1 Validaciones de datos en páginas web

1.1 INTRODUCCIÓN A LOS FORMULARIOS

Un formulario es un conjunto de controles que permiten al usuario interactuar con el documento o página. El objetivo de esta interacción suele ser, a menudo, la solicitud de información adicional o un mero intercambio de datos a petición del usuario.

Entre los diferentes controles que se pueden insertar o agregar a los formularios podemos encontrar acciones directas vinculadas a botones, solicitud de entradas de texto de una única línea, solicitud de entradas de texto multilínea, casillas de verificación, botones de única elección o tipo radio, selección de objetos y/o ficheros, entre otros.

En este capítulo vamos a ver los tipos básicos de formulario, cómo definirlos, cómo enviarlos, cómo hacerlos receptivos y, gran medida, cómo hacerlos usables y accesibles.

1.1.1 TIPOS DE FORMULARIO

Existen, fundamentalmente, cinco tipos de formulario que están definidos en función de su objetivo. Formularios de contacto, acceso, registro, suscripción y de entrada general.

No obstante, sea cual sea el tipo de formulario y su objetivo, para que cumpla con las expectativas de los clientes y con los requisitos establecidos por el RGPD (Reglamento General de Protección de Datos), deben incluir:

- Una casilla de verificación explícita con la que, los usuarios, puedan aceptar la política de privacidad del sitio. Por defecto, no puede estar seleccionada.
- Los enlaces pertinentes con la política de privacidad, el aviso legal, política de cookies, límite y responsabilidad, etcétera.
- Un texto, no demasiado largo, ni demasiado escueto, sobre qué datos se van a almacenar, quién será el responsable y cuál es el objetivo de dicho almacenamiento.
- Algún método para que el usuario ejerza su derecho a la eliminación de datos.
- Algún algoritmo de cifrado para que la información se gestione y manipule de forma encriptada. Esto es, que los datos que se envíen y/o almacenen estén codificados para evitar usos fraudulentos.

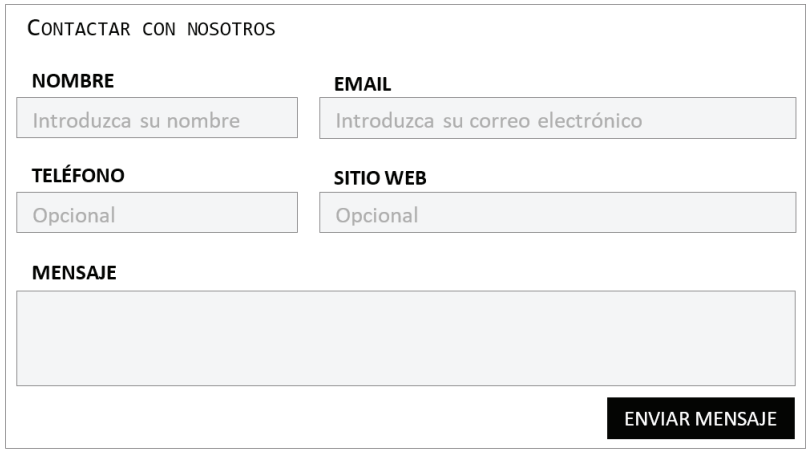
Únicamente se deben solicitar los campos que sean necesarios y estén bien justificados, es decir, no se debe solicitar una información concreta, como pueda ser tus hobbies, a no ser que se tenga una buena razón y esté justificada. Además, se debe tratar de conseguir que el medio sea lo suficientemente seguro y confiable, como para que invite a introducir los datos solicitados.

1.1.1.1 Formularios de contacto

Los formularios de contacto pueden llegar a ser un elemento clave en un sitio web porque permiten, o hacen posible, que la comunicación entre los usuarios y los proveedores sea directa y privada.

En general, un formulario de contacto debe solicitar sólo la información esencial para el proceso de comunicación. Esto es, no se deben pedir datos como la razón social, sitio web o teléfono sólo por fines comerciales o estadísticos.

Por ejemplo, el siguiente formulario podría ser considerado no fiable, además de ser ilegal.



Como se puede apreciar en la ilustración anterior, el sitio web no es un dato necesario y, aunque sea un dato opcional, puede suscitar desconfianza e impedir que el usuario haga uso de él. Además, como se ha mencionado anteriormente, es ilegal puesto que no presenta el checkbox de Política de Privacidad, entre otras cosas.

A continuación, se muestra el mismo ejemplo, pero corregido.

1.1.1.2 Formularios de suscripción

Los formularios de suscripción son, esencialmente, lo mismo que los formularios de contacto, pero más sencillos. Su objetivo es recuperar correos electrónicos para luego utilizarlos con fines comerciales o informativos.

En general, un formulario de suscripción sólo requiere de una caja de texto para recuperar el email, no obstante, es habitual pedir también el nombre para dirigirse a él. Por ejemplo, la siguiente ilustración podría una buena opción como formulario de suscripción.

1.1.1.3 Formularios de acceso

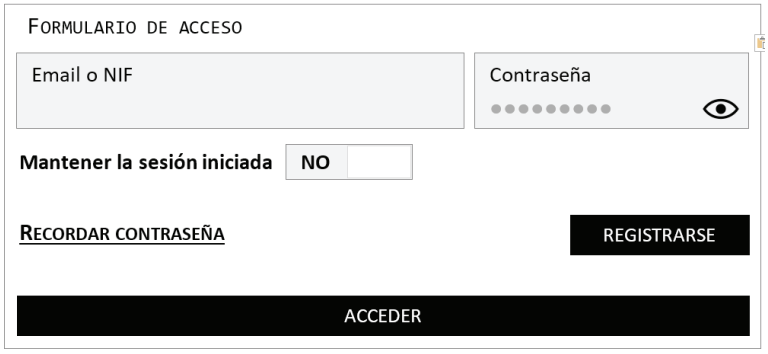
Los formularios de acceso son, junto con los formularios de contacto, los más recurrentes y utilizados en el mundo web. Su objetivo, como su propio nombre indica, es proporcionar acceso a información, productos o servicios que no están disponibles por vía pública o sin proporcionar una identidad.

