**Javascript - Formularios**

[JavaScript Form Validation](http://www.w3schools.com/js/js_validation.asp)

[HTML DOM Form submit() Method](http://www.w3schools.com/jsref/met_form_submit.asp)

[HTML DOM Form Object](http://www.w3schools.com/jsref/dom_obj_form.asp)

[ttps://uniwebsidad.com/libros/javascript/capitulo-7/validacion](https://uniwebsidad.com/libros/javascript/capitulo-7/validacion)

Los formularios en la web fueron el primer objetivo del lenguaje de programación JavaScript y actualmente siguen haciendo un uso intensivo de scripts, con el objetivo de aumentar la interacción inmediata con el usuario.

De esta manera, la validación de formularios mediante JavaScript sigue siendo útil, ya que acelera la corrección de errores, pero no implica que la validación por parte del lado del servidor no deba realizarse, ya que ésta es inevitable.

Respecto a la validación de formularios, se deben tener en cuenta dos tipos de estrategia en el proceso de validación: en tiempo real y al enviar el formulario.

**a)** La **validación en tiempo real** se basa en la detección de cierta actividad y la llamada a una función cuando ésta ocurre.

La principal ventaja de la validación en tiempo real es que el usuario todavía tiene reciente la información introducida en el campo, es decir, cuando el usuario acaba de introducir el contenido de un campo, en caso de un error en el contenido, se puede avisar inmediatamente al usuario.

Eventos relacionados con elementos de formularios en los que el usuario tenga que introducir algún tipo de información en un cuadro de texto o similar:

[**oncontextmenu**](http://www.w3schools.com/tags/ev_oncontextmenu.asp)

Cuando se pulsa el botón derecho del ratón sobre un elemento. Este evento funciona con los navegadores, pero las estiquetas de HTML [**<menu>**](http://www.w3schools.com/jsref/dom_obj_menu.asp) y [**<menuitem>**](http://www.w3schools.com/jsref/dom_obj_menuitem.asp) solo funcionan con Firefox por ahora. Ejemplo:

<http://www.w3schools.com/tags/tryit.asp?filename=tryhtml5_menuitem>

[**oninput**](http://www.w3schools.com/tags/ev_oninput.asp)

Cuando los valores de alguno de los siguientes elementos cambien.

**<input type="password">, <input type="search">, <input type="text"> y <textarea>**

**oninvalid**

Un elemento no es válido.

[**onchange**](http://www.w3schools.com/tags/ev_onchange.asp)

Se lanza cuando un elemento select, text o textarea de un formulario **pierde el foco y además su valor ha sido cambiado**, esto es, si se está introduciendo datos en una casilla de texto, el evento no se activará hasta que no se pase a otra casilla. Para eso podríamos usar onkeyup o el evento oninput, por ejemplo.

El evento onchange se puede utilizar con los siguientes elementos:

**<input type="checkbox">, <input type="file">, <input type="password">, <input type="radio">, <input type="range">, <input type="search">, <input type="text">, <keygen>, <select> y <textarea>**

[**onfocus**](http://www.w3schools.com/tags/ev_onfocus.asp)

Detecta cuando un elemento del formulario recibe el foco. Se utiliza sobre todo con input, textarea o window.

[**onblur**](http://www.w3schools.com/tags/ev_onblur.asp)

Detecta la pérdida del enfoque.

[**onreset**](http://www.w3schools.com/tags/ev_onreset.asp)

Cuando se pulsa el botón reset de un formulario.

[**onsearch**](http://www.w3schools.com/tags/ev_onsearch.asp)

Cuando se introduce algo en un campo tipo **<input type="search">**

No funciona en IE ni en Firefox

[**onselect**](http://www.w3schools.com/tags/ev_onselect.asp)

Cuando se ha seleccionado texto dentro de un elemento.

**b)** La **validación al enviar el formulario** se basa en la comprobación de cada uno de los componentes del formulario justo antes de que éste sea enviado al servidor. De esta manera, a partir del evento submit del objeto Form se lanza el script que realiza el conjunto de validaciones.

