

JavaScript - Events

What is an Event ?

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.

Please go through this small tutorial for a better understanding HTML Event Reference. Here we will see a few examples to understand a relation between Event and JavaScript

Example

```
<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

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a validate() function before submitting a form data to the webserver. If validate() function returns true, the form will be submitted, otherwise it will not submit the data.

```
<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

These two event types will help you create nice effects with images or even with text as well. The onmouseover event triggers when you bring your mouse over any element and the onmouseout triggers when you move your mouse out from that element. Try the following example.

```
<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

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

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
onbeforeunload	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

HTML 5 Standard Events

Attribute	Value	Description
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

HTML 5 Standard Events

Attribute	Value	Description
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 reached the end

HTML 5 Standard Events

Attribute	Value	Description
onerror	script	Triggers when an error occurs
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

HTML 5 Standard Events

Attribute	Value	Description
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

HTML 5 Standard Events

Attribute	Value	Description
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
ononline	script	Triggers when the document comes online

HTML 5 Standard Events

Attribute	Value	Description
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

HTML 5 Standard Events

Attribute	Value	Description
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

HTML 5 Standard Events

Attribute	Value	Description
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 and Cookies

What are Cookies ?

Web Browsers and Servers 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 users' 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.

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.

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.

Example

```
<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>
```

Output

Now your machine has a cookie called name. You can set multiple cookies using multiple key = value pairs separated by comma.

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.

Reading Cookies

Reading a cookie is just as simple as writing one, because the value of the document.cookie object is the cookie. So you can use this string whenever you want to access the cookie. The document.cookie string will keep a list of name=value pairs separated by semicolons, where name is the name of a cookie and value is its string value.

You can use strings' split() function to break a string into key and values as follows

Example

```
<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>
```

Note – Here length is a method of Array class which returns the length of an array. We will discuss Arrays in a separate chapter. By that time, please try to digest it.

Note – There may be some other cookies already set on your machine. The above code will display all the cookies set on your machine.

Setting Cookies Expiry Date

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiry date within the cookie. This can be done by setting the 'expires' attribute to a date and time.

Example

```
<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>
```

Deleting a Cookie

Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiry date to a time in the past.

Example

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

</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>
```

JavaScript - Page Redirection

What is Page Redirection ?

You might have encountered a situation where you clicked a URL to reach a page X but internally you were directed to another page Y. It happens due to page redirection. This concept is different from JavaScript Page Refresh.

There could be various reasons why you would like to redirect a user from the original page. We are listing down a few of the reasons

- You did not like the name of your domain and you are moving to a new one. In such a scenario, you may want to direct all your visitors to the new site. Here you can maintain your old domain but put a single page with a page redirection such that all your old domain visitors can come to your new domain.
- You have built-up various pages based on browser versions or their names or may be based on different countries, then instead of using your server-side page redirection, you can use client-side page redirection to land your users on the appropriate page.
- The Search Engines may have already indexed your pages. But while moving to another domain, you would not like to lose your visitors coming through search engines. So you can use client-side page redirection. But keep in mind this should not be done to fool the search engine, it could lead your site to get banned.

How Page Re-direction Works ?

The implementations of Page-Redirection are as follows

Example

It is quite simple to do a page redirect using JavaScript at client side. To redirect your site visitors to a new page, you just need to add a line in your head section as follows.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Redirect() {
          window.location = "https://www.ucsc.lk";
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button, you will be redirected to home page.</p>

    <form>
      <input type = "button" value = "Redirect Me" onclick = "Redirect();"/>
    </form>

  </body>
</html>
```

How Page Re-direction Works ?

Example 2

You can show an appropriate message to your site visitors before redirecting them to a new page. This would need a bit time delay to load a new page. The following example shows how to implement the same. Here setTimeout() is a built-in JavaScript function which can be used to execute another function after a given time interval.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Redirect() {
          window.location = "https://www.ucsc.lk";
        }
        document.write("You will be redirected to main page in 10 sec.");
        setTimeout('Redirect()', 10000);
      //-->
    </script>
  </head>

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

Output

You will be redirected to tutorialspoint.com main page in 10 seconds!

How Page Re-direction Works ?

Example 3

The following example shows how to redirect your site visitors onto a different page based on their browsers.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Redirect() {
          window.location = "https://www.ucsc.lk";
        }
        document.write("You will be redirected to main page in 10 sec.");
        setTimeout('Redirect()', 10000);
      //-->
    </script>
  </head>

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

JavaScript - Dialog Boxes

Alert Dialog Box

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

Example

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function Warn() {
          alert ("This is a warning message!");
          document.write ("This is a warning message!");
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick = "Warn();"/>
    </form>
  </body>
</html>
```

Confirmation Dialog Box

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false. You can use a confirmation dialog box as follows.