En general, un formulario de acceso se caracteriza por tener dos cajas de texto para el nombre de usuario y contraseña, una casilla de verificación o botón de tipo interruptor para mantener la sesión iniciada, un enlace para recuperar la contraseña en supuesto caso de olvido y, evidentemente, un botón para acceder.

Los nombres de usuario se presentan con el texto visible y, a menudo, son el correo electrónico de los usuarios, aunque puede valer cualquier tipo de identificador único como un DNI, código de cliente, nombre de usuario, etcétera.

Las contraseñas suelen presentarse con el texto oculto, con asteriscos o puntos y, a menudo, suelen proporcionar un modo de cambiar a modo visible para ver el texto escrito.

A continuación, se muestra un ejemplo:



Si observamos la ilustración anterior, veremos que el campo contraseña tiene un icono de ojo que funciona como botón para cambiar el modo de presentación de la misma, Si pulsamos una vez se podrán en modo visible y seremos capaces de leer lo escrito, pero si pulsamos otra vez, volverá a ocultarse, mostrándose como la vemos ahora mismo.

Además, se ha puesto un botón de acceso al formulario de registro. Esta funcionalidad también es habitual, sobre todo, porque un registro también suele ser un acceso, una vez que ha finalizado el proceso.

1.1.1.4 Formularios de registro

Los formularios de registro suelen ir de la mano con los de acceso. Si bien, no siempre se ofrece la posibilidad de registrarse online, si es lo más habitual.

En general, un formulario de registro se caracteriza por la presencia de campos propietarios como el nombre o email del usuario, pero también pueden contener elementos considerados información sensible como son la edad, género o país de nacimiento. La decisión de qué campos se deben solicitar al usuario debe estar regida por la lógica de negocio y el RGPD (Reglamento General de Protección de Datos). No obstante, recordemos que, cuantos menos campos se solicite, más cómodo y confiable podrá sentirse el usuario.

Los nombres de usuario para dicho registro son, a menudo, el correo electrónico de los usuarios, aunque puede valer cualquier tipo de identificador único como un DNI o código de cliente.

Las contraseñas suelen seguir un patrón estricto de definición para evitar la fácil recuperación a través de métodos como la fuerza bruta y uso fraudulento.

En general, podríamos decir que un buen patrón de contraseña es aquel que tiene, al menos, un carácter en mayúsculas, un carácter en minúsculas, un dígito o número, un carácter especial como pueda ser el símbolo de almohadilla o interrogación y con una longitud mínima de ocho caracteres.

Sobra decir que no se deben utilizar sustituciones de carácter a número como pueda ser sustituir la letra A por el número cuatro, ni fechas especiales como cumpleaños o aniversarios y, por supuesto, nada de nombres propios. Eso sí, hay que tratar de que sea fácil de recordar y difícil de adivinar.

A continuación, se muestra un ejemplo:

Cabe destacar que, es bueno que los campos de un formulario de registro se soliciten de forma que el usuario vaya tomando confianza de menos a más. Es decir, se debe tratar de conseguir que los usuarios confíen un poco más

cada vez que se avanza en el proceso de registro. Esto es así porque puede pasar que un usuario no se sienta del todo cómodo si se le solicita primero la edad antes que el nombre de usuario y contraseña.

Por último, sólo destacar una cosa más. Si el formulario de registro requiere de muchos campos, es mejor que sean agrupados por contexto y solicitados en varios pasos o pantallas.

1.1.1.5 Formularios de entrada general

Los formularios de entrada general son aquellos formularios que tienen el propósito de recuperar una información concreta y que, por lo general, están pensados para cubrir las necesidades que no están cubiertas en las casuísticas anteriores.

Por ser algo más explícito, un formulario de propósito general podría ser un formulario de inserción de comentarios, de eventos o de inscripción a sorteos, concursos o juegos.

1.1.2 ELEMENTOS DISPONIBLES EN HTML 5

1.1.2.1 Elemento form

El elemento FORM especifica que el contenido que se va a representar es un formulario, es decir, una estructura de interacción, formada por uno o varios elementos, que permiten introducir datos y enviarlos al servidor para su procesamiento.

Los formularios pueden albergar muy diversos elementos con diferentes formatos y estructuras, pero los más comunes quizás sean INPUT, TEXTAREA, BUTTON, SELECT, OPTION, OPTGROUP, FIELDSET, LABEL y OUTPUT, lo cuales se pasarán a ver a continuación.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
accept-charset	Especifica la codificación de caracteres que se debe usar cuando se realice el envío del formulario. Su valor predeterminado es UNKNOWN, que indica que la codificación de caracteres a utilizar debe ser la misma que la del documento actual. Entre sus valores más frecuentes podemos encontrar la codificación ISO-8859-1 y la codificación UTF-8.
action	Especifica el destino dónde enviar los datos, es decir, la dirección a la que se redirigirá cuando se vaya a realizar la acción de enviar los datos de formulario.
autocomplete	Especifica si los elementos de formulario deben ser manipulados y presentados por agente de usuario de manera automática o no. Sus valores son ON y OFF.
enctype	Especifica cómo se deben codificar los datos a enviar. Sus posibles valores son: • APPLICATION/X-WWW-FORM-URLENCODED: para indicar que se codifiquen todos los caracteres antes de enviarlos. Es la opción por defecto. • MULTIPART/FORM-DATA: para indicar que no se codifiquen los caracteres. • TEXT-PLAIN: para indicar que sólo los espacios sean codificados como símbolos de suma.

