# Validación de formularios

Otra de las principales funcionalidades de JavaScript es poder validar el contenido de un formulario antes de enviarlo al servidor, ahorrándole a éste una buena parte del trabajo de procesamiento de datos. De hecho, esta fue una de las razones por las que se inventó JavaScript, aunque hoy en día esta tarea también puede hacerse total o parcialmente (según el formulario) utilizando HTML5. Veremos en este documento cómo podemos combinar ambas tecnologías (JavaScript y HTML5) para la validación de formularios en el lado del cliente.

## 1. Validación tradicional de formularios

Antes de la aparición de HTML5, JavaScript era la única herramienta disponible en el lado del cliente para poder validar formularios. Veremos en este apartado algunos elementos de JavaScript que podemos tener en cuenta a la hora de realizar esta validación, y en el siguiente apartado veremos cómo podemos combinarlos con la validación nativa de HTML5.

### 1.1. Acceso a los datos del formulario

Ya hemos visto que para poder acceder al contenido de un documento HTML hemos de utilizar las funciones de acceso al DOM. Podemos asignar un *id* diferente a cada elemento del formulario y acceder a él con `document.getElementById(...)`, pero también podemos emplear el atributo `name` de cada campo del formulario (y del propio formulario) para acceder a ellos más directamente. Por ejemplo, para acceder al siguiente formulario:

```
<form name="miFormulario" action="pagina.php" method="post">
...
</form>
```

Usaríamos el código `document.miFormulario` (una vez el formulario ya esté cargado). Si dentro del formulario tenemos, por ejemplo, un campo de texto llamado email:

```
<form name="miFormulario" action="pagina.php" method="post">
...
    <input type="email" id="email" name="email" size="20">
...
</form>
```

accederíamos a él con el código `document.miFormulario.email`. Es decir, usamos el atributo `name` tanto del formulario al que queremos acceder como del campo concreto que queremos consultar. También podríamos utilizar el método `getElementById` visto antes para acceder a elementos del contenido, siempre que les pongamos un id. En el caso del campo email anterior, accederíamos directamente con `document.getElementById("email")`.

## 1.2. Consulta y comprobación de valores

Una vez hemos accedido al formulario o a alguno de sus campos (normalmente se accede a los campos), todos ellos tienen una serie de propiedades básicas cuyos valores podemos consultar o modificar (según el caso):

- `type`: indica el tipo de elemento que se trata. Para los elementos de tipo input, coincidirá con su atributo *type*. Para listas, será *select-one* (selección simple) o *select-multiple* (selección múltiple), y para las áreas de texto será *textarea*.
- `name`: indica el nombre del elemento (su atributo *name*)
- `value`: indica el valor que tiene actualmente el elemento. Esta propiedad puede tanto leerse como modificarse, por lo que desde JavaScript podemos dar valor a los campos de un formulario.

El siguiente ejemplo comprueba si el campo email anterior está vacío, y si es así, le asigna automáticamente un e-mail por defecto *admin@empresa.com*:

```
let mail = document.miFormulario.email;
if (mail.value == null || mail.value == "")
{
    mail.value="admin@empresa.com";
}
```

**El caso de los botones de radio y los *checkbox***

En el caso de estos dos campos, además de las propiedades anteriores tenemos una propiedad más llamada `checked`, que indica si el botón o el cuadro está marcado o no. El siguiente ejemplo comprueba si una casilla de *checkbox* llamada *casado* está marcada, mostrando un mensaje en caso afirmativo:

```
let casado = document.miFormulario.casado;
if (casado.checked)
{
    alert("Has indicado que estás casado");
}
```

El caso concreto de los botones de radio es algo más complejo, ya que todos comparten el mismo name. Veamos este formulario:

```
<form name="miFormulario" action="pagina.php" method="post">
    <input type="radio" name="idioma" value="ESP" />Español<br />
    <input type="radio" name="idioma" value="ENG" />Inglés<br />
    <input type="radio" name="idioma" value="FRA" />Francés<br />
    ...
</form>
```

Si quisiéramos acceder a `document.miFormulario.idioma`, accederíamos a los tres botones de radio a la vez. Tendríamos que utilizar una estructura *for* o similar para recorrer los botones y ver cuál está marcado. Aquí tenemos un ejemplo de eso:

```
let listaBotones = document.miFormulario.idioma;
for (let indice in listaBotones)
{
    if (listaBotones[indice].checked)
    {
        alert ("Has seleccionado " + listaBotones[indice].value);
    }
}
```

