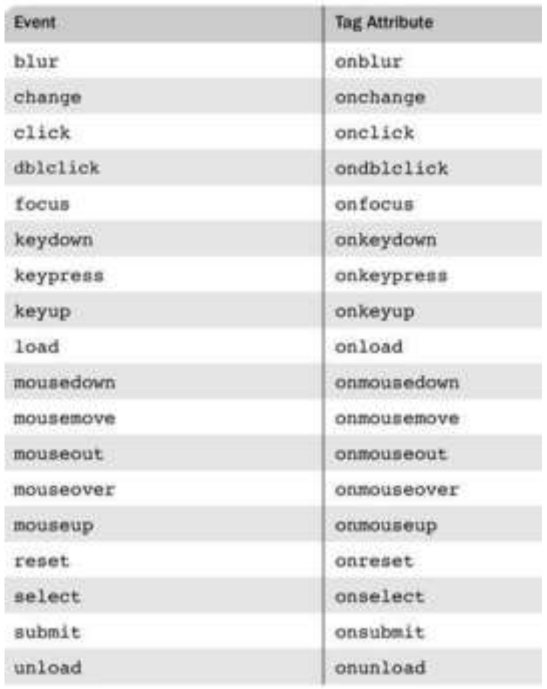
JS EVENTS & EVENT HANDLING

- In event-driven programming, parts of the program are executed at completely unpredictable times, often triggered by user interactions with the program that is executing.
- an **event** is an object that is **implicitly created by the browser and the JavaScript system** in response to something happening.
- An **event handler** is **a script that is implicitly executed** in response to the appearance of an event.
- Event handlers enable a Web document to be responsive to browser and user activities.
- One of the most common uses of event handlers is to check for simple errors and omissions in user input to the elements of a form, either when they are changed or when the form is submitted.
- Events are created by activities associated with specific XHTML elements. Eg: mouse click, form element clicks etc.
- The process of connecting an event handler to an event is called **registration**
- An event handler can be assigned in the following ways:
  - Via an element attribute directly in HTML:
    `<body onload="alert('Hello World!');">`
  - Via JavaScript assigning to element attribute:
    `document.onclick = clickHandler;`
  - Via JavaScript by calling addEventListener() method of an element.
    `element.addEventListener(event, function, useCapture);`
- *The write method of document should never be used in an event handler*

