3a3a;
  cursor: pointer;
  outline: none;
}
```

Java script

```
const loginForm =
document.getElementById("login-form");
```

```
const loginButton =
document.getElementById("login-form-submit");
const loginErrorMsg =
document.getElementById("login-error-msg");


loginButton.addEventListener("click", (e) => {
   e.preventDefault();
   const username = loginForm.username.value;
   const password = loginForm.password.value;


   if (username === "user" && password ===
"web_dev") {
       alert("You have successfully logged
in.");
       location.reload();
```

```
    } else {
        loginErrorMsg.style.opacity = 1;
    }
})
```