La propiedad *value* de cada botón será igual al atributo *value* de dicho botón (ESP, ENG o FRA, respectivamente, para el ejemplo anterior). También podemos, de forma abreviada, acceder al *value* del control, y obtendremos el valor actualmente seleccionado:

```
let idiomaSeleccionado = document.miFormulario.idioma.value;
```

**El caso de las listas**

En el caso de una lista (*select*), tenemos las propiedades:

- `selectedIndex`, que nos da la posición del elemento actualmente seleccionado
- `options`, que es una lista con todas las opciones, que podemos recorrer con una estructura *for* o similar

El siguiente ejemplo obtiene el elemento seleccionado de una lista con *id lista1*, y nos muestra con un mensaje su valor y el texto que hay visible en la lista (recordemos que cada *option* de la lista puede tener un *value*, y dentro de la etiqueta, su propio texto diferente del *value*):

```
let lista = document.getElementById("lista1");
let indiceSeleccionado = lista.selectedIndex;
let elementoSeleccionado = lista.options[indiceSeleccionado];
alert ("Has seleccionado " + elementoSeleccionado.text);
alert ("Su valor es " + elementoSeleccionado.value);
```

Nuevamente, también podemos acceder al *value* de la lista y obtendremos el valor actualmente seleccionado:

```
let valorSeleccionado = lista.value;
```

**Ejercicio 1:**

Crea un formulario en una página llamada **form1_js.html**. Este formulario tendrá por nombre *form1*, y dentro tendrá los siguientes campos:

- Un campo de texto llamado *login*, con una etiqueta asociada que ponga "Login".
- Un campo numérico llamado *nhijos*, con una etiqueta asociada que ponga "Número de hijos"
- Una casilla checkbox llamada *info* con un texto que ponga "Deseo recibir información por e-mail"
- Tres botones de radio que llamados *horario* con los textos *Mañana*, *Tarde* y *Noche* respectivamente, y encima de ellos una etiqueta asociada que ponga "Horario preferido para llamadas"
- Una lista desplegable llamada *operador*, y dentro los valores *Movil1*, *Taronje*, *Hablafone* y *Roigo*. Encima debe tener una etiqueta que diga "Seleccione operador"
- Un botón (tipo *button*) de *Enviar* que, al hacer clic sobre él, active una función llamada *mostrarInformacion* que saque con mensajes (*alert*) cada una de las opciones que hayamos escrito en cada campo del formulario. Por ejemplo, si hemos puesto como login "pepe", como número de hijos 2, hemos marcado la casilla de información, elegido turno de Tarde y el operador Taronje, deberá sacar, uno a uno, los siguientes mensajes:
  - Tu login es pepe
  - Tienes 2 hijos
  - Deseas recibir información
  - Has elegido el horario de Tarde
  - Has elegido el operador Taronje
- En el caso de que el número de hijos esté vacío, deberá rellenarlo con el valor 0 al pulsar el botón de Enviar

## 1.3. Algunas opciones de validación de campos

Vamos a ver algunas de las opciones más habituales para validar campos de formularios, aunque la mayoría de ellas ya las podemos hacer directamente utilizando HTML5.

**Comprobación de campo obligatorio vacío**

Para comprobar que un campo obligatorio está vacío, tenemos el atributo *required* en HTML5. Para validarlo desde JavaScript, debemos acceder a dicho campo (por su id o su nombre, como queramos), y comprobar si su valor es igual a "":

```
let login = document.miFormulario.login;
if (login.value == "")
{
    alert ("El campo login no puede estar vacío");
}
```

**Comprobación de tipo de dato adecuado**

Para comprobar que en un campo hemos introducido un tipo de dato adecuado, en muchas ocasiones podemos emplear el atributo *type* específico en la etiqueta *input*. Por ejemplo, con `type="number"` nos aseguramos de que el campo tendrá un valor numérico, y también podemos establecer el rango de valores permitido con los atributos *min* y *max*. Pero, además de lo anterior, existen algunas funciones auxiliares en JavaScript para hacer estas comprobaciones. Lo más habitual es comprobar que un campo numérico (por ejemplo, una edad, o un año), tenga realmente un número y no otra cosa. Para ello usaremos la función `isNaN` para comprobar si ese dato no es numérico. Así, por ejemplo, si tenemos un campo edad y queremos ver si el dato introducido en él es numérico, tendríamos algo como:

```
let edad = document.miFormulario.edad;
if (isNaN(edad.value))
{
    alert ("El dato introducido en el campo edad no es un número");
}
```