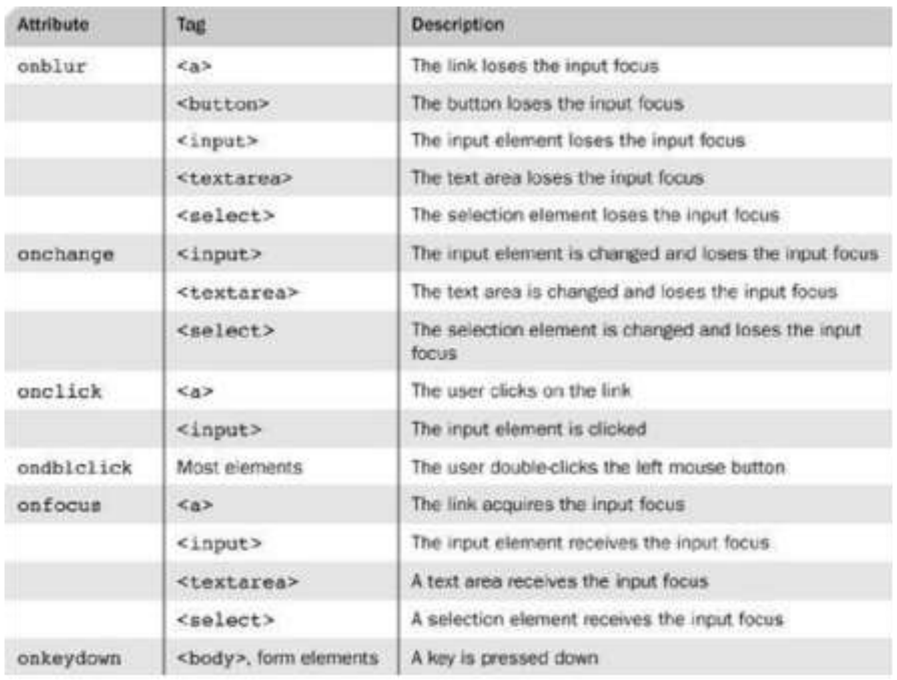
Events, tags and attributes

Common Events and their tag attributes

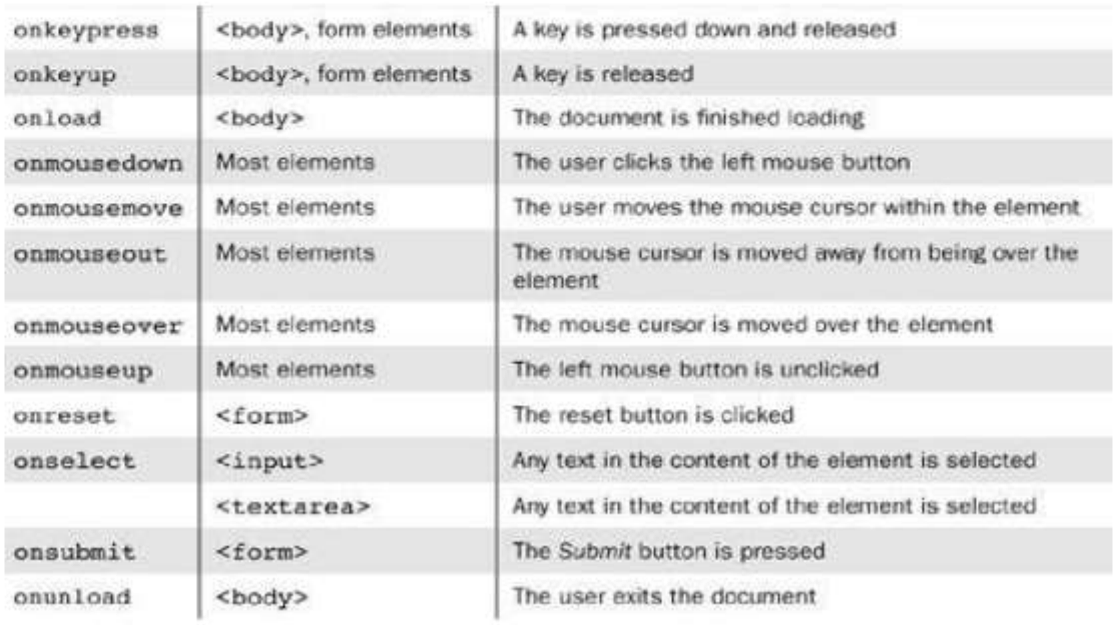
Event attributes and their tags



Event	Tag Attribute
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload



Attribute	Tag	Description
onblur	`<a>`	The link loses the input focus
	`<button>`	The button loses the input focus
	`<input>`	The input element loses the input focus
	`<textarea>`	The text area loses the input focus
	`<select>`	The selection element loses the input focus
onchange	`<input>`	The input element is changed and loses the input focus
	`<textarea>`	The text area is changed and loses the input focus
	`<select>`	The selection element is changed and loses the input focus
onclick	`<a>`	The user clicks on the link
	`<input>`	The input element is clicked
ondblclick	Most elements	The user double-clicks the left mouse button
onfocus	`<a>`	The link acquires the input focus
	`<input>`	The input element receives the input focus
	`<textarea>`	A text area receives the input focus
	`<select>`	A selection element receives the input focus
onkeydown	`<body>`, form elements	A key is pressed down

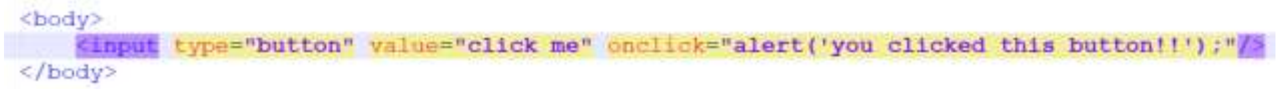


onkeypress	<body>, form elements	A key is pressed down and released
onkeyup	<body>, form elements	A key is released
onload	<body>	The document is finished loading
onmousedown	Most elements	The user clicks the left mouse button
onmousemove	Most elements	The user moves the mouse cursor within the element
onmouseout	Most elements	The mouse cursor is moved away from being over the element
onmouseover	Most elements	The mouse cursor is moved over the element
onmouseup	Most elements	The left mouse button is unclicked
onreset	<form>	The reset button is clicked
onselect	<input>	Any text in the content of the element is selected
	<textarea>	Any text in the content of the element is selected
onsubmit	<form>	The *Submit* button is pressed
onunload	<body>	The user exits the document

