TEMA 6. Eventos y formularios.

Desarrollo Web en Entorno Cliente.

Profesor: Juan José Gallego García

Índice :

Eventos.

Como ya sabemos los eventos se dan cuando el usuario interactúa con los elementos del documento HTML (p.e. al hacer clic), o cuando automáticamente se detectan cambios que han ocurrido. (p.e. carga completa del documento).

Los elementos HTML pueden contener controladores de eventos como atributos para que JS realice una acción determinada.

En esta unidad vamos a estudiar los eventos estándares que pueden reconocer la mayoría de los navegadores ya que cada navegador puede tener también otros propios.

En el siguiente ejemplo podemos ver cómo se establece el controlador de evento onclick del botón para llamar a la función fecha ( ) cuando se activa el evento clic .

```html
<!DOCTYPE html>
<html>
<head>
  <script>
    function fecha(){
        document.getElementById("pFecha").innerHTML = Date();
    }
  </script>
</head>
<body>
  <button onclick="fecha()">Ver fecha y hora</button>
  <p id="pFecha"></p>
</body>
</html>
```

También se puede realizar una acción sobre el propio elemento que contiene el evento mediante this

```html
<p onclick="this.innerHTML='modificado'" >Texto ...</p>
 <!-- Al hacer click en el párrafo 'Texto...' lo modifica.-->
```

O enviar el propio elemento como parámetro en la función que se llama.

```
function fecha(obj){
   alert (obj.innerHTML); // Muestra 'Ver fecha y hora'
}
.....
<button onclick="fecha(this)">Ver fecha y hora</button>
```

Formas de establecer controladores de eventos.

1ª) **Controladores de eventos en línea.** Como ya hemos visto son los establecidos como atributos en un elemento HTML. `<button onclick="fecha(this)">Ver fecha y hora</button>`

No es aconsejable usar esta forma, se considera mala práctica ya que se mezcla HTML y JS, además que se vuelve poco práctico cuando hay que establecer varios/muchos elementos.

2ª) **Como propiedades de controladores de eventos.** Se establece el controlador como una propiedad más del elemento con código JS.

Ej:

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.      <script>
5.          function fecha(){
6.             document.getElementById("pFecha").innerHTML = Date();
7.          }
8.          function f1(){
9.            var btn=document.getElementById("bot");
10.           btn.onclick=fecha;
11.
12.           //btn.onclick=''; //Eliminar evento.
13.         }
14.         window.onload=f1;
15.     </script>
16. </head>
17. <body>
18.   <button id="bot">Ver fecha y hora</button>
19.   <p id="pFecha"></p>
20. </body>
21. </html>
```

3ª) Usando los métodos addEventListener () y removeEventListener ()

Sintaxis: *elemento.addEventListener(evento,función, modoCaptura);*

- ***evento,*** nombre del evento.
- ***función,*** función que se ejecuta cuando se da el evento.
- ***modoCaptura(opcional),*** *establece en qué orden se ejecutan los controladores de eventos cuando existen elementos anidados con el mismo controlador (por ejemplo* ***clic****).Es opcional, por defecto* ***false,*** *se ejecuta antes el interno,* ***true*** *se ejecuta antes el externo.*

```
var btn=document.getElementById("bot");
btn.addEventListener('click',fecha); // Función que devuelve la fecha y hora.
```

```
btn.removeEventListener('click',fecha) // Borra controlador de eventos.
```

Podemos observar que el evento se establece sin 'on' sólo 'click'.
Forma sin enviar parámetros a la función. Ver a continuación cuando es necesario enviar parámetros.

```
var btn=document.getElementById("bot");
let fun=function(){fecha('valor')}; // Es necesario crear una referencia.
btn.addEventListener('click',fun);
btn.removeEventListener('click',fun) // Borra controlador de eventos.
```

Forma cuando se necesita enviar parámetros a la función y poder eliminar en cualquier momento el controlador de evento.

Eventos más comunes :

Evento	Descripción
onchange	Un elemento HTML ha cambiado.
onclick	Se ha hecho clic en un elemento.
onmouseover	El ratón se pasa por encima de un elemento.
onmouseout	El ratón sale de un elemento.
onkeydown	El usuario pulsa una tecla.
onload	El navegador ha acabado de cargar una página.

Lista completa de eventos y sus propiedades :

https://www.w3schools.com/jsref/dom_obj_event.asp

Formularios.

Aunque en el siguiente tema ya veremos cómo JS interactúa con los elementos html del documento (**DOM**), vamos a hacer una introducción con aquellos relacionados con los formularios (*elementos <input>*) ***para centrarnos principalmente en la validación*** de éstos del lado del cliente antes de enviar información al servidor.

Formas de validar formularios.

Cuando validamos un formulario estamos comprobando varias cosas como:

- Si los campos son obligatorios o no.
- Si el contenido solicitado corresponde al tipo o formato a enviar (fecha, email,etc…).
- Si el dato debe ser numérico, entre otros ...

Con HTML5 se introdujo nuevos atributos en los *elementos <input>* para facilitar la validación de formularios, pero si tenemos que realizar alguna otra operación durante la validación o personalizar los mensajes al usuario, es necesario usar JS. Tenemos:

1º) **Validación automática mediante atributos HTML**. En el siguiente ejemplo es obligatorio introducir datos en el formulario con el atributo required. Automáticamente se interrumpe la llamada al archivo especificado en el atributo action mientras no pase la validación. En la validación no es necesario JS.