**Comprobación de selección en lista**

Ya hemos visto que con la propiedad *selectedIndex* obtenemos la posición del elemento seleccionado en una lista. Si el usuario no ha seleccionado ninguna opción, o tiene seleccionada la primera de todas (que normalmente es un valor por defecto que hay que cambiar), esta propiedad vale 0 o null (según el tipo de lista), así que basta comprobar estos dos valores para ver si se ha elegido algo de la lista:

```
let lista = document.miFormulario.lista;
if (lista.selectedIndex == null || lista.selectedIndex == 0)
{
    alert ("Debes seleccionar algún elemento de la lista");
}
```

**Comprobación de patrón. Expresiones regulares**

Para comprobar que un determinado campo cumple un determinado patrón (por ejemplo, que tenga cuatro dígitos y una letra), se utilizan expresiones regulares, similares a las que existen para HTML5 (mediante el atributo *pattern*). En la siguiente tabla tenemos un resumen de los tipos de símbolos que podemos utilizar en las expresiones regulares:

| Símbolo | Significado |
|---|---|
| `^` | Se utiliza para indicar el comienzo de un texto o línea |
| `$` | Se utiliza para indicar el final de un texto o línea |
| `*` | Indica que el carácter anterior puede aparecer 0 o más veces |
| `+` | Indica que el carácter anterior puede aparecer 1 o más veces |
| `?` | Indica que el carácter anterior puede aparecer 0 o 1 vez |
| `.` | Representa a cualquier carácter, salvo el salto de línea |
| `x\|y` | Indica que puede haber un elemento x o y |
| `{n}` | Indica que el carácter anterior debe aparecer n veces |
| `{m, n}` | Indica que el carácter anterior debe aparecer entre m y n veces |
| `[abc]` | Cualquiera de los caracteres entre corchetes. Podemos especificar rangos con un guión, como por ejemplo `[A-L]` |
| `[^abc]` | Cualquier carácter que no esté entre los corchetes |
| `\b` | Un límite de palabra (espacio, salto de línea...) |
| `\B` | Cualquier cosa que no sea un límite de palabra |
| `\d` | Un dígito (equivaldría al intervalo `[0-9]`) |
| `\D` | Cualquier cosa que no sea un dígito (equivaldría a `[^0-9]`) |
| `\n` | Salto de línea |
| `\s` | Cualquier carácter separador (espacio, tabulación, salto de página o de línea) |
| `\S` | Cualquier carácter que no sea separador |
| `\t` | Tabulación |
| `\w` | Cualquier alfanumérico incluyendo subrayado (igual a `[A-Za-z0-9_]`) |
| `\W` | Cualquier no alfanumérico ni subrayado (`[^A-Za-z0-9_]`) |

Algunos símbolos especiales (como el punto, o el +), se representan literalmente anteponiéndoles la barra invertida. Por ejemplo, si queremos indicar el carácter del punto, se pondría `\.` Veamos algunos ejemplos:

| Ejemplo | Significado |
|---|---|
| `^\d{9,11}$` | Entre 9 y 11 dígitos |
| `(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{8,10})$` | Password seguro (8-10 caracteres, con al menos un dígito y una letra) |
| `^\d{1,2}\/\d{1,2}\/\d{2,4}$` | Fecha (d/m/a) |
| `^[\w-\.]{3,}@([\w-]{2,}\.)*([\w-]{2,}\.)[\w-]{2,4}$` | E-mail |

El siguiente ejemplo comprueba si en un campo llamado *dni* tenemos un DNI válido, formado por 8 dígitos y una letra mayúscula:

```
let dni = document.miFormulario.dni.value;
if (/^\d{8}[A-Z]$/.test(dni) == false)
{
    alert ("El DNI es incorrecto");
}
```

Observa que ponemos primero la expresión (entre dos barras `/`), seguida del método `test` y la variable que queremos comprobar si encaja en ese patrón.

## 1.4. Cómo validar todo el formulario

¿Cómo aunar todo lo que hemos visto y validar las opciones del formulario que queramos antes de enviarlo? Primero debemos tener en cuenta que el evento que se dispara cuando intentamos enviar el formulario es el evento `submit`. Por tanto, debemos añadir un *event handler* sobre el formulario, llamando a la función que queramos usar para validar. En la función podemos hacer uso del método `preventDefault`, por ejemplo, para evitar que el formulario se envíe si hay algún error.

```
document.miFormulario.addEventListener("submit", validar);

...
function validar(evento)
{
    let formularioCorrecto = true;

    // Código para comprobar cada campo
    // Pondríamos "formularioCorrecto" a false si alguno falla

    if (!formularioCorrecto)
        evento.preventDefault();
}
```

**Ejercicio 2:**

Crea un formulario en un documento llamado **form2_js.html**. El formulario deberá llamarse *form2*, y deberás definir una función llamada *validarFormulario* que comentaremos a continuación. El formulario debe tener los siguientes campos:

- Un campo llamado *login*, con una etiqueta asociada "Login"
- Un campo llamado *edad* (numérico), con una etiqueta asociada "Edad"
- Un campo llamado *nacimiento* (numérico) con una etiqueta "Año de nacimiento"
- Una lista desplegable llamada *provincia* con una etiqueta "Seleccione provincia". La primera opción de esa lista debe decir "-Seleccionar una provincia-", y tener *value*="". Las otras tres opciones deben ser Alicante, Valencia y Castellón.
- Un botón de tipo submit y con valor *Enviar*.