Event handling

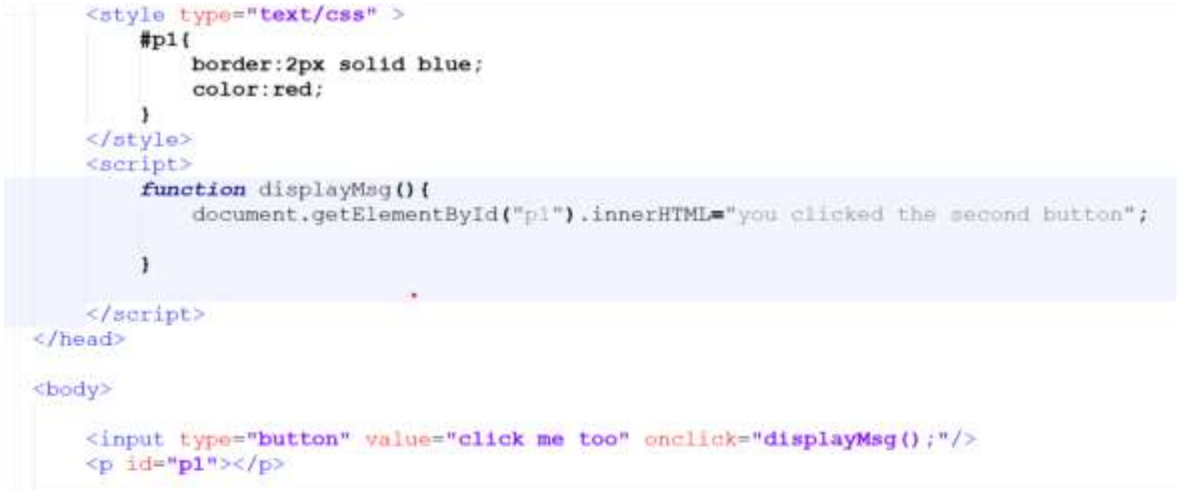
In DOM 0 event model there are two ways to register an event handler.

1. by assigning the event handler script to an event tag attribute.
   Example:



```
<body>
    <input type="button" value="click me" onclick="alert('you clicked this button!!');"/>
</body>
```

   When there is a use of function, the value of event tag attribute is the call to the function.
   Example:



```
        <style type="text/css" >
            #p1{
                border:2px solid blue;
                color:red;
            }
        </style>
        <script>
            function displayMsg(){
                document.getElementById("p1").innerHTML="you clicked the second button";

            }

        </script>
    </head>

    <body>

        <input type="button" value="click me too" onclick="displayMsg();"/>
        <p id="p1"></p>
```

2. An event handler function is registered by assigning its name to the associated event property on the selected object.

   Example:

//HTML file

```
<body>

    <input type="button" value="Click me!" id="B1" /><br />
    <p id="p1"></p>

    <script src="JSeventreg.js" type="text/javascript"></script>
</body>
```

//JS file

```
document.getElementById("B1").onclick=display;

function display(){
    document.getElementById("p1").innerHTML="Great! You did it.";
}
```

//HTML file

```
<form id="form1">

radio 1: <input type="radio" name="radio" id="r1" value="1"/> </br>
radio 2: <input type="radio" name="radio" id="r2" value="2"/></br>
<input type="text" id="text1" onfocus="this.blur();"/>

</form>

<script type="text/javascript" src="radio2.js">
</script>
```

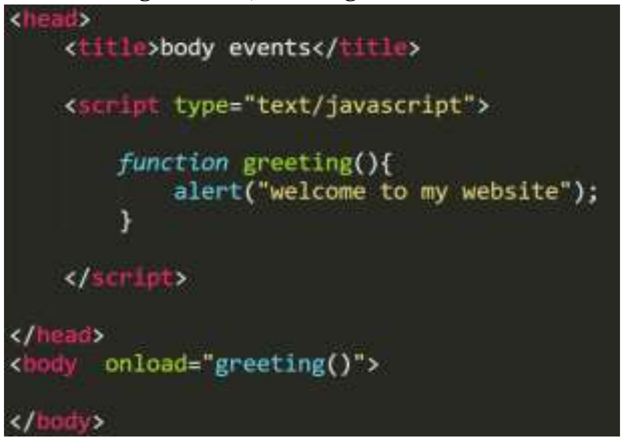
//JS File

```
document.getElementById("r1").onclick=display;  //event registration
document.getElementById("r2").onclick=display;
function display(x){
    var obj=document.getElementById("form1");

    for(var i=0;i<obj.radio.length;i++)
    {
            if(obj.radio[i].checked){
                x=obj.radio[i].value;
                break;
            }

    }

    if(x==1)
        document.getElementById("text1").value="button 1 clicked!";
    else
        document.getElementById("text1").value="button 2 clicked!";

}
```

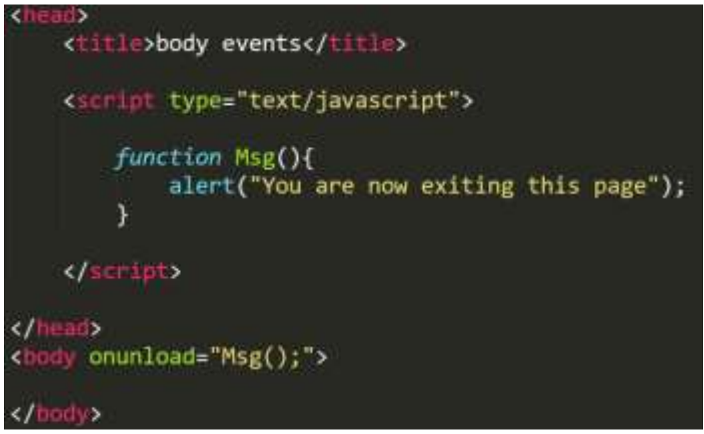
Examples:

❖ Handling Events from Body Elements

- The events most often created by body elements are load and unload.
- The onload and onunload events are triggered when the user enters or leaves the page.
- The onload event can be <u>used to check the visitor's browser type and browser version</u>, and <u>load the proper version of the web page</u> based on the information.
- The onload event can also be used to deal with cookies
- The onunload event occurs once a page has unloaded (or the browser window has been closed).
- onunload occurs when the user navigates away from the page (by clicking on a link, submitting a form, closing the browser window, reloading a page etc.).



```
<head>
    <title>body events</title>

    <script type="text/javascript">

        function greeting(){
            alert("welcome to my website");
        }

    </script>

</head>
<body  onload="greeting()">

</body>
```



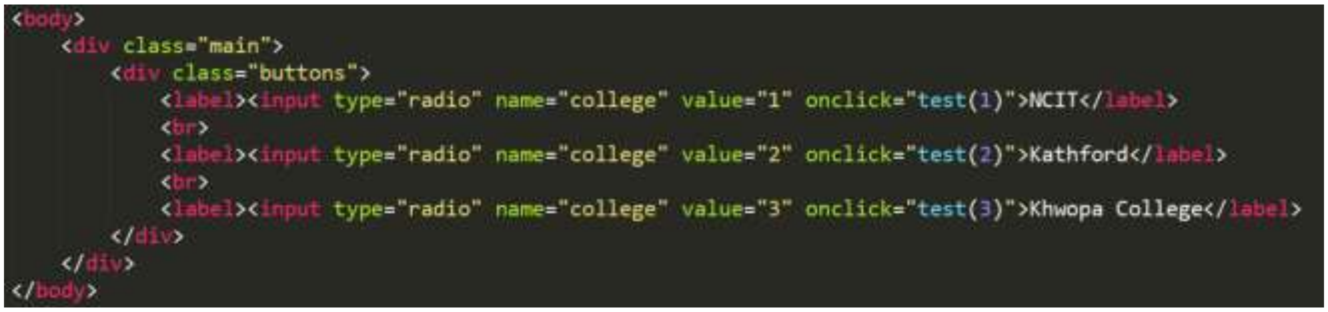
```
<head>
    <title>body events</title>

    <script type="text/javascript">

        function Msg(){
            alert("You are now exiting this page");
        }

    </script>

</head>
<body onunload="Msg();">

</body>
```

❖ Handling events from button elements

- Buttons in a Web document provide an effective way to collect simple input from the browser user.
- The most commonly used event created by button actions is click.

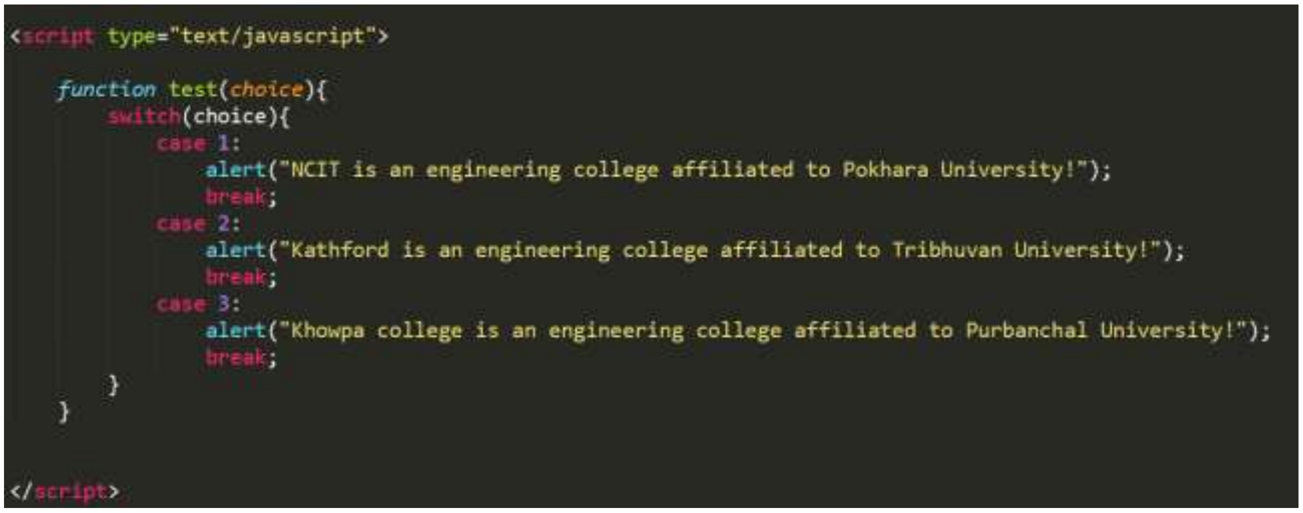
Examples:

