

JavaScript

CONTENIDO:

Propiedades básicas	2
Campos del formulario	4
Asignando Foco	6
Utilidades con formularios	7



Unidad 8 - Formularios

La programación de aplicaciones que contienen formularios web siempre ha sido una de las tareas fundamentales de JavaScript. De hecho, una de las principales razones por las que se inventó el lenguaje de programación JavaScript fue la necesidad de validar los datos de los formularios directamente en el navegador del usuario. De esta forma, se evitaba recargar la página cuando el usuario cometía errores al rellenar los formularios.

No obstante, la aparición de las aplicaciones AJAX ha relevado al tratamiento de formularios como la principal actividad de JavaScript. Ahora, el principal uso de JavaScript es el de las comunicaciones asíncronas con los servidores y el de la manipulación dinámica de las aplicaciones. De todas formas, el manejo de los formularios sigue siendo un requerimiento imprescindible para cualquier programador de JavaScript.

1.— Propiedades básicas

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. Como se trata de un array, el acceso a cada formulario se realiza con la misma sintaxis de los arrays. La siguiente instrucción accede al primer formulario de la página:

```
document.forms[0];
```

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

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

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".

2.— Campos del 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.

1. 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;
```

2. 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="nsrc" name="pregunta" id="pregunta_nsrc"/> 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);
}
```

3. 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);
```

4. 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).

3.— Asignando Foco

En programación, cuando un elemento está seleccionado y se puede escribir directamente en él 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 él 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).

Si en una página web el formulario es el elemento más importante, como por ejemplo en una página de búsqueda o en una página con un formulario para registrarse, se considera una buena práctica de usabilidad el asignar automáticamente el foco al primer elemento del formulario cuando se carga la página.

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 pri-

mer 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;
        }
    }
}
```

4.— Utilidades con formularios

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:

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.

A continuación, se cambia el mensaje que muestra el botón. Del original "Enviar" se pasa al más adecuado "Enviando..."

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 = " abcdefghijklmñopqrstuvwxyzABCDEFGHIJ-
KLMNÑOPQRSTUVWXYZ";
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 fun-
ción
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 espe-
ciales
// (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 || te-
cla_especial;
}

// Sólo números
<input type="text" id="texto" onkeypress="return permi-
te(event, 'num')" />

// Sólo letras
<input type="text" id="texto" onkeypress="return permi-
te(event, 'car')" />

// Sólo letras o números
<input type="text" id="texto" onkeypress="return permi-
te(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>`.

Si el carácter se encuentra dentro de los caracteres permitidos, se devuelve `true` y por tanto el comportamiento de `onkeypress` es el habitual y la tecla se escribe. Si el carácter no se encuentra dentro de los caracteres permitidos, se devuelve `false` y por tanto se impide el comportamiento normal de `onkeypress` y la tecla no llega a escribirse en el input.

Además, el script anterior siempre permite la pulsación de algunas teclas especiales. En concreto, las teclas `BackSpace` y `Supr` para borrar caracteres y las teclas `Flecha Izquierda` y `Flecha Derecha` para moverse en el cuadro de

texto siempre se pueden pulsar independientemente del tipo de caracteres permitidos.

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()"
```

Como el código JavaScript devuelve el valor resultante de la función `validacion()`, el formulario solamente se enviará al servidor si esa función devuelve `true`. En el caso de que la función `validacion()` devuelva `false`, el formulario permanecerá sin enviarse.

Dentro de la función `validacion()` se comprueban todas las condiciones impuestas por la aplicación. Cuando no se cumple una condición, se devuelve `false` y por tanto el formulario no se envía. Si se llega al final de la función, todas las condiciones se han cumplido correctamente, por lo que se devuelve `true` y el formulario se envía.

La notificación de los errores cometidos depende del diseño de cada aplicación. En el código del ejemplo anterior simplemente se muestran mensajes mediante la función `alert()` indicando el error producido. Las aplicaciones web mejor diseñadas muestran cada mensaje de error al lado del elemento de formulario correspondiente y también suelen mostrar un mensaje principal indicando que el formulario contiene errores.

Una vez definido el esquema de la función `validacion()`, se debe añadir a esta función el código correspondiente a todas las comprobaciones que se realizan sobre los elementos del formulario. A continuación, se muestran algunas de las validaciones más habituales de los campos de formulario.

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.

La palabra reservada `null` es un valor especial que se utiliza para indicar "ningún valor". Si el valor de una variable es `null`, la variable no contiene ningún valor de tipo objeto, array, numérico, cadena de texto o booleano.

La segunda parte de la condición obliga a que el texto introducido tenga una longitud superior a cero caracteres, esto es, que no sea un texto vacío.

Por último, la tercera parte de la condición (`/\s+$/ .test(valor)`) obliga a que el valor introducido por el usuario no sólo esté formado por espacios en blanco. Esta comprobación se basa en el uso de "expresiones regulares", un recurso habitual en cualquier lenguaje de programación pero que por su gran complejidad no se van a estudiar. Por lo tanto, sólo es necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión.



Este material forma parte del currículo del módulo **Implantación de Aplicaciones WEB** correspondiente al ciclo formativo de grado superior “**Administración de sistemas informáticos en red**”.

La información contenida en el magazine ha sido extraída de la **Web** y, en parte, de la experiencia profesional acumulada durante mis años de docencia.

Por tanto, es de uso público para tareas de docencia y aprendizaje.