method	Especifica el método de envío por el que se deben enviar los datos. Los posibles valores son GET y POST. La diferencia entre uno y otro es que: • Mientras que el método GET envía los datos del formulario como parámetros de la propia dirección o URL, el método POST los agrega como parte del cuerpo de la solicitud HTTP. • Mientras que el método GET permite una longitud máxima de 3000 caracteres, el método POST no posee limitaciones de tamaño. • Mientras que el método GET permite la adición en la barra de direcciones o favoritos, el método POST no. **NOTA** No se debe utilizar el método GET cuando se envían datos confidenciales o sensibles, sin embargo, puede ser la mejor opción cuando se trabaja con datos que no requieren de seguridad alguna, como es el caso de las QUERIES STRINGS o cadenas de consulta de Google.
name	Especifica o asigna el nombre del elemento. El atributo NAME puede ser utilizado para recuperar el valor de los campos del formulario desde el lado del servidor y puede resultar muy útil cuando se desean manipular los elementos desde JavaScript o CSS. Si el valor de este atributo lleva asignado como sufijo [] (los símbolos de corchetes), el identificador de nombre se definirá y actuará como si de un array numérico se tratase, es decir, se podrán referenciar los distintos elementos a través de un índice, siendo, el índice CERO el primero y, N-1, el último.
novalidate	Especifica si se deben validar los elementos del formulario o no. Cabe destacar que este atributo tendrá efecto con sólo declararlo, es decir, en cuanto esté presente, e independientemente de su valor, desactivará la validación de todos los elementos del formulario.
target	Específica a qué contexto o ventana se enviará la información. Entre los posibles valores que puede tomar, los más frecuentes son: • _BLANK: para indicar que se abra en una nueva pestaña o contexto, • _SELF: para indicar que se abra en la misma pestaña o contexto, • _PARENT: para indicar que se abra en la pestaña o contexto padre, • _TOP: para que se abra en el primer elemento BODY de la ventana. • [NOMBRE]: para que se abra en el marco identificado con ese nombre.

Ejemplo:

```
<form action="/action_page.php" method="get" name="frm">
    <label for="name">Nombre Completo:</label>
    <input type="text" id="name" name="name">

    <label>
        Nombre de usuario
        <input type="text" id="username" name="username">
    </label>

    <label for="password">Contraseña:</label>
    <input type="password" id="password" name="password">

    <label for="email">Email de contacto:</label>
```

```
    <input type="email" id="email" name="email">

    <label for="phone">Teléfono:</label>
    <input type="tel" id="phone" name="phone">

    <button type="submit">
        Guardar datos
    </button>
</form>
```

1.1.2.2 Elemento button

El elemento BUTTON especifica que el contenido que se va a representar es una acción. Como veremos, a diferencia del elemento INPUT, el elemento BUTTON puede albergar una gran variedad de contenidos como, por ejemplo, una imagen, un texto o, incluso, otros elementos HTML.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
autofocus	Especifica si se debe tomar el foco automáticamente después de ser renderizado y/o proceso de carga del documento. Sus valores pueden ser TRUE o FALSE.
disabled	Especifica si el elemento está deshabilitado o no. Esta propiedad tendrá efecto con sólo declararla, es decir, en cuanto esté presente e independientemente de su valor, el elemento aparecerá deshabilitado. No obstante, suele declararse con el valor DISABLED.
form	Especifica el formulario al que pertenece el elemento cuando se declaran sus elementos fuera del ámbito del formulario. Con respecto a sus posibles valores, el valor de este atributo debe ser el mismo que el indicado por el atributo ID del elemento FORM, de lo contrario, no se hará efectivo.
formaction	Especifica el destino dónde enviar los datos, es decir, la dirección a la que se redirigirá cuando se vaya a realizar la acción de enviar los datos de formulario. También es importante destacar que, cuando este atributo está presente, el atributo ACTION del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee que el formulario que se está definiendo puede ejecutar varias acciones que llevan a objetivos distintos.
formenctype	Especifica cómo se deben codificar los datos a enviar. También es importante destacar que, cuando este atributo está presente, el atributo ENCTYPE del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee enviar un mismo formulario con diferentes tipos de codificación. Sus posibles valores son los mismos que los definidos en el atributo ENCTYPE del elemento FORM.
formmethod	Especifica el método de envío por el que se deben enviar los datos. También es importante destacar que, cuando este atributo está presente, el atributo METHOD del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee enviar un mismo formulario con diferentes métodos. Sus posibles valores son los mismos que los definidos en el atributo METHOD del elemento FORM.

formnovalidate	Especifica si se deben validar los elementos del formulario o no. También es importante destacar que, cuando este atributo está presente, el atributo NOVALIDATE del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee validar el formulario en función de si se ejecuta una u otra acción. Cabe destacar que este atributo tendrá efecto con sólo declararlo, es decir, en cuanto esté presente, e independientemente de su valor, desactivará la validación de todos los elementos del formulario.
formtarget	Específica a qué contexto o ventana se enviará la información. También es importante destacar que, cuando este atributo está presente, el atributo TARGET del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee enviar un mismo formulario a diferentes ventanas o contextos. Sus posibles valores son los mismos que los definidos en el atributo TARGET del elemento FORM.
name	Especifica o asigna el nombre del elemento. El atributo NAME puede ser utilizado para recuperar el valor de los campos del formulario desde el lado del servidor y puede resultar muy útil cuando se desean manipular los elementos desde JavaScript o CSS. Si el valor de este atributo lleva asignado como sufijo [] (los símbolos de corchetes), el identificador de nombre se definirá y actuará como si de un array numérico se tratase, es decir, se podrán referenciar los distintos elementos a través de un índice, siendo, el índice CERO el primero y, N-1, el último.
type	Especifica el tipo de botón. Sus posibles valores son: • SUBMIT: para indicar que se envíen los datos del formulario, • BUTTON: para que permita hacer clic, pero no envíe los datos del formulario, • RESET: para indicar que se restablezcan todos los elementos del formulario a sus valores por defecto o preestablecidos.
value	Especifica el valor textual del elemento, por lo que no puede contener nada que no sea texto.

Ejemplo:

```
<button type=”submit”
        formaction=”./pages/login-test-2.php”
        formmethod=”POST”
        formenctype=”multipart/form-data”
        formtarget=”_blank”>
        value=”Probar test 2”
</button>
```

1.1.2.3 Elemento datalist

El elemento DATALIST especifica que el contenido que se va a representar es una lista de opciones predefinidas para un elemento INPUT.

La razón para utilizar este elemento es para proveer de una funcionalidad de autocompletado a los elementos INPUT. Los usuarios podrán ver las diferentes opciones como si de un desplegable se tratase, pero con la opción de ir buscando a través de coincidencias parciales proporcionadas mediante teclado.