```
<form id="f1" action="index.html"  method="get">
     <p>Su nombre: <input type="text" name="nombre" required /></p>
     <p>Su edad: <input type="text" name="edad" required /></p>
      <p><input type="submit" /></p>
</form>
```

Archivo index.html

Atributos para elementos <input> HTML :

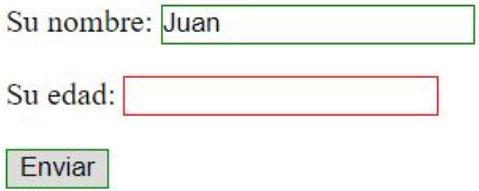
- **disabled** Desactiva el elemento <input>.
- **max** Especifica el máximo valor numérico.
- **min** Especificar el mínimo valor numérico.
- **pattern (regexp)** Especifica un patrón.
- **required** Especifica elemento <input> requerido.
- **type** Especifica el tipo de un <input>.

Otros atributos de <input> HTML

2º) **Validación usando selectores de pseudo-clases CSS.** Complementa el uso de atributos de restricción con pseudo-clases CSS indicando visualmente si cumple con la validación. Si añadimos los siguientes selectores CSS con <style> en el documento HTML del ejemplo anterior con el formulario :

```
<style type="text/css">
    input:invalid {border: 1px solid red;}
    input:valid {border: 1px solid green;}
 </style>
```

, obtenemos →



Como el <input> del nombre pasa la validación el borde es verde, pero al estar vacío el <input> de edad y ser requerido, el borde es rojo, según establecimos en los selectores CSS.

Principales selectores para **input:** CSS:

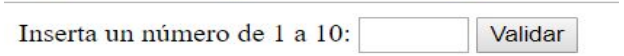
- **:disabled** Selecciona los <input> que tengan el atributo "disabled".
- **:invalid** Selecciona los <input> con valores inválidos.
- **:optional** Selecciona los <input> que no tengan el atributo "required".
- **:required** Selecciona los <input> que tengan el atributo "required".
- **:valid** Selecciona los <input> con valores válidos.

3º) **Validación usando JavaScript.** La mayoría de los navegadores soportan la API de validación de restricciones compuesto por métodos y propiedades que pueden manejarse con JS.

Método checkValidity( ). Este método devuelve true si el elemento <input> contiene datos válidos.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" >
    <script>
        function valida ( ) {
            var inp=document.getElementById ("inp");
            var mensa=document.getElementById ("mensa");
            if(!inp.checkValidity()) {
                mensa.innerHTML="Dato inválido";
             }else{
                mensa.innerHTML="";
             }
        }
        function fEvento( ) {
            var formu=document.getElementById("bot");
            formu.addEventListener('click',valida);  //Añade control evento 'clic'
        }
       window.onload=fEvento; // Necesario esperar que se carguen los elementos HTML
    </script>
</head>
<body>
    <label>Inserta un número de 1 a 10:</label> <input id="inp" type="number" max=10 min=1 required />
    <button id="bot" >Validar</button>
    <p id="mensa"></p>
</body>
</html>
```

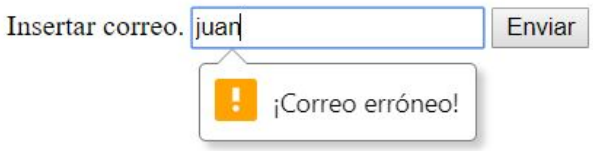
En este ejemplo se puede ver el uso del método **checkValidity( ) ,** no es necesario usar formulario para hacer un envío (submit) ya que sólo hacemos una validación del <input> al hacer 'clic'.



