**Acceso directo a los nodos**

### getElementsByTagName

### getElementsByName

### getElementById

$('#icon')

# Creación y eliminación de nodos

### Creación de elementos XHTML simples

Crear y añadir a la página un nuevo elemento XHTML sencillo consta de cuatro pasos diferentes:

1. Creación de un nodo de tipo Element que represente al elemento.
2. Creación de un nodo de tipo Text que represente el contenido del elemento.
3. Añadir el nodo Text como nodo hijo del nodo Element.
4. Añadir el nodo Element a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

// Crear nodo de tipo Element

var parrafo = document.createElement("p");

// Crear nodo de tipo Text

var contenido = document.createTextNode("Hola Mundo!");

// Añadir el nodo Text como hijo del nodo Element

parrafo.appendChild(contenido);

// Añadir el nodo Element como hijo de la pagina

document.body.appendChild(parrafo);

### Eliminación de nodos

En este caso, solamente es necesario utilizar la función removeChild():

var parrafo = document.getElementById("provisional");

parrafo.parentNode.removeChild(parrafo);

# Acceso directo a los atributos XHTML

Los atributos XHTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo XHTML detrás del nombre del nodo.

var enlace = document.getElementById("enlace");

alert(enlace.href); // muestra http://www...com

<a id="enlace" href="http://www...com">Enlace</a>

Las propiedades CSS no son tan fáciles de obtener como los atributos XHTML. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo style. El siguiente ejemplo obtiene el valor de la propiedad margin de la imagen:

var imagen = document.getElementById("imagen");

alert(imagen.style.margin);

<img id="imagen" style="margin:0; border:0;" src="logo.png" />

Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

var parrafo = document.getElementById("parrafo");

alert(parrafo.style.fontWeight); // muestra "bold"

<p id="parrafo" style="font-weight: bold;">...</p>

El único atributo XHTML que no tiene el mismo nombre en XHTML y en las propiedades DOM es el atributo class. Como la palabra class está reservada por JavaScript, no es posible utilizarla para acceder al atributo class del elemento XHTML. En su lugar, DOM utiliza el nombre className para acceder al atributo class de XHTML:

var parrafo = document.getElementById("parrafo");

alert(parrafo.class); // muestra "undefined"

alert(parrafo.className); // muestra "normal"

<p id="parrafo" class="normal">...</p>

# Modelos de eventos

### Modelo básico de eventos

Este modelo simple de eventos se introdujo para la versión 4 del estándar HTML y se considera parte del nivel más básico de DOM. Aunque sus características son limitadas, es el único modelo que es compatible en todos los navegadores y por tanto, el único que permite crear aplicaciones que funcionan de la misma manera en todos los navegadores.

### Modelo de eventos estándar

Las versiones más avanzadas del estándar DOM (DOM nivel 2) definen un modelo de eventos completamente nuevo y mucho más poderoso que el original. Todos los navegadores modernos lo incluyen, salvo Internet Explorer.

### Modelo de eventos de Internet Explorer

Internet Explorer utiliza su propio modelo de eventos, que es similar pero incompatible con el modelo estándar. Se utilizó por primera vez en Internet Explorer 4 y Microsoft decidió seguir utilizándolo en el resto de versiones, a pesar de que la empresa había participado en la creación del estándar de DOM que define el modelo de eventos estándar.

# Modelo básico de eventos

### Tipos de eventos

El nombre de cada evento se construye mediante el prefijo on, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina onclick y el evento asociado a la acción de mover el ratón se denomina onmousemove.

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

|  |  |  |
| --- | --- | --- |
| Evento | Descripción | Elementos para los que está definido |
| onblur | Deseleccionar el elemento | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onchange | Deseleccionar un elemento que se ha modificado | <input>, <select>, <textarea> |
| onclick | Pinchar y soltar el ratón | Todos los elementos |
| ondblclick | Pinchar dos veces seguidas con el ratón | Todos los elementos |
| onfocus | Seleccionar un elemento | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onkeydown | Pulsar una tecla (sin soltar) | Elementos de formulario y <body> |
| onkeypress | Pulsar una tecla | Elementos de formulario y <body> |
| onkeyup | Soltar una tecla pulsada | Elementos de formulario y <body> |
| onload | La página se ha cargado completamente | <body> |
| onmousedown | Pulsar (sin soltar) un botón del ratón | Todos los elementos |
| onmousemove | Mover el ratón | Todos los elementos |
| onmouseout | El ratón *"sale"* del elemento (pasa por encima de otro elemento) | Todos los elementos |
| onmouseover | El ratón *"entra"* en el elemento (pasa por encima del elemento) | Todos los elementos |
| onmouseup | Soltar el botón que estaba pulsado en el ratón | Todos los elementos |
| onreset | Inicializar el formulario (borrar todos sus datos) | <form> |
| onresize | Se ha modificado el tamaño de la ventana del navegador | <body> |
| onselect | Seleccionar un texto | <input>, <textarea> |
| onsubmit | Enviar el formulario | <form> |
| onunload | Se abandona la página (por ejemplo al cerrar el navegador) | <body> |

Los eventos más utilizados en las aplicaciones web tradicionales son onload para esperar a que se cargue la página por completo, los eventos onclick, onmouseover, onmouseout para controlar el ratón y onsubmit para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (onclick, onkeydown, onkeypress, onreset, onsubmit) permiten evitar la *"acción por defecto"* de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

### Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede responder ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan "manejador de eventos" y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

* Manejadores como atributos de los elementos XHTML.
* Manejadores como funciones JavaScript externas.
* Manejadores "semánticos".

#### Manejadores de eventos como atributos XHTML

Se trata del método más sencillo y a la vez menos profesional de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo del propio elemento XHTML. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />

En este método, se definen atributos XHTML con el mismo nombre que los eventos que se quieren manejar. El ejemplo anterior sólo quiere controlar el evento de pinchar con el ratón, cuyo nombre es onclick. Así, el elemento XHTML para el que se quiere definir este evento, debe incluir un atributo llamado onclick.

El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento. En este caso, el código JavaScript es muy sencillo (alert('Gracias por pinchar');), ya que solamente se trata de mostrar un mensaje.