Entre los atributos que admite en su configuración, cabe destacar que, el único atributo que necesita para funcionar, es el atributo ID. El valor de este atributo debe coincidir exactamente con el valor del atributo LIST declarado en el elemento INPUT al que está asociado.

Ejemplo:

```
<input list=”technologies”>

<datalist id=”technologies”>
  <option value=”HTML5”>
  <option value=”JavaScript”>
  <option value=”CSS3”>
  <option value=”SVG”>
</datalist>
```

1.1.2.4 Elemento fieldset

El elemento FIELDSET especifica que el contenido que se va a representar es una agrupación de elementos relacionados. En general, y salvo excepciones, todos los agentes de usuario dibujan un cuadro alrededor de este elemento, equivalente a un estilo de borde.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
disabled	Especifica si el elemento está deshabilitado o no. Esta propiedad tendrá efecto con sólo declararla, es decir, en cuanto esté presente e independientemente de su valor, el elemento aparecerá deshabilitado. No obstante, suele declararse con el valor DISABLED.
form	Especifica el formulario al que pertenece el elemento cuando se declaran sus elementos fuera del ámbito del formulario. Con respecto a sus posibles valores, el valor de este atributo debe ser el mismo que el indicado por el atributo ID del elemento FORM, de lo contrario, no se hará efectivo.
name	Especifica o asigna el nombre del elemento. El atributo NAME puede ser utilizado para recuperar el valor de los campos del formulario desde el lado del servidor y puede resultar muy útil cuando se desean manipular los elementos desde JavaScript o CSS. Si el valor de este atributo lleva asignado como sufijo [] (los símbolos de corchetes), el identificador de nombre se definirá y actuará como si de un array numérico se tratase, es decir, se podrán referenciar los distintos elementos a través de un índice, siendo, el índice CERO el primero y, N-1, el último.

Ejemplo:

```
<form action=”./login.php”>
    <fieldset>
        <legend>Acceso Privado</legend>

        <label for=”username”>Nombre de usuario</label>
        <input id=”username” name=”username” />
        <label for=”password”>Contraseña</label>
        <input id=”password” name=”password” />

        <button type=”submit”>Enviar</button>
    </fieldset>
</form>
```

1.1.2.5 Elemento input

El elemento INPUT especifica que el contenido que se va a representar es una entrada de datos.

Una de las cualidades más importantes que tiene el elemento INPUT es que tiene una gran variedad de validaciones nativas. Por ejemplo, es posible definir un elemento de entrada que sólo admita números, que sólo admita fechas en un formato específico o que valide las reglas de nombres de los correos electrónicos.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
accept	Especifica los tipos de archivo válidos cuando se utiliza un elemento INPUT de tipo FILE. Todos los posibles valores que puede tomar este atributo pueden encontrarse en IANA Media Types (*http://www.iana.org/assignments/media-types/*), aunque puede tomar un valor comodín como es AUDIO/*, VIDEO/* o IMAGE/*, que indican que se aceptan todas las extensiones de archivo de cada tipo.
alt	Permite especificar un texto alternativo para las imágenes que son definidas a través de un INPUT tipo IMAGE.
autocomplete	Especifica si los elementos de formulario deben ser manipulados y presentados por agente de usuario de manera automática o no. Sus valores son ON y OFF.
autofocus	Especifica si se debe tomar el foco automáticamente después de ser renderizado y/o proceso de carga del documento. Sus valores pueden ser TRUE o FALSE.
checked	Especifica si el elemento está chequeado o no. Cabe destacar que este atributo sólo es válido para los elementos INPUT de tipo RADIO y CHECKBOX y tendrá efecto con sólo declararla, es decir, en cuanto esté presente e independientemente de su valor, el elemento se marcará como chequeado o seleccionado.
disabled	Especifica si el elemento está deshabilitado o no. Esta propiedad tendrá efecto con sólo declararla, es decir, en cuanto esté presente e independientemente de su valor, el elemento aparecerá deshabilitado. No obstante, suele declararse con el valor DISABLED.
form	Especifica el formulario al que pertenece el elemento cuando se declaran sus elementos fuera del ámbito del formulario. Con respecto a sus posibles valores, el valor de este atributo debe ser el mismo que el indicado por el atributo ID del elemento FORM, de lo contrario, no se hará efectivo.
formaction	Especifica el destino dónde enviar los datos, es decir, la dirección a la que se redirigirá cuando se vaya a realizar la acción de enviar los datos de formulario. También es importante destacar que, cuando este atributo está presente, el atributo ACTION del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee que el formulario que se está definiendo pueda ejecutar varias acciones que llevan a objetivos distintos.
formenctype	Especifica cómo se deben codificar los datos a enviar. También es importante destacar que, cuando este atributo está presente, el atributo ENCTYPE del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee enviar un mismo formulario con diferentes tipos de codificación. Sus posibles valores son los mismos que los definidos en el atributo ENCTYPE del elemento FORM.