HTML code



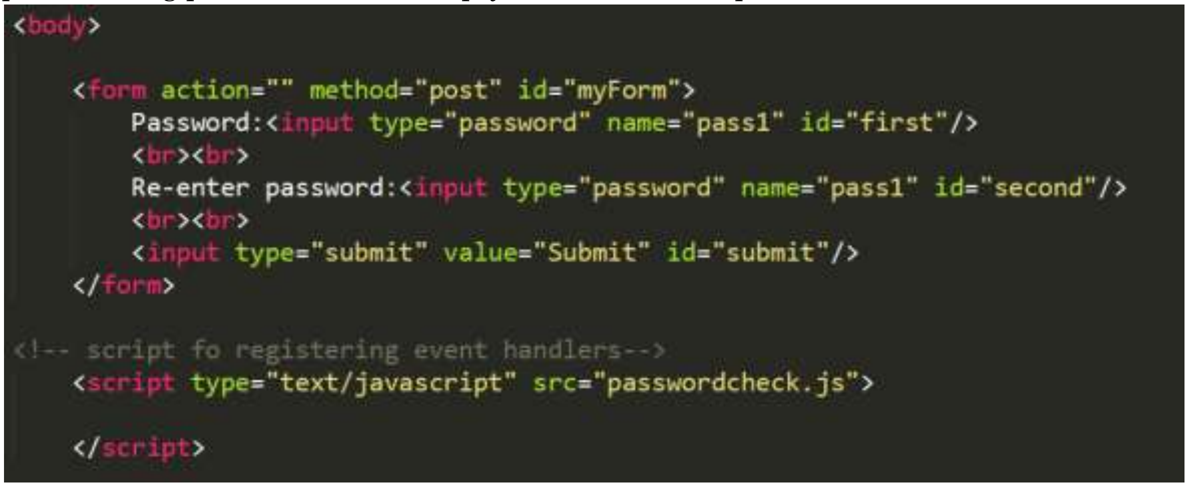
```
<body>
    <div class="main">
        <div class="buttons">
            <label><input type="radio" name="college" value="1" onclick="test(1)">NCIT</label>
            <br>
            <label><input type="radio" name="college" value="2" onclick="test(2)">Kathford</label>
            <br>
            <label><input type="radio" name="college" value="3" onclick="test(3)">Khwopa College</label>
        </div>
    </div>
</body>
```

JavaScript code



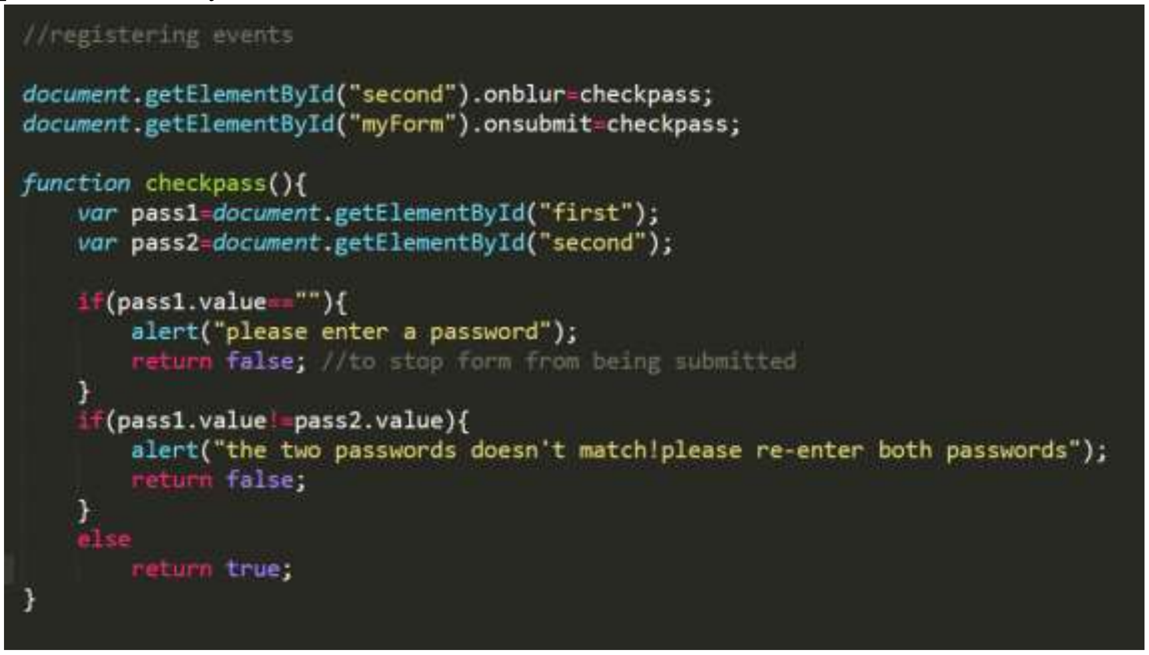
```
<script type="text/javascript">

    function test(choice){
        switch(choice){
            case 1:
                alert("NCIT is an engineering college affiliated to Pokhara University!");
                break;
            case 2:
                alert("Kathford is an engineering college affiliated to Tribhuvan University!");
                break;
            case 3:
                alert("Khowpa college is an engineering college affiliated to Purbanchal University!");
                break;
        }
    }


</script>
```