En este otro ejemplo, cuando el usuario pincha sobre el elemento <div> se muestra un mensaje y cuando el usuario pasa el ratón por encima del elemento, se muestra otro mensaje:

<div onclick="alert('Has pinchado con el ratón');" onmouseover="alert('Acabas de pasar el ratón por encima');">

Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima

</div>

Este otro ejemplo incluye una de las instrucciones más utilizadas en las aplicaciones JavaScript más antiguas:

<body onload="alert('La página se ha cargado completamente');">

...

</body>

El mensaje anterior se muestra después de que la página se haya cargado completamente, es decir, después de que se haya descargado su código HTML, sus imágenes y cualquier otro objeto incluido en la página.

#### Manejadores de eventos y variable this

JavaScript define una variable especial llamada this que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable this para referirse al elemento XHTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente:

Cuando el usuario pasa el ratón por encima del <div>, el color del borde se muestra de color negro. Cuando el ratón sale del <div>, se vuelve a mostrar el borde con el color gris claro original.

Elemento <div> original:

<div id="contenidos" style="width:150px; height:60px; border:thin solid silver">

Sección de contenidos...

</div>

Si no se utiliza la variable this, el código necesario para modificar el color de los bordes, sería el siguiente:

<div id="contenidos" style="width:150px; height:60px; border:thin solid silver" onmouseover="document.getElementById('contenidos').style.borderColor='black';" onmouseout="document.getElementById('contenidos').style.borderColor='silver';">

Sección de contenidos...

</div>

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable this, que hace referencia al elemento XHTML que ha provocado el evento. Así, el ejemplo anterior se puede reescribir de la siguiente manera:

<div id="contenidos" style="width:150px; height:60px; border:thin solid silver" onmouseover="this.style.borderColor='black';" onmouseout="this.style.borderColor='silver';">

Sección de contenidos...

</div>

El código anterior es mucho más compacto, más fácil de leer y de escribir y sigue funcionando correctamente aunque se modifique el valor del atributo id del <div>.

#### Manejadores de eventos como funciones externas

La definición de los manejadores de eventos en los atributos XHTML es el método más sencillo pero menos aconsejable de tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento XHTML.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:

<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />

Utilizando funciones externas se puede transformar en:

function muestraMensaje() {

alert('Gracias por pinchar');

}

<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento XHTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable this y por tanto, es necesario pasar esta variable como parámetro a la función:

function resalta(elemento) {

switch(elemento.style.borderColor) {

case 'silver':

case 'silver silver silver silver':

case '#c0c0c0':

elemento.style.borderColor = 'black';

break;

case 'black':

case 'black black black black':

case '#000000':

elemento.style.borderColor = 'silver';

break;

}

}

<div style="width:150px; height:60px; border:thin solid silver" onmouseover="resalta(this)" onmouseout="resalta(this)">

Sección de contenidos...

</div>

En el ejemplo anterior, la función externa es llamada con el parámetro this, que dentro de la función se denomina elemento. La complejidad del ejemplo se produce sobre todo por la forma en la que los distintos navegadores almacenan el valor de la propiedad borderColor.

Mientras que Firefox almacena (en caso de que los cuatro bordes coincidan en color) el valor black, Internet Explorer lo almacena como black black black black y Opera almacena su representación hexadecimal #000000.

#### Manejadores de eventos semánticos

Los métodos que se han visto para añadir manejadores de eventos (como atributos XHTML y como funciones externas) tienen un grave inconveniente: "ensucian" el código XHTML de la página.

Como es conocido, una de las buenas prácticas básicas en el diseño de páginas y aplicaciones web es la separación de los contenidos (XHTML) y su aspecto o presentación (CSS). Siempre que sea posible, también se recomienda separar los contenidos (XHTML) y su comportamiento o programación (JavaScript).

Mezclar el código JavaScript con los elementos XHTML solamente contribuye a complicar el código fuente de la página, a dificultar la modificación y mantenimiento de la página y a reducir la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos XHTML para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

<input id="pinchable" type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />

Se puede transformar en:

// Función externa

function muestraMensaje() {

alert('Gracias por pinchar');

}

// Asignar la función externa al elemento

document.getElementById("pinchable").onclick = muestraMensaje;

// Elemento XHTML

<input id="pinchable" type="button" value="Pinchame y verás" />

La técnica de los manejadores semánticos consiste en:

1. Asignar un identificador único al elemento XHTML mediante el atributo id.
2. Crear una función de JavaScript encargada de manejar el evento.
3. Asignar la función externa al evento correspondiente en el elemento deseado.

El último paso es la clave de esta técnica. En primer lugar, se obtiene el elemento al que se desea asociar la función externa:

document.getElementById("pinchable");

A continuación, se utiliza una propiedad del elemento con el mismo nombre que el evento que se quiere manejar. En este caso, la propiedad es onclick:

document.getElementById("pinchable").onclick = ...

Por último, se asigna la función externa mediante su nombre sin paréntesis. Lo más importante (y la causa más común de errores) es indicar solamente el nombre de la función, es decir, prescindir de los paréntesis al asignar la función:

document.getElementById("pinchable").onclick = muestraMensaje;

Si se añaden los paréntesis después del nombre de la función, en realidad se está ejecutando la función y guardando el valor devuelto por la función en la propiedad onclick de elemento.

// Asignar una función externa a un evento de un elemento

document.getElementById("pinchable").onclick = muestraMensaje;

// Ejecutar una función y guardar su resultado en una propiedad de un elemento

document.getElementById("pinchable").onclick = muestraMensaje();

La gran ventaja de este método es que el código XHTML resultante es muy "limpio", ya que no se mezcla con el código JavaScript. Además, dentro de las funciones externas asignadas sí que se puede utilizar la variable this para referirse al elemento que provoca el evento.

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos XHTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento onload:

window.onload = function() {

document.getElementById("pinchable").onclick = muestraMensaje;

}

La técnica anterior utiliza el concepto de funciones anónimas, que no se va a estudiar, pero que permite crear un código compacto y muy sencillo. Para asegurarse que un código JavaScript va a ejecutarse después de que la página se haya cargado completamente, sólo es necesario incluir esas instrucciones entre los símbolos { y }:

window.onload = function() {

...

}

En el siguiente ejemplo, se añaden eventos a los elementos de tipo input=text de un formulario complejo:

function resalta() {

// Código JavaScript

}

window.onload = function() {

var formulario = document.getElementById("formulario");

var camposInput = formulario.getElementsByTagName("input");

for(var i=0; i<camposInput.length; i++) {

if(camposInput[i].type == "text") {

camposInput[i].onclick = resalta;

}

}

}

# Obteniendo información del evento (objeto event)

Normalmente, los manejadores de eventos requieren información adicional para procesar sus tareas. Si una función por ejemplo se encarga de procesar el evento onclick, quizás necesite saber en que posición estaba el ratón en el momento de pinchar el botón.