formmethod	Especifica el método de envío por el que se deben enviar los datos. También es importante destacar que, cuando este atributo está presente, el atributo METHOD del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee enviar un mismo formulario con diferentes métodos. Sus posibles valores son los mismos que los definidos en el atributo METHOD del elemento FORM.
formnovalidate	Especifica si se deben validar los elementos del formulario o no. También es importante destacar que, cuando este atributo está presente, el atributo NOVALIDATE del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee validar el formulario en función de si se ejecuta una u otra acción. Cabe destacar que este atributo tendrá efecto con sólo declararlo, es decir, en cuanto esté presente, e independientemente de su valor, desactivará la validación de todos los elementos del formulario.
formtarget	Específica a qué contexto o ventana se enviará la información. También es importante destacar que, cuando este atributo está presente, el atributo TARGET del elemento FORM es anulado. No obstante, será una buena herramienta cuando se desee enviar un mismo formulario a diferentes ventanas o contextos. Sus posibles valores son los mismos que los definidos en el atributo TARGET del elemento FORM.
height	Especifica el alto del elemento en píxeles.
list	Especifica la lista de opciones predefinidas que se debe asociar a un elemento INPUT. Cabe destacar que, para que un elemento INPUT acepte las opciones predefinidas del elemento DATALIST, el valor del atributo ID del DATALIST debe ser el mismo que el valor del atributo LIST del elemento INPUT.
max	Especifica el límite superior del rango de valores aceptado, es decir, el valor máximo que puede aceptar o se puede introducir en el elemento. Aunque este atributo es válido para los elementos INPUT de tipo METER y PROGRESS, no todos los agentes de usuario lo soportan. Internet Explorer 10 es uno de ellos.
maxlength	Especifica la longitud máxima, en caracteres, del valor que puede ser ingresado en el elemento.
min	Especifica el límite inferior del rango de valores aceptado, es decir, el valor mínimo que puede aceptar o se puede introducir en el elemento. Aunque este atributo es válido para los elementos INPUT de tipo METER y PROGRESS, no todos los agentes de usuario lo soportan. Internet Explorer 10 es uno de ellos.
multiple	Especifica que el elemento admite la selección o inserción de más de un valor. Cabe destacar que este atributo sólo es válido para los elementos INPUT de tipo FILE. Además, tendrá efecto con sólo declararlo, es decir, en cuanto esté presente e independientemente de su valor, el elemento permitirá la selección de múltiples valores.
name	Especifica o asigna el nombre del elemento. El atributo NAME puede ser utilizado para recuperar el valor de los campos del formulario desde el lado del servidor y puede resultar muy útil cuando se desean manipular los elementos desde JavaScript o CSS. Si el valor de este atributo lleva asignado como sufijo [] (los símbolos de corchetes), el identificador de nombre se definirá y actuará como si de un array numérico se tratase, es decir, se podrán referenciar los distintos elementos a través de un índice, siendo, el índice CERO el primero y, N-1, el último.
pattern	Especifica la expresión regular con la que se validará la entrada de datos en el elemento. La forma de trabajar con expresiones regulares es similar a la de JavaScript, no obstante, en la web de HTML5 Pattern, ubicada en la dirección *http://html5pattern.com/*, se pueden encontrar multitud de ejemplos aptos para ser utilizados. Cabe destacar que este atributo sólo es válido para los elementos INPUT de tipo TEXT, DATE, SEARCH, URL, TEL, EMAIL y PASSWORD.

placeholder	Especifica el texto que se debe mostrar cómo pista de datos válidos cuando el elemento no contenga un valor. Aunque algunos agentes de usuario pueden no aceptar este atributo, en general es una buena idea hacerlo para ayudar a la usabilidad y accesibilidad web. Si va asociado con otro atributo como PATTERN, la pista a proporcionar debe estar en concordancia con la validez de la entrada.
readonly	Especifica que el elemento es de sólo lectura, es decir, que no permite la inserción de nuevos valores ni la modificación de su valor actual. Cabe destacar que este atributo tendrá efecto con sólo declararlo, es decir, en cuanto esté presente, e independientemente de su valor, provocará que se ponga en modo de sólo lectura. No obstante, no quiere decir que no se pueda seleccionar, resaltar o copiar.
required	Especifica que el elemento debe contener valor, aunque este sea un espacio en blanco, es decir, que no puede ser nulo ni vacío.
size	Especifica la ubicación del recurso externo que se desea cargar o mostrar. Cabe destacar que este atributo sólo es válido para los elementos INPUT de tipo IMAGE.
src	Especifica la ubicación del recurso externo que se desea cargar o mostrar. Este atributo sólo es válido para los elementos INPUT de tipo IMAGE.
step	Especifica el intervalo de incremento o decremento sobre el valor actual del elemento, es decir, el número de pasos que se deben sumar o restar al valor actual del elemento. Cabe destacar que este atributo sólo es válido para los elementos INPUT de tipo NUMBER, RANGE, DATE, DATETIME, DATTETIME-LOCAL, MONTH, TIME, y WEEK.
type	Especifica el tipo de elemento INPUT. Sus posibles valores son: • BUTTON: Para indicar que es un botón. • CHECKBOX: Para indicar que es una casilla de verificación. • DATE: Para indicar que es una fecha. • DATETIME: Para indicar que es una fecha con hora. Sólo está soportado por Safari y Opera. • DATETIME-LOCAL: Para indicar que es una fecha con hora sin zona horaria. Sólo está soportado por Chrome, Microsoft Edge y Opera. • EMAIL: Para indicar que es un correo electrónico. • FILE: Para indicar que es un control para subir archivos al servidor. • HIDDEN: Para indicar que es un campo oculto. • IMAGE: Para indicar que es una imagen. • MONTH: Para indicar que es un mes. Sólo está soportado por Chrome, Microsoft Edge y Opera. • NUMBER: Para indicar que es un número real. • PASSWORD: Para indicar que es una contraseña. • RADIO: Para indicar que es un botón de radio. • RESET: Para indicar que es un botón de reinicialización que establece los valores por defecto o preestablecidos. • SEARCH: Para indicar que es un campo de búsqueda. • TEL: Para indicar que es un teléfono Actualmente sólo soportado por Safari. • TEXT: Para indicar que es una caja o campo de texto. • TIME: Para indicar que es una hora. Actualmente no soportado por Safari. • URL: Para indicar que es una dirección web o URL. • WEEK: Para indicar que es un valor de semana. Sólo está soportado por Chrome, Microsoft Edge y Opera.
value	Especifica el valor textual del elemento, por lo que no puede contener nada que no sea texto.
width	Especifica el ancho del elemento en píxeles.