[**onsubmit**](http://www.w3schools.com/tags/ev_onsubmit.asp)

Cuando se pulsa el botón de enviar un formulario.

onkeyup, onkeydown, onkeypress

<http://www.w3schools.com/jsref/event_onkeydown.asp>

**Validación al enviar el formularios**

Generalmente, cuando necesitemos reunir información sobre los usuarios tendremos que recurrir a un formulario.

Una vez que el usuario rellena el formulario, pulsa el botón de envío para enviar la información al servidor web, donde un programa interpretará los datos y actuará sobre ellos.

Con frecuencia los datos se almacenan en una base de datos para su uso posterior, por lo que será útil asegurar que los datos introducidos por el usuario están ajustados y en el formato adecuado.

Mediante Javascript podemos verificar la información directamente en el cliente. A esto es a lo que se llama “validación del formulario”. Aunque un programa de servidor puede hacer la validación, es mucho más rápido utilizar Javascript.

**Objetos, propiedades y métodos de formularios**

Un formulario es un objeto que a su vez contiene otros objetos como botones, casillas de texto, etc. Veamos las propiedades, métodos y eventos más importantes que se utilizan para describir estos objetos y los procesos relacionados con ellos.

**OBJETO form**

<http://www.w3schools.com/jsref/dom_obj_form.asp>

**Colección de elementos:**

[elements](http://www.w3schools.com/jsref/coll_form_elements.asp)

Devuelve los elementos del formulario.

**Propiedades:**

[length](http://www.w3schools.com/jsref/prop_form_length.asp)

Devuelve el número de elementos del formulario.

**Métodos:**

[reset()](http://www.w3schools.com/jsref/met_form_reset.asp)

Restablece los valores del formulario.

[submit()](http://www.w3schools.com/jsref/met_form_submit.asp)

Envía el formulario.

**Elementos del formulario:**

Algunos elementos del formulario con algunas de sus propiedades y métodos son los siguientes.

HTMLInput element.

<https://developer.mozilla.org/es/docs/Web/API/HTMLInputElement>

**Input Button**

<http://www.w3schools.com/jsref/dom_obj_button.asp>

**Input Reset**

<http://www.w3schools.com/jsref/dom_obj_reset.asp>

**Input Submit**

<http://www.w3schools.com/jsref/dom_obj_submit.asp>

**Input Number**

<http://www.w3schools.com/jsref/dom_obj_number.asp>

Ver: ej\_number.html

**Input Password**

<http://www.w3schools.com/jsref/dom_obj_password.asp>

**Input Text**

<http://www.w3schools.com/jsref/dom_obj_text.asp>

**Textarea**

http://www.w3schools.com/jsref/dom\_obj\_textarea.asp

[**Input Checkbox**](http://www.w3schools.com/jsref/dom_obj_checkbox.asp)

<http://www.w3schools.com/jsref/dom_obj_checkbox.asp>

Propiedades:

[checked](http://www.w3schools.com/jsref/prop_checkbox_checked.asp)

Si la casilla está seleccionada o no.

[defaultChecked](http://www.w3schools.com/jsref/prop_checkbox_defaultchecked.asp)

Si la casilla está seleccionada o no por defecto. Booleano.

defaultValue

Valor por defecto del checkbox

**Input Radio**

<http://www.w3schools.com/jsref/dom_obj_radio.asp>

Propiedades:

[checked](http://www.w3schools.com/jsref/prop_checkbox_checked.asp)

Si la casilla está seleccionada o no.

[defaultChecked](http://www.w3schools.com/jsref/prop_checkbox_defaultchecked.asp)

Si la casilla está seleccionada o no por defecto. Booleano.

defaultValue

Valor por defecto del checkbox

**Select**

<http://www.w3schools.com/jsref/dom_obj_select.asp>

El objeto “Select” tiene la propiedad “options”. Esta propiedad es un vector que contiene los elementos de la lista. Para hacer referencia a cada uno de ellos utilizaremos su índice dentro de este vector.

