JavaScript Form Validation

t is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

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 Events

The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.

For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
Click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
Blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
Load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Let's discuss some examples over events and their handlers.

Click Event

```
<html>
<head> Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
   <!--
   function clickevent()
   {
      document.write("This is JavaTpoint");
   }
   //-->
</script>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
</body>
</html>
```

MouseOver Event

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
   <!--
   function mouseoverevent()
   {
      alert("This is JavaTpoint");
   }
   //-->
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>
```

Focus Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>
<!--
   function focusevent()
   {
      document.getElementById("input1").style.background=" aqua";
   }
//-->
</script>
</body>
</html>
```

Keydown Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()"/>
<script>
<!--
   function keydownevent()
   {
      document.getElementById("input1");
      alert("Pressed a key");
   }
//-->
</script>
</body>
</html>
```

Load event

```
<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page successfully loaded');">
<script>
<!--
```

```
7. document.write("The page is loaded successfully");
8. //-->
9. </script>
10. </body>
11. </html>
```

avaScript addEventListener()

The **addEventListener()** method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.

Syntax

```
1. element.addEventListener(event, function, useCapture);
```

Although it has three parameters, the parameters ***event*** and ***function*** are widely used. The third parameter is optional to define. The values of this function are defined as follows.

Parameter Values

event: It is a required parameter. It can be defined as a string that specifies the event's name.

> *Note: Do not use any prefix such as "on" with the parameter value. For example, Use "click" instead of using "onclick".*

function: It is also a required parameter. It is a JavaScript function which responds to the event occur.

useCapture: It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is **false**.

```
1. <!DOCTYPE html>
2. <html>
3. <body>
4. <p> This is an example of adding multiple events to the same element. </p>
5. <p> Click the following button to see the effect. </p>
6. <button id = "btn"> Click me </button>
7. <p id = "para"></p>
8. <p id = "para1"></p>
9. <script>
10. function fun() {
11.     alert("Welcome to the javaScript ");
12. }
```

```
13.
14.  function fun1() {
15.     document.getElementById("para").innerHTML =  "This is second function";
16.
17.  }
18.  function fun2() {
19.     document.getElementById("para1").innerHTML =  "This is third function";
20.  }
21.  var mybtn = document.getElementById("btn");
22.  mybtn.addEventListener("click", fun);
23.  mybtn.addEventListener("click", fun1);
24.  mybtn.addEventListener("click", fun2);
25.  </script>
26.  </body>
27.  </html>
```