Ejemplo:

```
<form action=”/action_page.php” method=”get” name=”frm”>
    <label for=”field1”>Campo de texto:</label>
    <input type=”text” id=”field1” name=”field1”>

    <label for=”field2”>Campo sólo números:</label>
    <input type=”number” id=”field2” name=”field2”>

    <label for=”field3”>Campo fecha:</label>
    <input type=”date” id=”field3” name=”field3”>

    <label for=”field4”>Campo URL:</label>
    <input type=”url” id=”field4” name=”field4”>

    <label for=”field5”>Campo para emails:</label>
    <input type=”email” id=”field5” name=”field5”>

    <label for=”field6”>Campo para teléfonos:</label>
    <input type=”tel” id=”field6” name=”field6”>

    <label for=”field7”>Campo para imágenes:</label>
    <input type=”image” id=”field7” name=”field7”>

    <label for=”field8”>Elemento para adjuntar archivos:</label>
    <input type=”file” id=”field8” name=”field8”>

    <label for=”field9”>Casilla de verificación:</label>
    <input type=”checkbox” id=”field9” name=”field9”>

    <label for=”field10”>Campo tipo radio:</label>
    <input type=”radio” id=”field10” name=”field10”>

    <label for=”field11”>Campo oculto:</label>
    <input type=”hidden” id=”field11” name=”field11”>

    <label for=”submit”>Botón enviar:</label>
    <input type=”submit” id=”submit” name=”submit”>

    <button type=”submit”>
        Guardar datos
    </button>
</form>
```

1.1.2.6 Elemento label

El elemento LABEL especifica que el contenido que se va a representar es una etiqueta para un elemento de formulario INPUT, METER, PROGRESS, SELECT o TEXTAREA.

Cuando se utiliza elemento LABEL, la estructura del documento se vuelve más legible y consistente. Además, beneficia tanto a la usabilidad web, como a la accesibilidad web porque los elementos pequeños como son las casillas de verificación pueden manipularse a través del elemento LABEL.

Además, al tener el control de formulario una etiqueta asociada, las herramientas de asistencia, como los lectores de pantalla, pueden leerla y comunicárselo a los usuarios con discapacidad visual total o parcial.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
for	Especifica el ID del elemento de formulario con el que está vinculada la etiqueta. El contenido al que se hace referencia a través de este atributo funcionará como texto descriptivo de la entrada de datos.
form	Especifica el formulario al que pertenece el elemento cuando se declaran sus elementos fuera del ámbito del formulario. Con respecto a sus posibles valores, el valor de este atributo debe ser el mismo que el indicado por el atributo ID del elemento FORM, de lo contrario, no se hará efectivo.

Ejemplo:

```
<form action=”/action_page.php” method=”get” name=”frm”>
    <label for=”name”>Nombre:</label>
    <input type=”text” id=”name” name=”name”>

    <!-- ... -->
</form>
```

1.1.2.7 Elemento legend

El elemento LEGEND especifica que el contenido que se va a representar es un título para un elemento FIELDSET.

Ejemplo:

```
<form action=”./login.php”>
    <fieldset>
        <legend>Acceso a ejemplo.com</legend>
        <!-- ... -->
    </fieldset>
</form>
```

Aunque el uso del elemento LEGEND no está recomendado fuera del elemento FIELDSET, se puede aplicar siempre que se desee. No obstante, a efectos de semántica web, sólo tendrá efecto cuando se encuentre dentro de un elemento FIELDSET.

1.1.2.8 Elemento meter

El elemento METER especifica que el contenido que se va a representar es una medición escalar, es decir, como una barra de progreso con rango conocido o valor fraccional.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
form	Especifica el formulario al que pertenece el elemento cuando se declaran sus elementos fuera del ámbito del formulario. Con respecto a sus posibles valores, el valor de este atributo debe ser el mismo que el indicado por el atributo ID del elemento FORM, de lo contrario, no se hará efectivo.
high	Especifica el valor a partir del cual se considera que es un valor alto, siendo este, igual o menor que el valor del atributo MAX.
low	Especifica el valor a partir del cual se considera que es un valor bajo, siendo este, igual o menor que el valor del atributo MIN.
max	Especifica el límite superior del rango de valores aceptado, es decir, el valor máximo que puede aceptar o se puede introducir en el elemento.
min	Especifica el límite inferior del rango de valores aceptado, es decir, el valor mínimo que puede aceptar o se puede introducir en el elemento.
optimum	Especifica el valor a partir del cual se considera que es un valor óptimo, siendo este, mayor que el valor del atributo MIN y menor que el atributo MAX.
value	Especifica el valor numérico inicial del elemento.

Ejemplo:

```
<form action=”/action_page.php” method=”get” name=”frm”>
    <label for=”available-space”>Espacio disponible en disco:</label>
    <meter id=”available-space” value=”324” min=”0” max=”1024”>
        324MB libres de 1024 MB
    </meter>

    <!-- ... -->
</form>
```

Cabe destacar que, aunque pueden parecerse, una medición escalar no es una barra de progreso. Por esta razón, el elemento METER no debe utilizarse para mostrar valores de progreso. El correcto uso de este elemento es para uso o capacidad de disco, para indicar el valor de tareas que tuvieron éxito en un conjunto definido o situaciones similares.

1.1.2.9 Elemento optgroup

El elemento OPTGROUP especifica que el contenido que se va a representar es un grupo de opciones relacionadas de un elemento SELECT.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
disabled	Especifica si el elemento está deshabilitado o no. Esta propiedad tendrá efecto con sólo declararla, es decir, en cuanto esté presente e independientemente de su valor, el elemento aparecerá deshabilitado. No obstante, suele declararse con el valor DISABLED.
label	Especifica la etiqueta que se mostrará para diferenciar el grupo de opciones relacionadas. Este atributo sólo es aplicable para el elemento OPTGROUP, descendiente directo del elemento SELECT.

Ejemplo:

```
<form action="/action_page.php" method="get" name="frm">
    <label for="motorcycles">Motos:</label>
    <select id="motorcycles">
        <optgroup label="Harley Davidson">
            <option value="01">Sportster</option>
            <option value="02">Softail</option>
            <option value="03">Touring</option>
        </optgroup>
        <optgroup label="Indian">
            <option value="04">Chief</option>
            <option value="05">Springfield</option>
            <option value="06">Scout Sixty</option>
        </optgroup>
    </select>

    <!-- ... -->
</form>
```

1.1.2.10 Elemento option

El elemento OPTION especifica que el contenido que se va a representar es una opción para elemento SELECT.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
disabled	Especifica si el elemento está deshabilitado o no. Esta propiedad tendrá efecto con sólo declararla, es decir, en cuanto esté presente e independientemente de su valor, el elemento aparecerá deshabilitado. No obstante, suele declararse con el valor DISABLED.
label	Especifica la etiqueta que se mostrará para identificar o etiquetar la opción. Cuando este atributo está presente, su valor será mostrado en vez de su contenido interno. No obstante, no todos los agentes de usuario proporcionan soporte nativo a este atributo.
selected	Especifica si la opción debe marcarse como seleccionada.
value	Especifica el valor textual del elemento, por lo que no puede contener nada que no sea texto.