No obstante, el caso más habitual en el que es necesario conocer información adicional sobre el evento es el de los eventos asociados al teclado. Normalmente, es muy importante conocer la tecla que se ha pulsado, por ejemplo para diferenciar las teclas normales de las teclas especiales (ENTER, tabulador, Alt, Ctrl., etc.).

JavaScript permite obtener información sobre el ratón y el teclado mediante un objeto especial llamado event. Desafortunadamente, los diferentes navegadores presentan diferencias muy notables en el tratamiento de la información sobre los eventos.

La principal diferencia reside en la forma en la que se obtiene el objeto event. Internet Explorer considera que este objeto forma parte del objeto window y el resto de navegadores lo consideran como el único argumento que tienen las funciones manejadoras de eventos.

Aunque es un comportamiento que resulta muy extraño al principio, todos los navegadores modernos excepto Internet Explorer crean mágicamente y de forma automática un argumento que se pasa a la función manejadora, por lo que no es necesario incluirlo en la llamada a la función manejadora. De esta forma, para utilizar este "argumento mágico", sólo es necesario asignarle un nombre, ya que los navegadores lo crean automáticamente.

En resumen, en los navegadores tipo Internet Explorer, el objeto event se obtiene directamente mediante:

var evento = window.event;

Por otra parte, en el resto de navegadores, el objeto event se obtiene mágicamente a partir del argumento que el navegador crea automáticamente:

function manejadorEventos(elEvento) {

var evento = elEvento;

}

Si se quiere programar una aplicación que funcione correctamente en todos los navegadores, es necesario obtener el objeto event de forma correcta según cada navegador. El siguiente código muestra la forma correcta de obtener el objeto event en cualquier navegador:

function manejadorEventos(elEvento) {

var evento = elEvento || window.event;

}

Una vez obtenido el objeto event, ya se puede acceder a toda la información relacionada con el evento, que depende del tipo de evento producido.

### Información sobre el evento

La propiedad type indica el tipo de evento producido, lo que es útil cuando una misma función se utiliza para manejar varios eventos:

var tipo = evento.type;

La propiedad type devuelve el tipo de evento producido, que es igual al nombre del evento pero sin el prefijo on.

Mediante esta propiedad, se puede rehacer de forma más sencilla el ejemplo anterior en el que se resaltaba una sección de contenidos al pasar el ratón por encima:

function resalta(elEvento) {

var evento = elEvento || window.event;

switch(evento.type) {

case 'mouseover':

this.style.borderColor = 'black';

break;

case 'mouseout':

this.style.borderColor = 'silver';

break;

}

}

window.onload = function() {

document.getElementById("seccion").onmouseover = resalta;

document.getElementById("seccion").onmouseout = resalta;

}

<div id="seccion" style="width:150px; height:60px; border:thin solid silver">

Sección de contenidos...

</div>

### Información sobre los eventos de teclado

De todos los eventos disponibles en JavaScript, los eventos relacionados con el teclado son los más incompatibles entre diferentes navegadores y por tanto, los más difíciles de manejar. En primer lugar, existen muchas diferencias entre los navegadores, los teclados y los sistemas operativos de los usuarios, principalmente debido a las diferencias entre idiomas.

Además, existen tres eventos diferentes para las pulsaciones de las teclas (onkeyup, onkeypress y onkeydown). Por último, existen dos tipos de teclas: las teclas normales (como letras, números y símbolos normales) y las teclas especiales (como ENTER, Alt, Shift, etc.)

Cuando un usuario pulsa una tecla normal, se producen tres eventos seguidos y en este orden: onkeydown, onkeypress y onkeyup. El evento onkeydown se corresponde con el hecho de pulsar una tecla y no soltarla; el evento onkeypress es la propia pulsación de la tecla y el evento onkeyup hace referencia al hecho de soltar una tecla que estaba pulsada.

La forma más sencilla de obtener la información sobre la tecla que se ha pulsado es mediante el evento onkeypress. La información que proporcionan los eventos onkeydown y onkeyup se puede considerar como más técnica, ya que devuelven el código interno de cada tecla y no el carácter que se ha pulsado.

A continuación se incluye una lista con todas las propiedades diferentes de todos los eventos de teclado tanto en Internet Explorer como en el resto de navegadores:

* Evento keydown:
  + Mismo comportamiento en todos los navegadores:
    - Propiedad keyCode: código interno de la tecla
    - Propiedad charCode: no definido
* Evento keypress:
  + Internet Explorer:
    - Propiedad keyCode: el código del carácter de la tecla que se ha pulsado
    - Propiedad charCode: no definido
  + Resto de navegadores:
    - Propiedad keyCode: para las teclas normales, no definido. Para las teclas especiales, el código interno de la tecla.
    - Propiedad charCode: para las teclas normales, el código del carácter de la tecla que se ha pulsado. Para las teclas especiales, 0.
* Evento keyup:
  + Mismo comportamiento en todos los navegadores:
    - Propiedad keyCode: código interno de la tecla
    - Propiedad charCode: no definido

Para convertir el código de un carácter (no confundir con el código interno) al carácter que representa la tecla que se ha pulsado, se utiliza la función String.fromCharCode().

A continuación se incluye un script que muestra toda la información sobre los tres eventos de teclado:

window.onload = function() {

document.onkeyup = muestraInformacion;

document.onkeypress = muestraInformacion;

document.onkeydown = muestraInformacion;

}

function muestraInformacion(elEvento) {

var evento = window.event || elEvento;

var mensaje = "Tipo de evento: " + evento.type + "<br>" +

"Propiedad keyCode: " + evento.keyCode + "<br>" +

"Propiedad charCode: " + evento.charCode + "<br>" +

"Carácter pulsado: " + String.fromCharCode(evento.charCode);

info.innerHTML += "<br>--------------------------------------<br>" + mensaje

}

...