**Propiedades:**

| [length](http://www.w3schools.com/jsref/prop_select_length.asp) | Número de elementos de la lista. |
| --- | --- |
| [multiple](http://www.w3schools.com/jsref/prop_select_multiple.asp) | Si se permite selección múltiple o no. |
| [selectedIndex](http://www.w3schools.com/jsref/prop_select_selectedindex.asp) | Índice de la opción seleccionada. |
| [size](http://www.w3schools.com/jsref/prop_select_size.asp) | Tamaño de la lista desplegable. |
| [value](http://www.w3schools.com/jsref/prop_select_value.asp) | Valor de la opción seleccionada. |

**Métodos:**

| [add()](http://www.w3schools.com/jsref/met_select_add.asp) | Añadir una opción a la lista. |
| --- | --- |
| [remove()](http://www.w3schools.com/jsref/met_select_remove.asp) | Elimina una opción de la lista. |

**Colección de elementos:**

[options](http://www.w3schools.com/jsref/coll_select_options.asp)

Devuelve una colección de todos los elementos de la lista.

**Propiedades de options:**

| length | Número de opciones. |
| --- | --- |
| selectedIndex | Índice de la opción seleccionada. Empieza en el número 0. |

**Métodos de options:**

| [index] | Devuelve el elemento cuyo índice es index.  Empieza en 0.  Si está fuera de rango, devuelve null. |
| --- | --- |
| add(option[,index]) | Añade una opción a la lista, si no se especifica el index, lo añade al final. |
| item(*index*) | Devuelve el elemento con el índice index. Empieza en el 0. Si es fuera de rango, devuelve null. |
| namedItem(*id*) | Devuelve el elemento con el id que se le indique. Si no existe devuelve null. |
| remove(*index*) | Elimina el elemento en el index que se indique. |

**Verificar contraseñas**

En ocasiones se le pide a los usuarios que tecleen una nueva contraseña dos veces para asegurarnos de que no se equivoca al teclearla.

Podemos comunicarle al usuario, por ejemplo mediante ventanas de alerta, si olvidó escribir la contraseña o si las contraseñas no coinciden.

En el caso de que no se cumplan las condiciones exigidas, el manejador de eventos “onsubmit” actuará en consecuencia deteniendo el envío del formulario.

**Ejemplo – Contraseñas: verificacontras.html**

En el momento que el usuario intenta enviar el formulario comprobamos si se cumplen todos los requisitos. Para ello ejecutamos una función que devolverá como valor “true” o “false”, de forma que se envíe o no el formulario según convenga.

La función realiza varias comprobaciones:

Si el valor introducido en la primera casilla de contraseña es “vacío”, alerta al usuario, activa la casilla y devuelve “false” para detener el envío de los datos.

Si el valor de las casillas de contraseña no coincide, alerta al usuario, activa la primera casilla y selecciona su contenido y devuelve “false” para detener el envío de los datos.

En última instancia, si las comprobaciones anteriores no detienen el envío, devuelve “true” para continuar con el evento “submit”.

<!DOCTYPE html>

<html lang="es">

<meta charset="utf-8" />

<title></title>

<script>

function validar(formulario){

let pass1=document.getElementById("pass1");

let pass2=document.getElementById("pass2");

if (pass1.value==""){

alert("Debe introducir una contraseña");

pass1.focus();

return false;

}

if (pass1.value != pass2.value){

alert("Las contraseñas introducidas no coinciden")

pass1.focus();

pass1.select();

return false;

}

return true;

}

</script>

</head>

<body>

<form action="" onsubmit="return validar()">

<label for=”pass1”>Escriba una contraseña:</label>

<input type="password" name="pass1" id="pass1"><br />

<label for=”pass2”>Repita la contraseña:</label>

<input type="password" name="pass2" id="pass2"><br />

<input type="submit" value="Enviar"><br />

<input type="reset" value="Borrar">

</form>

</body>

</html>

**Lista desplegable de direcciones**

Podemos diseñar una lista desplegable de formulario de forma que, al elegir un elemento de la misma, enlacemos con una página determinada.