Ejemplo:

```
<form action="/action_page.php" method="get" name="frm">
    <label for="rate">Calificación:</label>
    <select id="rate">
        <option value="01">Mala</option>
        <option value="02">Media</option>
        <option value="03">Buena</option>
    </select>

    <!-- ... -->
</form>
```

1.1.2.11 Elemento output

El elemento OUTPUT especifica que el contenido que se va a representar es el resultado de una operación. Esto puede ser una buena opción a implementar cuando se trata de mostrar el resultado de una operación en dónde el usuario introduce varios valores a través de elementos INPUT.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
for	Especifica los ID de los elementos de formulario con los que se operará para mostrar el resultado en el elemento OUTPUT.
form	Especifica el formulario al que pertenece el elemento cuando se declaran sus elementos fuera del ámbito del formulario. Con respecto a sus posibles valores, el valor de este atributo debe ser el mismo que el indicado por el atributo ID del elemento FORM, de lo contrario, no se hará efectivo.
name	Especifica o asigna el nombre del elemento. El atributo NAME puede ser utilizado para recuperar el valor de los campos del formulario desde el lado del servidor y puede resultar muy útil cuando se desean manipular los elementos desde JavaScript o CSS. Si el valor de este atributo lleva asignado como sufijo [] (los símbolos de corchetes), el identificador de nombre se definirá y actuará como si de un array numérico se tratase, es decir, se podrán referenciar los distintos elementos a través de un índice, siendo, el índice CERO el primero y, N-1, el último.

Ejemplo:

```
<form oninput=”res.value=parseInt(op1.value) + parseInt(op2.value)”>0
    <label>Suma de dos operandos</label>

    <input type=”range” id=”op1” value=”50”>100
    +
    <input type=”range” id=”op2” value=”50”>
    =
    <output name=”res” for=”op1 op2”>100</output>
</form>
```

1.1.2.12 Elemento progress

El elemento PROGRESS especifica que el contenido que se va a representar es una barra de progreso.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
max	Especifica el límite superior del rango de valores aceptado, es decir, el valor máximo que puede aceptar o se puede introducir en el elemento.
value	Especifica el valor numérico inicial del elemento.

Ejemplo:

```
<form oninput=”res.value=parseInt(op1.value) + parseInt(op2.value)”>0
    <label>Progreso de instalación</label>
    <progress id=”progress” value=”54” max=”100”> 54% </progress>
</form>
```

Cabe destacar que, aunque puedan parecerse, una barra de progreso no es una medición escalar. Por esta razón, el elemento PROGRESS no debe utilizarse para mostrar valores indicadores. El correcto uso de este elemento es únicamente para mostrar cómo progresa, o ha progresado, una tarea o proceso.

1.1.2.13 Elemento select

El elemento SELECT especifica que el contenido que se va a representar es un desplegable con una lista de opciones predefinida.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
autofocus	Especifica si se debe tomar el foco automáticamente después de ser renderizado y/o proceso de carga del documento. Sus valores pueden ser TRUE o FALSE.
disabled	Especifica si el elemento está deshabilitado o no. Esta propiedad tendrá efecto con sólo declararla, es decir, en cuanto esté presente e independientemente de su valor, el elemento aparecerá deshabilitado. No obstante, suele declararse con el valor DISABLED.
form	Especifica el formulario al que pertenece el elemento cuando se declaran sus elementos fuera del ámbito del formulario. Con respecto a sus posibles valores, el valor de este atributo debe ser el mismo que el indicado por el atributo ID del elemento FORM, de lo contrario, no se hará efectivo.
multiple	Especifica que el elemento admite la selección o inserción de más de un valor. Cabe destacar que este atributo tendrá efecto con sólo declararlo, es decir, en cuanto esté presente e independientemente de su valor, el elemento permitirá la selección de múltiples valores.
name	Especifica o asigna el nombre del elemento. El atributo NAME puede ser utilizado para recuperar el valor de los campos del formulario desde el lado del servidor y puede resultar muy útil cuando se desean manipular los elementos desde JavaScript o CSS. Si el valor de este atributo lleva asignado como sufijo [] (los símbolos de corchetes), el identificador de nombre se definirá y actuará como si de un array numérico se tratase, es decir, se podrán referenciar los distintos elementos a través de un índice, siendo, el índice CERO el primero y, N-1, el último.
required	Especifica que el elemento debe contener valor, aunque este sea un espacio en blanco, es decir, que no puede ser nulo ni vacío.
size	Especifica el ancho en caracteres que debe tener el elemento.

Ejemplo:

```
<form action=”/action_page.php” method=”get” name=”frm”>
    <label for=”rate”>Calificación:</label>
    <select id=”rate”>
        <option value=”01”>Mala</option>
        <option value=”02”>Media</option>
        <option value=”03”>Buena</option>
    </select>
    <!-- ... -->
</form>
```

1.1.2.14 Elemento textarea

El elemento TEXTAREA especifica que el contenido que se va a representar es una caja de texto con la opción de multilínea, es decir, un control de entrada de datos de múltiples líneas.