<div id="info"></div>

Al pulsar la tecla a en el navegador Firefox, se muestra la siguiente sucesión de eventos:

--------------------------------------

Tipo de evento: keydown

Propiedad keyCode: 65

Propiedad charCode: 0

## Carácter pulsado: ?Tipo de evento: keypress

Propiedad keyCode: 0

Propiedad charCode: 97

## Carácter pulsado: aTipo de evento: keyup

Propiedad keyCode: 65

Propiedad charCode: 0

Carácter pulsado: ?

Al pulsar la tecla A (la misma tecla, pero habiendo activado previamente las mayúsculas) se muestra la siguiente sucesión de eventos en el navegador Firefox:

--------------------------------------

Tipo de evento: keydown

Propiedad keyCode: 65

Propiedad charCode: 0

## Carácter pulsado: ?Tipo de evento: keypress

Propiedad keyCode: 0

Propiedad charCode: 65

## Carácter pulsado: ATipo de evento: keyup

Propiedad keyCode: 65

Propiedad charCode: 0

Carácter pulsado: ?

En los eventos keydown y keyup, la propiedad keyCode sigue valiendo lo mismo en los dos casos. El motivo es que keyCode almacena el código interno de la tecla, por lo que si se pulsa la misma tecla, se obtiene el mismo código, independientemente de que una misma tecla puede producir caracteres diferentes (por ejemplo mayúsculas y minúsculas).

En el evento keypress, el valor de la propiedad charCode varía, ya que el carácter a, no es el mismo que el carácter A. En este caso, el valor de charCode coincide con el código ASCII del carácter pulsado.

Siguiendo en el navegador Firefox, si ahora se pulsa una tecla especial, como por ejemplo el tabulador, se muestra la siguiente información:

--------------------------------------

Tipo de evento: keydown

Propiedad keyCode: 9

Propiedad charCode: 0

## Carácter pulsado: ?Tipo de evento: keypress

Propiedad keyCode: 9

Propiedad charCode: 0

## Carácter pulsado: ?Tipo de evento: keyup

Propiedad keyCode: 9

Propiedad charCode: 0

Carácter pulsado: ?

Las teclas especiales no disponen de la propiedad charCode, ya que sólamente se guarda el código interno de la tecla pulsada en la propiedad keyCode, en este caso el código 9. Si se pulsa la tecla Enter, se obtiene el código 13, la tecla de la flecha superior produce el código 38, etc. No obstante, dependiendo del teclado utilizado para pulsar las teclas y dependiendo de la disposición de las teclas en función del idioma del teclado, estos códigos podrían variar.

El evento keypress es el único que permite obtener el *carácter realmente pulsado*, ya que al pulsar sobre la tecla 2 habiendo pulsado la tecla Shift previamente, se obtiene el carácter ", que es precisamente el que muestra el evento keypress.

El siguiente código de JavaScript permite obtener de forma correcta en cualquier navegador el carácter correspondiente a la tecla pulsada:

function manejador(elEvento) {

var evento = elEvento || window.event;

var caracter = evento.charCode || evento.keyCode;

alert("El carácter pulsado es: " + String.fromCharCode(caracter));

}

document.onkeypress = manejador;

### Información sobre los eventos de ratón

La información más relevante sobre los eventos relacionados con el ratón es la de las coordenadas de la posición del puntero del ratón. Aunque el origen de las coordenadas siempre se encuentra en la esquina superior izquierda, el punto que se toma como referencia de las coordenadas puede variar.

De esta forma, es posible obtener la posición del ratón respecto de la pantalla del ordenador, respecto de la ventana del navegador y respecto de la propia página HTML (que se utiliza cuando el usuario ha hecho scroll sobre la página). Las coordenadas más sencillas son las que se refieren a la posición del puntero respecto de la ventana del navegador, que se obtienen mediante las propiedades clientX y clientY:

function muestraInformacion(elEvento) {

var evento = elEvento || window.event;

var coordenadaX = evento.clientX;

var coordenadaY = evento.clientY;

alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " + coordenadaY);

}

document.onclick = muestraInformacion;

Las coordenadas de la posición del puntero del ratón respecto de la pantalla completa del ordenador del usuario se obtienen de la misma forma, mediante las propiedades screenX y screenY:

var coordenadaX = evento.screenX;

var coordenadaY = evento.screenY;

En muchas ocasiones, es necesario obtener otro par de coordenadas diferentes: las que corresponden a la posición del ratón respecto del origen de la página. Estas coordenadas no siempre coinciden con las coordenadas respecto del origen de la ventana del navegador, ya que el usuario puede hacer scroll sobre la página web. Internet Explorer no proporciona estas coordenadas de forma directa, mientras que el resto de navegadores sí que lo hacen. De esta forma, es necesario detectar si el navegador es de tipo Internet Explorer y en caso afirmativo realizar un cálculo sencillo:

// Detectar si el navegador es Internet Explorer

var ie = navigator.userAgent.toLowerCase().indexOf('msie')!=-1;

if(ie) {

coordenadaX = evento.clientX + document.body.scrollLeft;

coordenadaY = evento.clientY + document.body.scrollTop;

}

else {

coordenadaX = evento.pageX;

coordenadaY = evento.pageY;

}

alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " + coordenadaY + " respecto de la página web");

La variable ie vale true si el navegador en el que se ejecuta el script es de tipo Internet Explorer (cualquier versión) y vale false en otro caso. Para el resto de navegadores, las coordenadas respecto del origen de la página se obtienen mediante las propiedades pageX y pageY. En el caso de Internet Explorer, se obtienen sumando la posición respecto de la ventana del navegador (clientX, clientY) y el desplazamiento que ha sufrido la página (document.body.scrollLeft, document.body.scrollTop).

# Propiedades básicas de formularios y elementos

JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios. En primer lugar, cuando se carga una página web, el navegador crea automáticamente un array llamado forms y que contiene la referencia a todos los formularios de la página.

Para acceder al array forms, se utiliza el objeto document, por lo que document.forms es el array que contiene todos los formularios de la página.

Además del array de formularios, el navegador crea automáticamente un array llamado elements por cada uno de los formularios de la página. Cada array elements contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario. Utilizando la sintaxis de los arrays, la siguiente instrucción obtiene el primer elemento del primer formulario de la página:

document.forms[0].elements[0];

La sintaxis de los arrays no siempre es tan concisa. El siguiente ejemplo muestra cómo obtener directamente el último elemento del primer formulario de la página:

document.forms[0].elements[document.forms[0].elements.length-1];

Aunque esta forma de acceder a los formularios es rápida y sencilla, tiene un inconveniente muy grave. ¿Qué sucede si cambia el diseño de la página y en el código HTML se cambia el orden de los formularios originales o se añaden nuevos formularios? El problema es que *"el primer formulario de la página"* ahora podría ser otro formulario diferente al que espera la aplicación.