No olvidemos que para hacer referencia a una opción determinada de la lista debemos emplear su índice dentro del vector “options”.

**Ejemplo – Lista**: **validalista.html**

En este caso, el valor de cada elemento de la lista será una dirección de enlace, excepto la opción que nos sirve para proporcionar indicaciones al usuario, cuyo valor será “vacío”.

Además, podemos forzar a que cada vez que se cargue el documento el elemento seleccionado sea el de índice “0”. Esto lo haremos manejando el evento “load” en la etiqueta <body>.

Cuando cambiamos el elemento seleccionado de la lista se ejecuta la función que:

En primer lugar comprueba si la opción elegida es la primera. En ese caso no cambia la dirección de la ventana.

Si el elemento seleccionado es cualquier otro, la nueva localización de la ventana será la elegida por el usuario.

Por último, el botón para usuarios con navegador sin soporte para Javascript no lo visualizarán el resto de usuarios. Por lo tanto, el envío del formulario a un programa de servidor sólo se producirá en el caso de que Javascript no se ejecute.

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="utf-8" />

<title>Validar listas</title>

<script>

function pagina(){

let nuevapag=this.options[this.selectedIndex].value;

if(nuevapag != ""){

window.location.href=nuevapag;

}

}

window.onload=function () {

document.getElementById("lista").selectedIndex=0;

document.getElementById("lista").onchange=pagina;

}

</script>

</head>

<body>

<form action="" method="get">

<select name="lista" id="lista">

<option value="">Seleccione un documento</option>

<option value="http://www.google.es">Google</option>

<option value="http://www.yahoo.es">Yahoo</option>

<option value="http://www.gmail.com">Gmail</option>

</select>

</body>

</html>

**Botones de opción**

En un formulario, cada objeto de tipo “radio” sólo nos permite marcar una de las opciones. Si ninguna de ellas está seleccionada por defecto, podemos forzar al usuario a elegir una de ellas antes de enviar el formulario.

**Ejemplo – Botones de opción**

En el momento que el usuario pretende remitir los datos del formulario se ejecuta la función que realiza las comprobaciones oportunas.

La función verifica si alguna de las opciones está seleccionada a través de un bucle que recorre todos los elementos del vector “radio”. Si no ocurre así, comunica al usuario esta incidencia y retorna un valor falso para detener el evento “submit”.

En este ejemplo solamente tenemos dos opciones, de forma que podríamos utilizar la siguiente función:

function validar(persona) {

if (!persona.sexo[0].checked && !persona.sexo[1].checked) {

alert("Especifique el sexo");

return false;

} else {

return true;

}

}

Sin embargo, cuando el número de opciones es mayor, el código se complica y resulta más cómodo trabajar con un bucle.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script>

function validar(formulario){

let opcion=-1;

if (formulario.opc[0].checked){

opcion=0;

}

if (formulario.opc[1].checked) {

opcion=1;

}

if(opcion==-1){

alert("Especifique el sexo");

return false;

} else {

return true;

}

}

</script>

</head>

<body>

<form method="get" action="ver.php" onsubmit="return validar(this)">

<label>Sexo:</label><br />

<input type="radio" name="opc" value="h"><label>Hombre</label><br />

<input type="radio" name="opc" value="m"><label>Mujer</label><br />

<input type="submit" />

<input type="reset" />

</form>

</body>

</html>

**Formularios dinámicos**

**Control del cursor y del foco**

En este subapartado se verán distintas técnicas que mejorarán la usabilidad del formulario, que, aunque puedan ser consideradas como pequeños detalles, ayudan a los usuarios de manera casi transparente.

**Enviar el cursor a un campo determinado**

Cuando se entra en la web de Google, el usuario sólo debe introducir las cadenas clave que buscar y pulsar la tecla Intro. Se trata de un proceso muy simple y es uno de los factores del éxito de Google, pero detrás de este funcionamiento existe código que hace que el campo de texto de búsqueda reciba el foco en cuanto se abra la página de inicio.