Al intentar enviar el formulario, se activará la función *validarFormulario()* comentada antes, que deberemos implementar para comprobar lo siguiente:

- El campo *login* no puede estar vacío
- El campo *edad* puede estar vacío, pero si no lo está, debe tener un número entre 0 y 200
- El campo nacimiento debe tener 4 dígitos
- La lista desplegable debe tener un elemento seleccionado que no sea el primero.
- Si todo lo anterior se cumple, se deberá mostrar un mensaje "Todo es correcto" y enviar el formulario (el envío fallará porque la página destino no la habremos implementado). Si algún campo no es válido, se deberá mostrar el correspondiente mensaje de error y evitar el envío.

# 2. Validación de formularios combinada con HTML5

Con la aparición de HTML5, se dotó a este lenguaje de marcado de la posibilidad de validar la información introducida en los campos de los formularios, a través de atributos como:

- **required**: determina que un campo es obligatorio
- **pattern**: obliga a que los datos de un campo tengan un patrón determinado
- **min**, **max**: determina los límites inferior superior en campos con valores numéricos
- Además, algunos tipos de campos (e-mail, URL...) tienen una auto-validación para comprobar que la información que hemos introducido se puede corresponder con lo que ese campo espera (un e-mail, una dirección web, etc, según el caso).

Podéis consultar aquí más información sobre cómo aplicar estos atributos en los campos de un formulario. En cuanto al atributo *pattern*, la sintaxis que podemos seguir para definir las expresiones regulares es la misma que la que hemos explicado en este documento unos párrafos atrás.

Sin embargo, la validación que ofrece HTML5 se queda un poco corta en dos aspectos fundamentalmente:

- Tiene sus limitaciones a la hora de validar cierta información algo más específica o compleja. Por ejemplo, comprobar que dos campos tengan el mismo valor, como sería el caso de una confirmación de password en un formulario.

- Ofrece unos mensajes de error predefinidos ante los errores de validación (no configurables)

Afortunadamente, podemos incorporar nuestro código JavaScript en nuestras webs y combinarlo con la validación de HTML5 para suplir esas carencias, de forma que todo quede integrado en un único mecanismo de validación que aúna las ventajas de ambos (HTML5 y JavaScript).

## 2.1. Validaciones personalizadas

Como comentábamos antes, una de las deficiencias más importantes que ofrece la validación de formularios con HTML5 es que es algo limitada. Si, por ejemplo, queremos comprobar que el usuario ha puesto el mismo password en dos campos distintos para asegurarnos de que no se ha equivocado al escribirlo, no disponemos de ningún atributo específico para esto. Sin embargo, gracias a JavaScript, y en concreto al método `setCustomValidity` podemos controlar casos como éste, e incluso personalizar el mensaje que se mostrará al usuario si se produce un error de validación.

Este método recibe como parámetro el mensaje de error que mostraremos en el campo afectado. De este modo, cuando intentemos enviar el formulario, el envío se detendrá si alguno de los campos tiene algún mensaje de error pendiente. Para anular el mensaje y volver a decir que todo está correcto, llamaremos de nuevo al método pasándole una cadena vacía.

El siguiente ejemplo comprueba si dos campos de password *pass1* y *pass2* son iguales. Si no lo son, establece el mensaje "Los passwords no coinciden" en el segundo campo. Si todo es correcto, vuelve a dejar el mensaje vacío.