En un entorno tan cambiante como el diseño web, es muy difícil confiar en que el orden de los formularios se mantenga estable en una página web. Por este motivo, siempre debería evitarse el acceso a los formularios de una página mediante el array document.forms.

Una forma de evitar los problemas del método anterior consiste en acceder a los formularios de una página a través de su nombre (atributo name) o a través de su atributo id. El objeto document permite acceder directamente a cualquier formulario mediante su atributo name:

var formularioPrincipal = document.formulario;

var formularioSecundario = document.otro\_formulario;

<form name="formulario" >

...

</form>

<form name="otro\_formulario" >

...

</form>

Accediendo de esta forma a los formularios de la página, el script funciona correctamente aunque se reordenen los formularios o se añadan nuevos formularios a la página. Los elementos de los formularios también se pueden acceder directamente mediante su atributo name:

var formularioPrincipal = document.formulario;

var primerElemento = document.formulario.elemento;

<form name="formulario">

<input type="text" name="elemento" />

</form>

Obviamente, también se puede acceder a los formularios y a sus elementos utilizando las funciones DOM de acceso directo a los nodos. El siguiente ejemplo utiliza la habitual función document.getElementById() para acceder de forma directa a un formulario y a uno de sus elementos:

var formularioPrincipal = document.getElementById("formulario");

var primerElemento = document.getElementById("elemento");

<form name="formulario" id="formulario" >

<input type="text" name="elemento" id="elemento" />

</form>

Independientemente del método utilizado para obtener la referencia a un elemento de formulario, cada elemento dispone de las siguientes propiedades útiles para el desarrollo de las aplicaciones:

* type: indica el tipo de elemento que se trata. Para los elementos de tipo <input> (text, button, checkbox, etc.) coincide con el valor de su atributo type. Para las listas desplegables normales (elemento <select>) su valor es select-one, lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es select-multiple. Por último, en los elementos de tipo <textarea>, el valor de type es textarea.
* form: es una referencia directa al formulario al que pertenece el elemento. Así, para acceder al formulario de un elemento, se puede utilizar document.getElementById("id\_del\_elemento").form
* name: obtiene el valor del atributo name de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar.
* value: permite leer y modificar el valor del atributo value de XHTML. Para los campos de texto (<input type="text"> y <textarea>) obtiene el texto que ha escrito el usuario. Para los botones obtiene el texto que se muestra en el botón. Para los elementos *checkbox* y *radiobutton* no es muy útil, como se verá más adelante

Por último, los eventos más utilizados en el manejo de los formularios son los siguientes:

* onclick: evento que se produce cuando se pincha con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones que permite definir XHTML (<input type="button">, <input type="submit">, <input type="image">).
* onchange: evento que se produce cuando el usuario cambia el valor de un elemento de texto (<input type="text"> o <textarea>). También se produce cuando el usuario selecciona una opción en una lista desplegable (<select>). Sin embargo, el evento sólo se produce si después de realizar el cambio, el usuario *pasa* al siguiente campo del formulario, lo que técnicamente se conoce como que *"el otro campo de formulario ha perdido el foco"*.
* onfocus: evento que se produce cuando el usuario selecciona un elemento del formulario.
* onblur: evento complementario de onfocus, ya que se produce cuando el usuario ha *deseleccionado* un elemento por haber seleccionado otro elemento del formulario. Técnicamente, se dice que el elemento anterior *"ha perdido el foco"*.

# Utilidades básicas para formularios

### Obtener el valor de los campos de formulario

La mayoría de técnicas JavaScript relacionadas con los formularios requieren leer y/o modificar el valor de los campos del formulario. Por tanto, a continuación se muestra cómo obtener el valor de los campos de formulario más utilizados.

#### Cuadro de texto y textarea

El valor del texto mostrado por estos elementos se obtiene y se establece directamente mediante la propiedad value.

<input type="text" id="texto" />

var valor = document.getElementById("texto").value;

<textarea id="parrafo"></textarea>

var valor = document.getElementById("parrafo").value;

#### Radiobutton

Cuando se dispone de un grupo de radiobuttons, generalmente no se quiere obtener el valor del atributo value de alguno de ellos, sino que lo importante es conocer cuál de todos los radiobuttons se ha seleccionado. La propiedad checked devuelve true para el radiobutton seleccionado y false en cualquier otro caso. Si por ejemplo se dispone del siguiente grupo de radiobuttons:

<input type="radio" value="si" name="pregunta" id="pregunta\_si"/> SI

<input type="radio" value="no" name="pregunta" id="pregunta\_no"/> NO

<input type="radio" value="nsnc" name="pregunta" id="pregunta\_nsnc"/> NS/NC

El siguiente código permite determinar si cada radiobutton ha sido seleccionado o no:

var elementos = document.getElementsByName("pregunta");

for(var i=0; i<elementos.length; i++) {

alert(" Elemento: " + elementos[i].value + "\n Seleccionado: " + elementos[i].checked);

}

#### Checkbox

Los elementos de tipo checkbox son muy similares a los radiobutton, salvo que en este caso se debe comprobar cada checkbox de forma independiente del resto. El motivo es que los grupos de radiobutton son mutuamente excluyentes y sólo se puede seleccionar uno de ellos cada vez. Por su parte, los checkbox se pueden seleccionar de forma independiente respecto de los demás.