El código que envía el cursor al primer campo de texto del formulario es muy simple. En el ejemplo que se presenta a continuación, el código se enlaza al manejador del evento onload del objeto Document, ya que de esta manera, una vez cargada la página web, la primera acción que se realizará será mover el foco al campo de destino:

<body onload="document.formulario.campo.focus()">

En el caso de que el campo de texto tenga un texto predeterminado, es una ayuda que este texto aparezca inicialmente seleccionado, ya que de este modo, cuando el usuario pulse cualquier tecla, se sustituirá el contenido predeterminado por el nuevo texto (evitando el proceso de eliminación del texto predeterminado).

La selección del texto en un campo de texto se realiza con el método select().

Así pues, el código anterior quedaría de la siguiente manera:

<body onload="document.formulario.campo.focus(); document.formulario.campo.select()">

Por lo tanto, en primer lugar, se sitúa el foco y, a continuación, se selecciona el texto contenido en el campo.

Si en el proceso de validación de un campo de texto éste no la supera, normalmente se devuelve el cursor al campo erróneo seleccionando el texto erróneo, con el objetivo de que el usuario sobrescriba el texto con el valor correcto.

Los métodos que permiten estas acciones son focus() y select(), pero deben ser llamados en un cierto punto del código (desde la función de validación). Para ello, a continuación, se plantea una función que encapsula estos dos métodos:

function asignaFoco(nombreForm, nombreCampo) {

var elemento = document.forms[nombreForm].elements[nombreCampo];

elemento.focus();

elemento.select();

}

Sobre la función anterior, se debe tener en cuenta que es válida para campos de tipo texto, ya que en el caso de botones de opción o listas desplegables la selección del contenido no tiene ningún sentido.

**Avanzar el foco mediante la tecla intro**

El movimiento entre los distintos campos del formulario se realiza con la tecla de tabulación, pero el usuario está más habituado al uso de la tecla Intro, por lo que a continuación se va a presentar una técnica que permite navegar por los distintos campos del formulario utilizando la tecla Intro.

Se utiliza el evento keypress en cada campo desde el que debe avanzar el foco a otro campo del formulario. El manejador de eventos llamará a la función siguienteFoco(), que se presenta a continuación:

<input type="text" name="campo1" id="campo1" onkeypress="return siguienteFoco(this.form, "campo2", event)">

La función siguienteFoco() realiza lo siguiente:

• En primer lugar, se debe comprobar que la tecla presionada es Intro, que tiene los códigos ASCII 13 y 3.

• En segundo lugar, aplica el foco al campo siguiente que ha sido pasado como parámetro de la función:

function siguienteFoco(form, siguienteCampo, evt) {

evt = (evt) ? evt : event;

var codCaracter = (evt.charCode) ? evt.charCode :

((evt.which) ? evt.which : evt.keyCode);

if (codCaracter == 13 || codCaracter == 3) {

form.elements[siguienteCampo].focus();

return false;

}

return true;

}

**Control de los campos del formulario**

Existen dos mecanismos para evitar o bloquear el acceso a ciertos campos de un formulario:

**a) Desactivar un campo**, de manera que éste aparece visible al usuario, pero no está activo, lo que implica que no se pueda introducir texto ni que éste se pueda modificar.

**b) Ocultar un campo**. En este caso, el campo desaparece del formulario y no es visible para el usuario final, aunque el campo sí que existe realmente.

En este subapartado, se van a estudiar los dos mecanismos que permiten desactivar y ocultar campos mediante unos ejemplos sencillos.

**Desactivar campos**

La desactivación de campos de formulario se implementa a partir de la propiedad disabled, que admite un valor booleano:

document.formulario.campoTexto.disabled = true;

El código anterior desactivaría el campo "campoTexto", de manera que no sería accesible para el usuario y, en general, mostraría un aspecto sombreado de color gris. Aunque los scripts pueden leer y escribir valores en los campos desactivados, éstos no son enviados al servidor si no se activan.