Example

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
        function getConfirmation() {
          var retVal = confirm("Do you want to continue ?");
          if( retVal == true ) {
            document.write ("User wants to continue!");
            return true;
          } else {
            document.write ("User does not want to continue!");
            return false;
          }
        }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type = "button" value = "Click Me" onclick = "getConfirmation();"/>
    </form>
  </body>
</html>
```

Prompt Dialog Box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called `prompt()` which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method `prompt()` will return the entered value from the text box. If the user clicks the Cancel button, the window method `prompt()` returns null.

Example

```
<html>
<head>
<script type = "text/javascript">
<!--
function getValue() {
    var retVal = prompt("Enter your name : ", "your name here");
    document.write("You have entered : " + retVal);
}
//-->
</script>
</head>

<body>
<p>Click the following button to see the result: </p>
<form>
<input type = "button" value = "Click Me" onclick = "getValue();"/>
</form>
</body>
</html>
```

JavaScript - Void Keyword

Alert Dialog Box

void is an important keyword in JavaScript which can be used as a unary operator that appears before its single operand, which may be of any type. This operator specifies an expression to be evaluated without returning a value.

Syntax

The syntax of void can be either of the following two

```
<head>
<script type = "text/javascript">
<!--
void func()
javascript:void func()
or:
void(func())
javascript:void(func())
//-->
</script>
</head>
```

Confirmation Dialog Box

Example 1

The most common use of this operator is in a client-side javascript: URL, where it allows you to evaluate an expression for its side-effects without the browser displaying the value of the evaluated expression.

Here the expression **alert ('Warning!!!')** is evaluated but it is not loaded back into the current document

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

  <body>
    <p>Click the following, This won't react at all...</p>
    <a href = "javascript:void(alert('Warning!!!'))">Click me!</a>
  </body>
</html>
```

Confirmation Dialog Box

Example 2

Take a look at the following example. The following link does nothing because the expression "0" has no effect in JavaScript. Here the expression "0" is evaluated, but it is not loaded back into the current document.

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

  <body>
    <p>Click the following, This won't react at all...</p>
    <a href = "javascript:void(0)">Click me!</a>
  </body>
</html>
```

Confirmation Dialog Box

Example 3

Another use of void is to purposely generate the undefined value as follows.

```
<html>
  <head>
    <script type = "text/javascript">
      <!--
      function getValue() {
        var a,b,c;

        a = void ( b = 5, c = 7 );
        document.write('a = ' + a + ' b = ' + b + ' c = ' + c );
      }
      //-->
    </script>
  </head>

  <body>
    <p>Click the following to see the result:</p>
    <form>
      <input type = "button" value = "Click Me" onclick = "getValue();" />
    </form>
  </body>
</html>
```

JavaScript - Objects Overview

JavaScript - Objects Overview

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

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

```
<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.

```
<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  
}
```

The 'with' Keyword

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

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

JavaScript - Form Validation

JavaScript - Form Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- Basic Validation – First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- Data Format Validation – Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Example

```
<html>
  <head>
    <title>Form Validation</title>
    <script type = "text/javascript">
      <!--
        // Form validation code will come here.
      //-->
    </script>
  </head>

  <body>
    <form action = "/cgi-bin/test.cgi" name = "myForm" onsubmit = "return(validate());">
      <table cellspacing = "2" cellpadding = "2" border = "1">

        <tr>
          <td align = "right">Name</td>
          <td><input type = "text" name = "Name" /></td>
        </tr>

        <tr>
          <td align = "right">EMail</td>
          <td><input type = "text" name = "EMail" /></td>
        </tr>

        <tr>
          <td align = "right">Zip Code</td>
          <td><input type = "text" name = "Zip" /></td>
        </tr>

        <tr>
          <td align = "right">Country</td>
          <td>
            <select name = "Country">
              <option value = "-1" selected>[choose yours]</option>
              <option value = "1">USA</option>
              <option value = "2">UK</option>
              <option value = "3">INDIA</option>
            </select>
          </td>
        </tr>

        <tr>
          <td align = "right"></td>
          <td><input type = "submit" value = "Submit" /></td>
        </tr>

      </table>
    </form>
  </body>
</html>
```

Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling validate() to validate data when onsubmit event is occurring. The following code shows the implementation of this validate() function.

```
<script type = "text/javascript">
<!--
// Form validation code will come here.
function validate() {

if( document.myForm.Name.value == "" ) {
    alert( "Please provide your name!" );
    document.myForm.Name.focus();
    return false;
}
if( document.myForm.EMail.value == "" ) {
    alert( "Please provide your Email!" );
    document.myForm.EMail.focus();
    return false;
}
if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value ) ||
document.myForm.Zip.value.length != 5 ) {

    alert( "Please provide a zip in the format #####." );
    document.myForm.Zip.focus();
    return false;
}
if( document.myForm.Country.value == "-1" ) {
    alert( "Please provide your country!" );
    return false;
}
return( true );
}
//-->
</script>
```

Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Example

Try the following code for email validation.

```
<script type = "text/javascript">
<!--
function validateEmail() {
    var emaiIID = document.myForm.EMail.value;
    atpos = emaiIID.indexOf("@");
    dotpos = emaiIID.lastIndexOf(".");
    if (atpos < 1 || ( dotpos - atpos < 2 )) {
        alert("Please enter correct email ID")
        document.myForm.EMail.focus();
        return false;
    }
    return( true );
}
//-->
</script>
```

Thank you