Si se dispone de los siguientes checkbox:

<input type="checkbox" value="condiciones" name="condiciones" id="condiciones"/> He leído y acepto las condiciones

<input type="checkbox" value="privacidad" name="privacidad" id="privacidad"/> He leído la política de privacidad

Utilizando la propiedad checked, es posible comprobar si cada checkbox ha sido seleccionado:

var elemento = document.getElementById("condiciones");

alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);

elemento = document.getElementById("privacidad");

alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);

#### Select

Las listas desplegables (<select>) son los elementos en los que es más difícil obtener su valor. Si se dispone de una lista desplegable como la siguiente:

<select id="opciones" name="opciones">

<option value="1">Primer valor</option>

<option value="2">Segundo valor</option>

<option value="3">Tercer valor</option>

<option value="4">Cuarto valor</option>

</select>

En general, lo que se requiere es obtener el valor del atributo value de la opción (<option>) seleccionada por el usuario. Obtener este valor no es sencillo, ya que se deben realizar una serie de pasos. Además, para obtener el valor seleccionado, deben utilizarse las siguientes propiedades:

* options, es un array creado automáticamente por el navegador para cada lista desplegable y que contiene la referencia a todas las opciones de esa lista. De esta forma, la primera opción de una lista se puede obtener mediante document.getElementById("id\_de\_la\_lista").options[0].
* selectedIndex, cuando el usuario selecciona una opción, el navegador actualiza automáticamente el valor de esta propiedad, que guarda el índice de la opción seleccionada. El índice hace referencia al array options creado automáticamente por el navegador para cada lista.

// Obtener la referencia a la lista

var lista = document.getElementById("opciones");

// Obtener el índice de la opción que se ha seleccionado

var indiceSeleccionado = lista.selectedIndex;

// Con el índice y el array "options", obtener la opción seleccionada

var opcionSeleccionada = lista.options[indiceSeleccionado];

// Obtener el valor y el texto de la opción seleccionada

var textoSeleccionado = opcionSeleccionada.text;

var valorSeleccionado = opcionSeleccionada.value;

alert("Opción seleccionada: " + textoSeleccionado + "\n Valor de la opción: " + valorSeleccionado);

Como se ha visto, para obtener el valor del atributo value correspondiente a la opción seleccionada por el usuario, es necesario realizar varios pasos. No obstante, normalmente se abrevian todos los pasos necesarios en una única instrucción:

var lista = document.getElementById("opciones");

// Obtener el valor de la opción seleccionada

var valorSeleccionado = lista.options[lista.selectedIndex].value;

// Obtener el texto que muestra la opción seleccionada

var valorSeleccionado = lista.options[lista.selectedIndex].text;

Lo más importante es no confundir el valor de la propiedad selectedIndex con el valor correspondiente a la propiedad value de la opción seleccionada. En el ejemplo anterior, la primera opción tiene un value igual a 1. Sin embargo, si se selecciona esta opción, el valor de selectedIndex será 0, ya que es la primera opción del array options (y los arrays empiezan a contar los elementos en el número 0).

### Establecer el foco en un elemento

En programación, cuando un elemento está seleccionado y se puede escribir directamente en el o se puede modificar alguna de sus propiedades, se dice que tiene el foco del programa.

Si un cuadro de texto de un formulario tiene el foco, el usuario puede escribir directamente en el sin necesidad de pinchar previamente con el ratón en el interior del cuadro. Igualmente, si una lista desplegable tiene el foco, el usuario puede seleccionar una opción directamente subiendo y bajando con las flechas del teclado.

Al pulsar repetidamente la tecla TABULADOR sobre una página web, los diferentes elementos (enlaces, imágenes, campos de formulario, etc.) van obteniendo el foco del navegador (el elemento seleccionado cada vez suele mostrar un pequeño borde punteado).

Para asignar el foco a un elemento de XHTML, se utiliza la función focus(). El siguiente ejemplo asigna el foco a un elemento de formulario cuyo atributo id es igual a primero:

document.getElementById("primero").focus();

<form id="formulario" action="#">

<input type="text" id="primero" />

</form>

Ampliando el ejemplo anterior, se puede asignar automáticamente el foco del programa al primer elemento del primer formulario de la página, independientemente del id del formulario y de los elementos:

if(document.forms.length > 0) {

if(document.forms[0].elements.length > 0) {

document.forms[0].elements[0].focus();

}

}

El código anterior comprueba que existe al menos un formulario en la página mediante el tamaño del array forms. Si su tamaño es mayor que 0, se utiliza este primer formulario. Empleando la misma técnica, se comprueba que el formulario tenga al menos un elemento (if(document.forms[0].elements.length > 0)). En caso afirmativo, se establece el foco del navegador en el primer elemento del primer formulario (document.forms[0].elements[0].focus();).

Para que el ejemplo anterior sea completamente correcto, se debe añadir una comprobación adicional. El campo de formulario que se selecciona no debería ser de tipo hidden:

if(document.forms.length > 0) {

for(var i=0; i < document.forms[0].elements.length; i++) {

var campo = document.forms[0].elements[i];

if(campo.type != "hidden") {

campo.focus();

break;

}

}

}

### Evitar el envío duplicado de un formulario

Uno de los problemas habituales con el uso de formularios web es la posibilidad de que el usuario pulse dos veces seguidas sobre el botón "Enviar". Si la conexión del usuario es demasiado lenta o la respuesta del servidor se hace esperar, el formulario original sigue mostrándose en el navegador y por ese motivo, el usuario tiene la tentación de volver a pinchar sobre el botón de "Enviar".

En la mayoría de los casos, el problema no es grave e incluso es posible controlarlo en el servidor, pero puede complicarse en formularios de aplicaciones importantes como las que implican transacciones económicas.

Por este motivo, una buena práctica en el diseño de aplicaciones web suele ser la de deshabilitar el botón de envío después de la primera pulsación. El siguiente ejemplo muestra el código necesario:

<form id="formulario" action="#">

...

<input type="button" value="Enviar" onclick="this.disabled=true; this.value=’Enviando...’; this.form.submit()" />

</form>

Cuando se pulsa sobre el botón de envío del formulario, se produce el evento onclick sobre el botón y por tanto, se ejecutan las instrucciones JavaScript contenidas en el atributo onclick:

1. En primer lugar, se deshabilita el botón mediante la instrucción this.disabled = true;. Esta es la única instrucción necesaria si sólo se quiere deshabilitar un botón.
2. A continuación, se cambia el mensaje que muestra el botón. Del original "Enviar" se pasa al más adecuado "Enviando..."
3. Por último, se envía el formulario mediante la función submit() en la siguiente instrucción: this.form.submit()

El botón del ejemplo anterior está definido mediante un botón de tipo <input type="button" />, ya que el código JavaScript mostrado no funciona correctamente con un botón de tipo <input type="submit" />. Si se utiliza un botón de tipo submit, el botón se deshabilita antes de enviar el formulario y por tanto el formulario acaba sin enviarse.

### Limitar el tamaño de caracteres de un textarea

La carencia más importante de los campos de formulario de tipo textarea es la imposibilidad de limitar el máximo número de caracteres que se pueden introducir, de forma similar al atributo maxlength de los cuadros de texto normales.