La activación del anterior campo se realizaría con la siguiente sentencia:

document.formulario.campoTexto.disabled = false;

**Ocultar y mostrar controles**

Es posible ocultar ciertos campos de un formulario, de manera que sólo se muestren en el caso de que se cumpla cierta condición, por ejemplo que el usuario haya seleccionado alguna opción en algún campo de selección o marque una casilla de selección.

A continuación, se presenta un ejemplo de formulario que dispone de un conjunto de campos ocultos, que son mostrados al usuario final cuando éste responde afirmativamente a la tercera pregunta:

<form name="cuestionario">

...

<p>3. ¿Quieres conocer los nuevos modelos?<br />

<input type="radio" id="resp0" name="respuesta" onclick="gestDecision(event)"

/>No

<input type="radio" id="resp1" name="respuesta" onclick="gestDecision(event)"

/>Si

<div id="conocerModelos" style="display:none; margin-left:20px">

</p>

<p>

3a. ¿Cuál es tu marca preferida?

<select name="modelos">

<option value="">Elige uno:</option>

<option value="1">Ferrari</option>

<option value="2">Mercedes</option>

<option value="3">BMW</option>

</select>

</p>

</div>

<p>4. ¿Cuántos años tiene tu actual vehículo?

<select name="edadVehiculo">

<option value="">Elige una opción:</option>

<option value="1">Menos de 3 años:</option>

<option value="2">Entre 3 y 7 años</option>

<option value="3">Más de 7 años</option>

</select>

</p>

...

</form>

En el código anterior se ha utilizado una capa para introducir en ella la lista desplegable de las marcas. De esta manera, en principio, la lista está oculta al establecer el atributo display a none de la capa, no se muestra inicialmente y es el manejador del evento click de las opciones radio el que cambiará el estado de la capa:

function gestDecision(evt) {

evt = (evt) ? evt : event;

var campo = (evt.target) ? evt.target : evt.srcElement;

var capa = document.getElementById("conocerModelos");

if (campo.id =="resp1") {

capa.style.display = "block";

} else {

capa.style.display = "none";

}

}

Como se observa en la función, se utiliza el objeto Event para capturar el campo y, a continuación, se asigna a la variable *capa* el contenedor de los campos, de manera que si el campo seleccionado es "resp1", se muestra la capa oculta o siguen los campos ocultos en caso contrario.

**Permitir sólo un tipo de datos en un campo**

En ciertas aplicaciones web, es interesante restringir el tipo de datos que se puede introducir en un campo de texto. Posteriormente, se plantean ejemplos que definen una función, que es llamada desde el evento keypress.

En primer lugar, la siguiente función restringe la entrada de valores a los números entre el 0 y el 9, además, permite la introducción de valores ASCII menores que el 32, ya que entre estos valores se encuentran los caracteres alfanuméricos que incluyen las teclas Retroceso 8, Tabulador 9 e Intro 13:

function soloNumeros(evt){

evt = (evt) ? evt : event;

var codCaracter = (evt.keyCode) ? evt.keyCode : (evt.which) ?

evt.which : 0;

if (codCaracter > 31 && (codCaracter < 48 || codCaracter > 57)) {

alert("Introduce sólo valores numéricos!!!");

return false;

}

return true;

}

En el siguiente ejemplo se plantea una función que valida la introducción de varias letras. Para ello, se debe tener en cuenta el hecho de que las mayúsculas y minúsculas tienen distintos códigos y además no son correlativos. La siguiente función realiza la comprobación de la introducción de letras, tanto en mayúsculas como en minúsculas:

function soloLetras(evt){

evt = (evt) ? evt : event;

var codCaracter = (evt.keyCode) ? evt.keyCode : (evt.which) ?

evt.which : 0;

if (codCaracter > 31 && (codCaracter < 65 || codCaracter > 90) && (codCaracter <

97 || codCaracter > 122)) {

alert("Introduce sólo letras!!!");

return false;

}

return true;

}