❖ Handling events from text box and password elements
Example: checking password field is empty or not AND two passwords matches or not



```
<body>

    <form action="" method="post" id="myForm">
        Password:<input type="password" name="pass1" id="first"/>
        <br><br>
        Re-enter password:<input type="password" name="pass1" id="second"/>
        <br><br>
        <input type="submit" value="Submit" id="submit"/>
    </form>

<!-- script fo registering event handlers-->
    <script type="text/javascript" src="passwordcheck.js">

    </script>
```

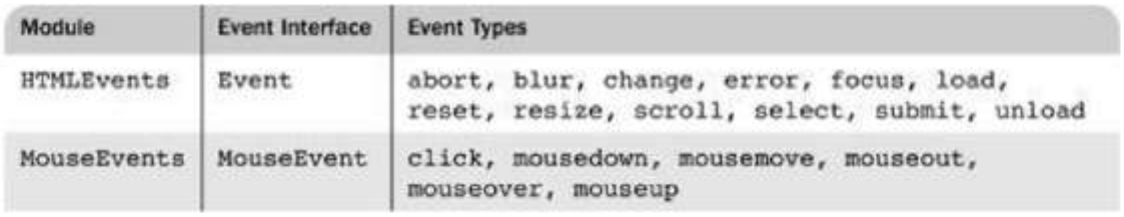
passwordcheck.js



```
//registering events

document.getElementById("second").onblur=checkpass;
document.getElementById("myForm").onsubmit=checkpass;

function checkpass(){
    var pass1=document.getElementById("first");
    var pass2=document.getElementById("second");

    if(pass1.value==""){
        alert("please enter a password");
        return false; //to stop form from being submitted
    }
    if(pass1.value!=pass2.value){
        alert("the two passwords doesn't match!please re-enter both passwords");
        return false;
    }
    else
        return true;
}
```

DOM 2 Event Model

- DOM 2 is modularized - one module is Events, which has two submodules, HTMLEvents and MouseEvents, whose interfaces are Event (blur, change, etc.) and MouseEvent (click, mouseup, etc.)



Module	Event Interface	Event Types
HTMLEvents	Event	abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload
MouseEvents	MouseEvent	click, mousedown, mousemove, mouseout, mouseover, mouseup

- When an event occurs, an event object is created at some node in the document tree. The node is called **target node**.
- There are three phases for handling events:
  1. Capturing phase
  2. Target node phase
  3. Bubbling phase
- Event handler registration is done with the **addEventListener** method
  - It takes three parameters

    Name of the event, as a string literal

    The **handler function**.

    A **Boolean value** that specifies whether the event is enabled during the capturing phase (If the value true is specified, the handler is enabled for the capturing phase. In fact, an enabled handler can be called only during capturing. If the value is false, the handler is not enabled and can be called either at the target node or on any node reached during bubbling.)