Así sería el formulario del ejemplo:

```
<form action="pagina.php" name="form1" id="form1">
    <label>
        Login:
        <input type="text" name="login" id="login" required>
    </label><br>
    <label>
        Password:
        <input type="password" name="pass1" id="pass1" required>
    </label><br>
    <label>
        Repetir password:
        <input type="password" name="pass2" id="pass2" required>
    </label><br>
    <input type="submit" value="Enviar" id="enviar">
</form>
```

Y así definiríamos el código JavaScript para gestionar la validación de los passwords:

```
let pass1 = document.getElementById("pass1");
let pass2 = document.getElementById("pass2");
pass1.addEventListener("input", validarPass);
pass2.addEventListener("input", validarPass);

function validarPass()
{
    if (pass1.value != pass2.value)
        pass2.setCustomValidity("Los passwords no coinciden");
    else
        pass2.setCustomValidity("");
}
```

Notar que, si cargamos el formulario y dejamos vacío el login o los passwords, se sigue disparando el error de que es obligatorio. Dicho de otro modo, la función de validación nos ayuda a completar las validaciones que hayamos indicado con HTML5, sin que éstas se vean anuladas.

## 2.2. Personalizar mensajes de error

A la hora de modificar el mensaje de error por defecto de un campo que se valida automáticamente, también podemos hacer uso del método `setCustomValidity`. En este caso, podemos hacer uso del evento *invalid* que se genera cuando el campo no es válido, y tendremos que asegurarnos de vaciar el mensaje de error después (por ejemplo, en cuanto se vuelva a escribir algo en el campo):

```
let login = document.getElementById("login");
login.addEventListener("invalid", function() {
    this.setCustomValidity("El login es obligatorio");
});
login.addEventListener("input", function() {
    this.setCustomValidity("");
});
```

En algunos casos nos puede interesar mostrar distintos mensajes de error dependiendo del tipo de fallo. Por ejemplo un campo e-mail obligatorio puede fallar porque esté vacío, o porque el valor que hayamos introducido no se corresponda con un e-mail válido. Los campos del formulario disponen, a través de la propiedad `validity`, de una serie de subpropiedades que podemos consultar para comprobar qué ha pasado exactamente. Algunas de estas subpropiedades son:

- `valid` es *true* cuando el campo actual es válido
- `valueMissing` es *true* cuando es un campo obligatorio vacío
- `typeMismatch` es *true* cuando el tipo de información no se corresponde con lo que espera el campo (por ejemplo, e-mails no válidos)

- `patternMismatch` es *true* cuando la entrada no se corresponde con lo especificado en un atributo *pattern*
- ...

El siguiente ejemplo muestra un mensaje u otro en un campo e-mail con id *email*, dependiendo del error producido:

```
let email = document.getElementById("email");
email.addEventListener("invalid", function() {
    if (email.validity.valueMissing)
        email.setCustomValidity("El e-mail no puede estar vacío");
    else if (email.validity.typeMismatch)
        email.setCustomValidity("El formato de email no es correcto");
});
email.addEventListener("input", function() {
    email.setCustomValidity("");
});
```

## 2.3. CSS y validación de formularios

Existen algunos selectores (más concretamente, pseudo-clases) en CSS que podemos emplear para definir ciertos estilos relacionados con la validación de formularios.

- La pseudo-clase `:required` se aplicará a los campos de formularios que sean requeridos. Por ejemplo, de este modo podemos ponerlos con un fondo amarillo:

```
input:required
{
    background-color: yellow;
}
```

- Las pseudo-clases `:valid` e `:invalid` se aplican sobre campos cuyo valor sea válido o inválido, respectivamente.

```
input:valid
{
    background-color: green;
}
input:invalid
{
    background-color: red;
}
```

Notar que en ocasiones estas pseudo-clases pueden solaparse. Si un campo es obligatorio e inválido e intentamos cambiar la misma propiedad desde las dos reglas (por ejemplo el color de fondo, como en el caso anterior), sólo prevalecerá una de ellas (la que se aplique más tarde). En este sentido, conviene que los estilos de campo obligatorio no se solapen con los de validez, y afecten a otras propiedades (color del borde, por ejemplo).

**Ejercicio 3:**

Crea un formulario en un documento llamado **form3_js.html**. El formulario deberá llamarse *form3*, y contendrá estos campos con sus respectivas restricciones:

- Un campo llamado *login*, obligatorio
- Dos campos de tipo *password* para introducir el password. Serán obligatorios, y además los dos passwords deben coincidir, y estar formados por al menos 8 caracteres, compuestos por letras (minúsculas o mayúsculas) y dígitos.
- Un campo llamado *email*, obligatorio
- Un botón de tipo submit y con valor *Enviar*.

Se debe comprobar la validez de los campos, y mostrar mensajes de error en formato HTML5 (sin *alerts*). Deberás configurar mensajes de error personalizados en el caso del password (para mostrar si no coinciden, o para indicar el patrón que debe tener). También se deben marcar con fondo rojo los controles que hayan provocado error al intentar enviar el formulario. Dicho color de fondo rojo se restablecerá a blanco en cuanto se vuelva a escribir algo sobre el control.