JavaScript permite añadir esta característica de forma muy sencilla. En primer lugar, hay que recordar que con algunos eventos (como onkeypress, onclick y onsubmit) se puede evitar su comportamiento normal si se devuelve el valor false.

Evitar el comportamiento normal equivale a modificar completamente el comportamiento habitual del evento. Si por ejemplo se devuelve el valor false en el evento onkeypress, la tecla pulsada por el usuario no se tiene en cuenta. Si se devuelve false en el evento onclick de un elemento como un enlace, el navegador no carga la página indicada por el enlace.

Si un evento devuelve el valor true, su comportamiento es el habitual:

<textarea onkeypress="return true;"></textarea>

En el textarea del ejemplo anterior, el usuario puede escribir cualquier carácter, ya que el evento onkeypress devuelve true y por tanto, su comportamiento es el normal y la tecla pulsada se transforma en un carácter dentro del textarea.

Sin embargo, en el siguiente ejemplo:

<textarea onkeypress="return false;"></textarea>

Como el valor devuelto por el evento onkeypress es igual a false, el navegador no ejecuta el comportamiento por defecto del evento, es decir, la tecla presionada no se transforma en ningún carácter dentro del textarea. No importa las veces que se pulsen las teclas y no importa la tecla pulsada, ese textarea no permitirá escribir ningún carácter.

Aprovechando esta característica, es sencillo limitar el número de caracteres que se pueden escribir en un elemento de tipo textarea: se comprueba si se ha llegado al máximo número de caracteres permitido y en caso afirmativo se evita el comportamiento habitual del evento y por tanto, los caracteres adicionales no se añaden al textarea:

function limita(maximoCaracteres) {

var elemento = document.getElementById("texto");

if(elemento.value.length >= maximoCaracteres ) {

return false;

}

else {

return true;

}

}

<textarea id="texto" onkeypress="return limita(100);"></textarea>

En el ejemplo anterior, con cada tecla pulsada se compara el número total de caracteres del textarea con el máximo número de caracteres permitido. Si el número de caracteres es igual o mayor que el límite, se devuelve el valor false y por tanto, se evita el comportamiento por defecto de onkeypress y la tecla no se añade.

### Restringir los caracteres permitidos en un cuadro de texto

En ocasiones, puede ser útil bloquear algunos caracteres determinados en un cuadro de texto. Si por ejemplo un cuadro de texto espera que se introduzca un número, puede ser interesante no permitir al usuario introducir ningún carácter que no sea numérico.

Igualmente, en algunos casos puede ser útil impedir que el usuario introduzca números en un cuadro de texto. Utilizando el evento onkeypress y unas cuantas sentencias JavaScript, el problema se resuelve fácilmente:

function permite(elEvento, permitidos) {

// Variables que definen los caracteres permitidos

var numeros = "0123456789";

var caracteres = " abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ";

var numeros\_caracteres = numeros + caracteres;

var teclas\_especiales = [8, 37, 39, 46];

// 8 = BackSpace, 46 = Supr, 37 = flecha izquierda, 39 = flecha derecha

// Seleccionar los caracteres a partir del parámetro de la función

switch(permitidos) {

case 'num':

permitidos = numeros;

break;

case 'car':

permitidos = caracteres;

break;

case 'num\_car':

permitidos = numeros\_caracteres;

break;

}

// Obtener la tecla pulsada

var evento = elEvento || window.event;

var codigoCaracter = evento.charCode || evento.keyCode;

var caracter = String.fromCharCode(codigoCaracter);

// Comprobar si la tecla pulsada es alguna de las teclas especiales

// (teclas de borrado y flechas horizontales)

var tecla\_especial = false;

for(var i in teclas\_especiales) {

if(codigoCaracter == teclas\_especiales[i]) {

tecla\_especial = true;

break;

}

}

// Comprobar si la tecla pulsada se encuentra en los caracteres permitidos

// o si es una tecla especial

return permitidos.indexOf(caracter) != -1 || tecla\_especial;

}

// Sólo números

<input type="text" id="texto" onkeypress="return permite(event, 'num')" />

// Sólo letras

<input type="text" id="texto" onkeypress="return permite(event, 'car')" />

// Sólo letras o números

<input type="text" id="texto" onkeypress="return permite(event, 'num\_car')" />

El funcionamiento del script anterior se basa en permitir o impedir el comportamiento habitual del evento onkeypress. Cuando se pulsa una tecla, se comprueba si el carácter de esa tecla se encuentra dentro de los caracteres permitidos para ese elemento <input>.

# Validación

La principal utilidad de JavaScript en el manejo de los formularios es la validación de los datos introducidos por los usuarios. Antes de enviar un formulario al servidor, se recomienda validar mediante JavaScript los datos insertados por el usuario. De esta forma, si el usuario ha cometido algún error al rellenar el formulario, se le puede notificar de forma instantánea, sin necesidad de esperar la respuesta del servidor.

Notificar los errores de forma inmediata mediante JavaScript mejora la satisfacción del usuario con la aplicación (lo que técnicamente se conoce como "mejorar la experiencia de usuario") y ayuda a reducir la carga de procesamiento en el servidor.

Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario. En esta función, se comprueban si los valores que ha introducido el usuario cumplen las restricciones impuestas por la aplicación.

Aunque existen tantas posibles comprobaciones como elementos de formulario diferentes, algunas comprobaciones son muy habituales: que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, que la dirección de email indicada sea correcta, que la fecha introducida sea lógica, que se haya introducido un número donde así se requiere, etc.

A continuación se muestra el código JavaScript básico necesario para incorporar la validación a un formulario:

<form action="" method="" id="" name="" onsubmit="return validacion()">

...

</form>

Y el esquema de la función validacion() es el siguiente:

function validacion() {

if (condicion que debe cumplir el primer campo del formulario) {

// Si no se cumple la condicion...

alert('[ERROR] El campo debe tener un valor de...');

return false;

}

else if (condicion que debe cumplir el segundo campo del formulario) {

// Si no se cumple la condicion...

alert('[ERROR] El campo debe tener un valor de...');

return false;

}

...

else if (condicion que debe cumplir el último campo del formulario) {

// Si no se cumple la condicion...

alert('[ERROR] El campo debe tener un valor de...');

return false;

}

// Si el script ha llegado a este punto, todas las condiciones

// se han cumplido, por lo que se devuelve el valor true

return true;

}

El funcionamiento de esta técnica de validación se basa en el comportamiento del evento onsubmit de JavaScript. Al igual que otros eventos como onclick y onkeypress, el evento 'onsubmit varía su comportamiento en función del valor que se devuelve.