Método setCustomValidity() . Permite establecer un mensaje personalizado cuando los datos de entrada no son válidos. Para ello utiliza la propiedad validity que veremos a continuación.

```
<script>
  function valida ( ) {
      var email = document.getElementById("mail");
      if (email.validity.typeMismatch) {
         email.setCustomValidity("¡Correo erróneo!");
      } else {
         email.setCustomValidity("");
      }
  }
  function fEvento( ) {
      var email = document.getElementById("mail");
      email.addEventListener("keyup", valida);
  }
  window.onload=fEvento;
</script>
```

```
<form>
<label>Insertar correo</label>
<input type="email" id="mail" name="mail" required>
<button>Enviar</button>
</form>
```



Propiedades :

- ***validity :*** *Contiene otras propiedades más específicas relacionadas con la validez del dato.*
- ***validationMessage :*** *Contiene el mensaje, personalizado o por defecto, que va a mostrar cuando la propiedad validity está en false.*
- ***willValidate :*** *Indica si el elemento <input> es validable.*

Propiedades de validity:

- ***validity.customError :*** *Devuelve true si el elemento tiene un error personalizado.*
- ***validity.patternMismatch:*** *Devuelve true si el valor del elemento no cumple el patrón indicado.*
- ***validity.rangeOverflow:*** *Devuelve true si el valor del elemento es mayor al máximo permitido.*
- ***validity.rangeUnderflow:****Devuelve true si el valor del elemento ese menor al mínimo permitido.*
- ***validity.stepMismatch:*** *Devuelve true si el valor del elemento no cumple las reglas indicadas por el atributo step.*
- ***validity.tooLong:*** *Devuelve true si el valor del elemento es mayor al indicado como longitud máxima.*
- ***validity.typeMismatch:*** *Devuelve true si el valor del elemento no tiene la sintaxis correcta.*
- ***validity.valid:*** *Devuelve true si el valor del elemento no tiene problemas de validación.*
- ***validity.valueMissing:*** *Devuelve true si el elemento no tiene valor pero es un campo requerido.*

Validación usando RegExp (expresión regular) en atributo pattern:

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto. En JavaScript, las expresiones regulares también son objetos. Estos patrones se pueden utilizar en métodos **exec**, **match** y **test** de RegExp, en métodos **replace**, **search** y **split** de String, y en la validación de formularios con el atributo pattern de un elemento <input> HTML.

Sintaxis de un objeto **RegExp** : */patrón/modificadores* o con *new RegExp ('expresión')*

Ej: `var pat1=/iesremedios/i;` ó `var pat1 = new RegExp(/[Ii]esremedios/);`

- Se podría usar para buscar la cadena "iesremedios" y con el modificador **i** le decimos que no distinga entre mayúsculas y minúsculas.

Ej: en atributo **pattern**: Equivalente a la búsqueda anterior `pattern="[Ii]esremedios"` , válido si la entrada comienza con 'i' mayúscula o minúscula y contiene la cadena . *(Sobre expresiones regulares...)* *(Más sobre expresiones regulares...)*

Ver un ejemplo completo a continuación….

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" >
    <script>
        function valida ( ) {
            var email = document.getElementById("mail");
            if (email.validity.typeMismatch) {
                email.setCustomValidity("¡Correo erróneo!");
            } else {
                email.setCustomValidity("");
            }
        }
        function fEvento( ) {
            var bot = document.getElementById("bot");
            bot.addEventListener("click", valida);
        }
       window.onload=fEvento; // Necesario esperar que se carguen los elementos HTML
    </script>
</head>
<body>
        <form id="form1" action="index.html">
            <label>Insertar DNI.</label>
            <input type="text" id="id" name="dni" pattern="^[0-9]{8}[TRWAGMYFPDXBNJZSQVHLCKET]$"  required><br><br>
            <label>Insertar correo</label>
            <input type="email" id="mail" name="mail" required><br><br>
            <button id="bot">Enviar</button>
        </form>
</body>
</html>
```

Parámetro (evento) : Si usamos un control total de validación mediante JS, puede darse el caso de que aunque mostremos un mensaje de validación al usuario, no quita que se realice la llamada a la página establecida en el parámetro action del formulario. Para interrumpir la llamada usamos el parámetro de evento pasado a la función junto con el método preventDefault().

```
<script>
  function valida (e) {
     var nombre=document.forms["form1"]["nombre"].value;
     var edad = document.forms["form1"].edad.value; // ó como sigue …
     var edad = document.getElementById("edad1").value;
     if (nombre=="" || edad=="") {
     alert ("Rellenar campo..." );
     e.preventDefault (); // Interrumpe la llamada al archivo al que apunta
                          // el formulario con 'action'.
     }
  }
  function fEvento ( ) {
     var formu=document.getElementById ("f1");
     formu.addEventListener ('submit',valida);
  }
  window.onload=fEvento; // Necesario esperar que se carguen los elementos
HTML
</script>
```

```
<form id="f1" name="form1" action="index.html"  method="get">
    <p>Su nombre: <input type="text" name="nombre" /></p>
    <p>Su edad: <input id="edad1" type="text" name="edad"
/></p>
    <p><input type="submit" /></p>
</form>
```

Bibliografía

- LibrosWeb
- W3Schools
- Developer Mozilla Docs