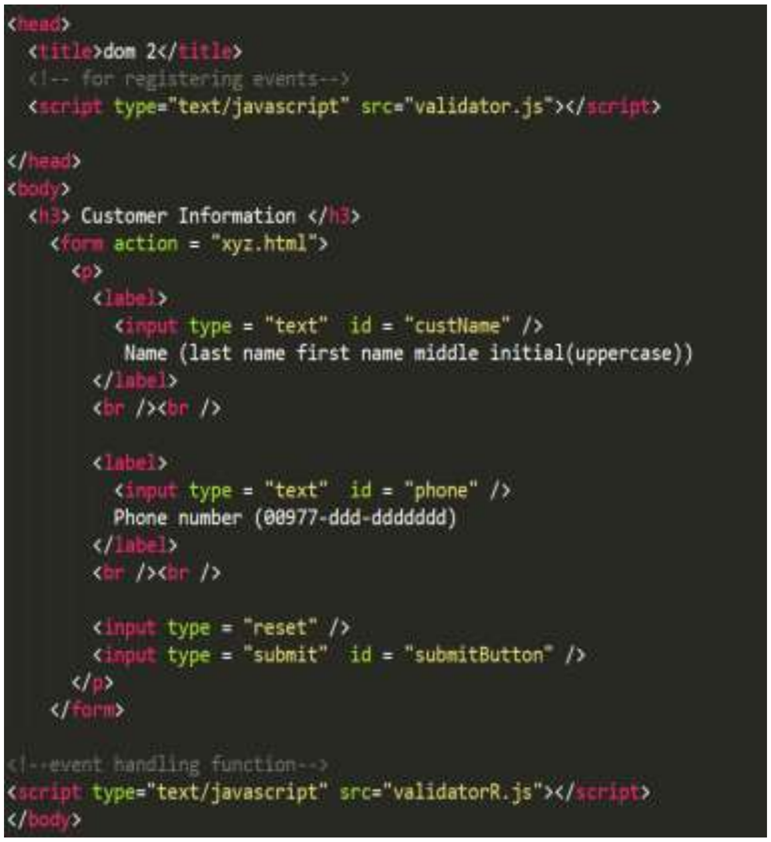
  Example: node.addEventListener("change", chkName, false);

- **Capturing phase:**
  - The event created at the target node starts at the document root node and propagates down the tree to the target node.
  - If there are any handlers for the event that are registered on any node encountered in this propagation, including the document node but not the target node, these handlers are checked to determine whether they are enabled.
  - Any enabled handler for the event that is found during capturing is executed.
- **Target node phase:**
  - When the event reaches the target node, the second phase, called the target node phase, takes place.
  - In this phase, the handlers registered for the event at the target node are executed.
  - The process is similar to what happens with the DOM 0 event model.
- **Bubbling phase:**
  - After execution of any appropriate handlers at the target node, the third phase begins.
  - This is the bubbling phase, in which the event bubbles back up the document tree to the document node(Event goes back to the root).
  - On this trip back up the tree, any handler registered for the event at any node on the way is executed (whether it is enabled or not).
- Any handler can stop further event propagation by calling the **stopPropagation** method of the Event object
- DOM 2 model uses the Event object method, **preventDefault,** to stop default operations, such as submission of a form, if an error has been detected.

- A temporary handler can be created by registering it and then unregistering it with **removeEventListener.**
  - The first parameter accepts an event type.
  - the second one accepts the event handler.
  - The third one point out the handler is for the capturing phase (true) or bubbling phase (false).
- The **currentTarget** property of Event always references the object on which the handler is being executed.

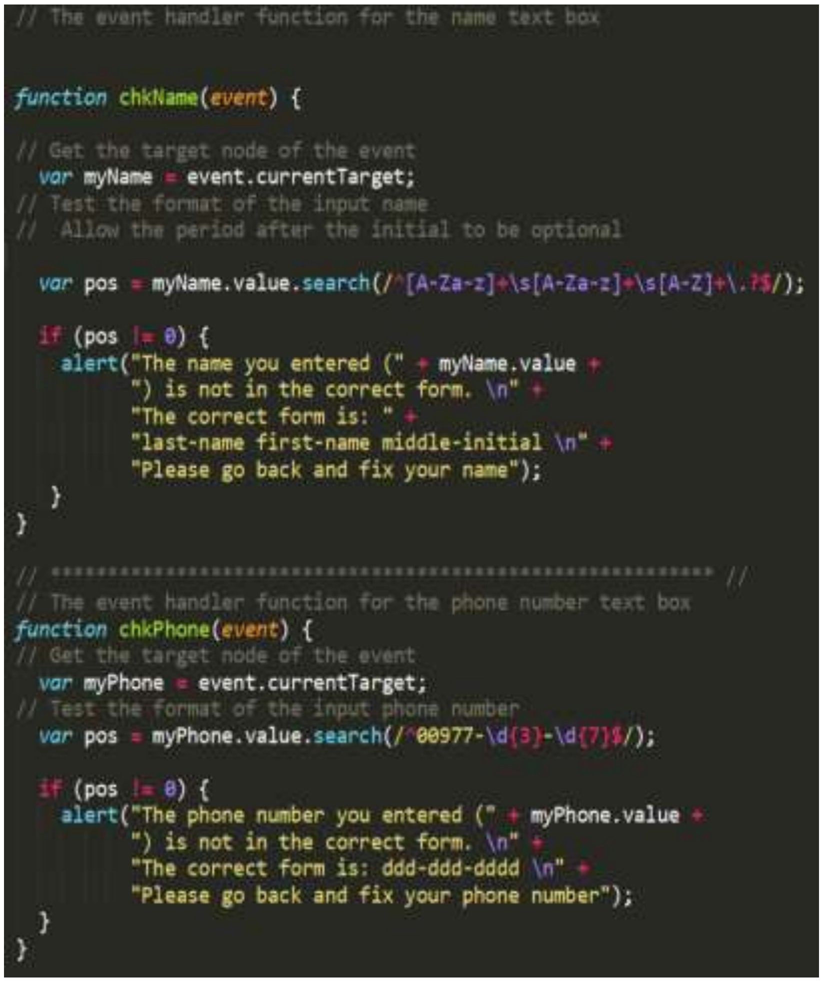
Example:

HTML file



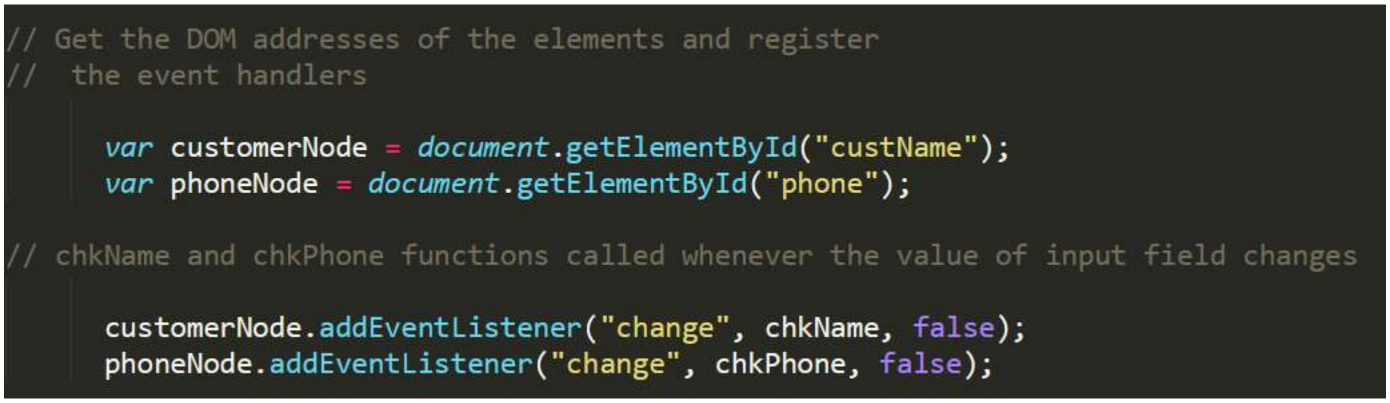
```html
<head>
  <title>dom 2</title>
  <!-- for registering events-->
  <script type="text/javascript" src="validator.js"></script>

</head>
<body>
  <h3> Customer Information </h3>
    <form action = "xyz.html">
      <p>
        <label>
          <input type = "text"  id = "custName" />
           Name (last name first name middle initial(uppercase))
        </label>
        <br /><br />

        <label>
          <input type = "text"  id = "phone" />
          Phone number (00977-ddd-ddddddd)
        </label>
        <br /><br />

        <input type = "reset" />
        <input type = "submit"  id = "submitButton" />
      </p>
    </form>

<!--event handling function-->
<script type="text/javascript" src="validatorR.js"></script>
</body>
```

validator.js



```javascript
// The event handler function for the name text box


function chkName(event) {

// Get the target node of the event
  var myName = event.currentTarget;
// Test the format of the input name
//  Allow the period after the initial to be optional

  var pos = myName.value.search(/^[A-Za-z]+\s[A-Za-z]+\s[A-Z]+\.?$/);

  if (pos != 0) {
    alert("The name you entered (" + myName.value +
          ") is not in the correct form. \n" +
          "The correct form is: " +
          "last-name first-name middle-initial \n" +
          "Please go back and fix your name");
  }
}

// ********************************************************* //
// The event handler function for the phone number text box
function chkPhone(event) {
// Get the target node of the event
  var myPhone = event.currentTarget;
// Test the format of the input phone number
  var pos = myPhone.value.search(/^00977-\d{3}-\d{7}$/);

  if (pos != 0) {
    alert("The phone number you entered (" + myPhone.value +
          ") is not in the correct form. \n" +
          "The correct form is: ddd-ddd-dddd \n" +
          "Please go back and fix your phone number");
  }
}
```

validatorR.js



```javascript
// Get the DOM addresses of the elements and register
//  the event handlers

     var customerNode = document.getElementById("custName");
     var phoneNode = document.getElementById("phone");

// chkName and chkPhone functions called whenever the value of input field changes

     customerNode.addEventListener("change", chkName, false);
     phoneNode.addEventListener("change", chkPhone, false);
```