Así, si el evento onsubmit devuelve el valor true, el formulario se envía como lo haría normalmente. Sin embargo, si el evento onsubmit devuelve el valor false, el formulario no se envía. La clave de esta técnica consiste en comprobar todos y cada uno de los elementos del formulario. En cuando se encuentra un elemento incorrecto, se devuelve el valor false. Si no se encuentra ningún error, se devuelve el valor true.

Por lo tanto, en primer lugar se define el evento onsubmit del formulario como:

onsubmit="return validacion()"

### Validar un campo de texto obligatorio

Se trata de forzar al usuario a introducir un valor en un cuadro de texto o textarea en los que sea obligatorio. La condición en JavaScript se puede indicar como:

valor = document.getElementById("campo").value;

if( valor == null || valor.length == 0 || /^\s+$/.test(valor) ) {

return false;

}

Para que se de por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducido sea mayor que cero y que no se hayan introducido sólo espacios en blanco.

### Validar un campo de texto con valores numéricos

Se trata de obligar al usuario a introducir un valor numérico en un cuadro de texto. La condición JavaScript consiste en:

valor = document.getElementById("campo").value;

if( isNaN(valor) ) {

return false;

}

Si el contenido de la variable valor no es un número válido, no se cumple la condición. La ventaja de utilizar la función interna isNaN() es que simplifica las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.

### Validar que se ha seleccionado una opción de una lista

Se trata de obligar al usuario a seleccionar un elemento de una lista desplegable. El siguiente código JavaScript permite conseguirlo:

indice = document.getElementById("opciones").selectedIndex;

if( indice == null || indice == 0 ) {

return false;

}

<select id="opciones" name="opciones">

<option value="">- Selecciona un valor -</option>

<option value="1">Primer valor</option>

<option value="2">Segundo valor</option>

<option value="3">Tercer valor</option>

</select>

### Validar una dirección de email

Se trata de obligar al usuario a introducir una dirección de email con un formato válido. Por tanto, lo que se comprueba es que la dirección parezca válida, ya que no se comprueba si se trata de una cuenta de correo electrónico real y operativa. La condición JavaScript consiste en:

valor = document.getElementById("campo").value;

if( !(/\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)/.test(valor)) ) {

return false;

}

### Validar una fecha

Las fechas suelen ser los campos de formulario más complicados de validar por la multitud de formas diferentes en las que se pueden introducir. El siguiente código asume que de alguna forma se ha obtenido el año, el mes y el día introducidos por el usuario:

var ano = document.getElementById("ano").value;

var mes = document.getElementById("mes").value;

var dia = document.getElementById("dia").value;

valor = new Date(ano, mes, dia);

if( !isNaN(valor) ) {

return false;

}

La función Date(ano, mes, dia) es una función interna de JavaScript que permite construir fechas a partir del año, el mes y el día de la fecha. Es muy importante tener en cuenta que el número de mes se indica de 0 a 11, siendo 0 el mes de Enero y 11 el mes de Diciembre. Los días del mes siguen una numeración diferente, ya que el mínimo permitido es 1 y el máximo 31.

### Validar un número de DNI

Se trata de comprobar que el número proporcionado por el usuario se corresponde con un número válido de Documento Nacional de Identidad o DNI. Aunque para cada país o región los requisitos del documento de identidad de las personas pueden variar, a continuación se muestra un ejemplo genérico fácilmente adaptable. La validación no sólo debe comprobar que el número esté formado por ocho cifras y una letra, sino que también es necesario comprobar que la letra indicada es correcta para el número introducido:

valor = document.getElementById("campo").value;

var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];

if( !(/^\d{8}[A-Z]$/.test(valor)) ) {

return false;

}

if(valor.charAt(8) != letras[(valor.substring(0, 8))%23]) {

return false;

}

La primera comprobación asegura que el formato del número introducido es el correcto, es decir, que está formado por 8 números seguidos y una letra. Si la letra está al principio de los números, la comprobación sería /^[A-Z]\d{8}$/. Si en vez de ocho números y una letra, se requieren diez números y dos letras, la comprobación sería /^\d{10}[A-Z]{2}$/ y así sucesivamente.

La segunda comprobación aplica el algoritmo de cálculo de la letra del DNI y la compara con la letra proporcionada por el usuario. El algoritmo de cada documento de identificación es diferente, por lo que esta parte de la validación se debe adaptar convenientemente.

### Validar un número de teléfono

Los números de teléfono pueden ser indicados de formas muy diferentes: con prefijo nacional, con prefijo internacional, agrupado por pares, separando los números con guiones, etc.

El siguiente script considera que un número de teléfono está formado por nueve dígitos consecutivos y sin espacios ni guiones entre las cifras:

valor = document.getElementById("campo").value;

if( !(/^\d{9}$/.test(valor)) ) {

return false;

}

### Validar que un checkbox ha sido seleccionado

Si un elemento de tipo checkbox se debe seleccionar de forma obligatoria, JavaScript permite comprobarlo de forma muy sencilla:

elemento = document.getElementById("campo");

if( !elemento.checked ) {

return false;

}

Si se trata de comprobar que todos los checkbox del formulario han sido seleccionados, es más fácil utilizar un bucle:

formulario = document.getElementById("formulario");

for(var i=0; i<formulario.elements.length; i++) {

var elemento = formulario.elements[i];

if(elemento.type == "checkbox") {

if(!elemento.checked) {

return false;

}

}

}

### Validar que un radiobutton ha sido seleccionado

Aunque se trata de un caso similar al de los checkbox, la validación de los radiobutton presenta una diferencia importante: en general, la comprobación que se realiza es que el usuario haya seleccionado algún radiobutton de los que forman un determinado grupo. Mediante JavaScript, es sencillo determinar si se ha seleccionado algún radiobutton de un grupo:

opciones = document.getElementsByName("opciones");

var seleccionado = false;

for(var i=0; i<opciones.length; i++) {

if(opciones[i].checked) {

seleccionado = true;

break;

}

}

if(!seleccionado) {

return false;

}