Entre los atributos que admite en su configuración, se deben destacar los siguientes:

Atributo	*Descripción*
autofocus	Especifica si se debe tomar el foco automáticamente después de ser renderizado y/o proceso de carga del documento. Sus valores pueden ser TRUE o FALSE.
cols	Especifica la anchura, en caracteres, del elemento.
disabled	Especifica si el elemento está deshabilitado o no. Esta propiedad tendrá efecto con sólo declararla, es decir, en cuanto esté presente e independientemente de su valor, el elemento aparecerá deshabilitado. No obstante, suele declararse con el valor DISABLED.
form	Especifica el formulario al que pertenece el elemento cuando se declaran sus elementos fuera del ámbito del formulario. Con respecto a sus posibles valores, el valor de este atributo debe ser el mismo que el indicado por el atributo ID del elemento FORM, de lo contrario, no se hará efectivo.
maxlength	Especifica la longitud máxima, en caracteres, del valor que puede ser ingresado en el elemento.
name	Especifica o asigna el nombre del elemento. El atributo NAME puede ser utilizado para recuperar el valor de los campos del formulario desde el lado del servidor y puede resultar muy útil cuando se desean manipular los elementos desde JavaScript o CSS. Si el valor de este atributo lleva asignado como sufijo [] (los símbolos de corchetes), el identificador de nombre se definirá y actuará como si de un array numérico se tratase, es decir, se podrán referenciar los distintos elementos a través de un índice, siendo, el índice CERO el primero y, N-1, el último.
placeholder	Especifica el texto que se debe mostrar cómo pista de datos válidos cuando el elemento no contenga un valor. Aunque algunos agentes de usuario pueden no aceptar este atributo, en general es una buena idea hacerlo para ayudar a la usabilidad y accesibilidad web. Si va asociado con otro atributo como PATTERN, la pista a proporcionar debe estar en concordancia con la validez de la entrada.
readonly	Especifica que el elemento es de sólo lectura, es decir, que no permite la inserción de nuevos valores ni la modificación de su valor actual. Cabe destacar que este atributo tendrá efecto con sólo declararlo, es decir, en cuanto esté presente, e independientemente de su valor, provocará que se ponga en modo de sólo lectura. No obstante, no quiere decir que no se pueda seleccionar, resaltar o copiar.
required	Especifica que el elemento debe contener valor, aunque este sea un espacio en blanco, es decir, que no puede ser nulo ni vacío.
rows	Especifica la altura del elemento en líneas de caracteres.
wrap	Especifica si el texto debe incluir o no la definición de nuevas líneas cuando se realiza el envío del formulario para que se ajuste al ancho del elemento establecido. Sus posibles valores son HARD y SOFT, los cuales indican que se agreguen nuevas líneas al texto enviado o no, respectivamente. El carácter de nueva línea añadido al texto será en base al valor del atributo COLS, es decir, cuando el texto llegue a la longitud establecida por COLS, agregará o no un nuevo salto de línea en función de si el atributo WRAP está o no establecido a HARD. El uso de este atributo no se recomienda si se desea mantener un buen nivel de usabilidad y accesibilidad web puesto que puede provocar pérdidas en la legibilidad de los datos si no se utiliza adecuadamente.

Ejemplo:

```
<form action=”/action_page.php” method=”post” name=”frm”>
    <label>
        Descripción
        <textarea id=”desc” rows=”24” cols=”80”></textarea>
    </label>

    <!-- ... -->
</form>
```

1.1.3 ELEMENTOS DISPONIBLES EN CSS

CSS no presenta ninguna propiedad específica para elementos de formulario, sin embargo, cabe destacar que estos elementos pueden aprovecharse de las mismas ventajas que cualquier otro elemento de bloque o caja. Es decir, pueden ser personalizados con propiedades de todo tipo, como son FONT-FAMILY, TEXT-ALIGN, POSITION, DISPLAY, WIDTH, HEIGHT, PADDING, MARGIN, BORDER, BACKGROUND, etc.

No obstante, entre todas ellas, cabe destacar dos:

1.1.3.1 Propiedad box-sizing

Especifica cómo deben asignarse y calcularse el alto y ancho de los elementos. Esto es, si deben incluir los márgenes internos (padding) y/o los bordes, o no. Entre sus posibles valores podemos encontrar:

- **CONTENT-BOX**: Indica que se debe incluir sólo el contenido, e ignorar los márgenes internos y bordes. Es el valor por defecto.
- **BORDER-BOX**: Indica que se deben incluir el contenido, padding y bordes.

NOTA

En general, se puede afirmar que, trabajar con cajas o capas incluyendo los márgenes internos y los bordes es más fácil de manejar y facilita el diseño adaptativo, aunque no siempre.

1.1.3.2 Propiedad resize

Especifica si los elementos TEXTAREA deben permitir la manipulación del tamaño del elemento desde la interfaz que presenta el agente de usuario. Sus posibles valores son:

- **NONE**: Indica que el usuario no puede cambiar el tamaño del elemento.
- **HORIZONTAL**: Indica que el usuario puede cambiar, únicamente, el tamaño del elemento horizontalmente.
- **VERTICAL**: Indica que el usuario puede cambiar, únicamente, el tamaño del elemento verticalmente.
- **BOTH**: Indica que el usuario puede cambiar el tamaño del elemento en ambas direcciones.

NOTA

Aunque esta propiedad puede resultar muy útil, su soporte está muy limitado. De hecho, no está soportado por Internet Explorer, Microsoft Edge 78 o inferior, ni Opera 14 o inferior.

Ejemplo:

```
textarea { resize: none; }
```

1.2 GESTIÓN DE FORMULARIOS DESDE JAVASCRIPT

Cada vez que se produce la carga o actualización de una página, JavaScript rellena la propiedad **forms** del objeto **document** con una referencia a todos los formularios que encuentra de forma automática. Esta propiedad, básicamente, es un array de objetos que puede ser accedido a través de su índice.

Por ejemplo, si quisiésemos ver todos los formularios de la página podríamos iterar esta propiedad de la siguiente forma:

```
for(var i = 0; i < document.forms.length; i++){
    console.log(document.forms[i]);
}
```

A su vez, cada vez que el sistema añade un formulario, este es analizado y se le añaden todos los elementos que lo componen. Todos estos elementos de formulario están disponibles a través de la propiedad **elements**, que también puede ser manipulada como si fuese un array.

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

El acceso a los elementos de formulario a través de estos arrays es un método francamente rápido, sin embargo, en entornos dinámicos no suele ser posible utilizarlo porque la posición de los elementos o, incluso, de los formularios puede cambiar en función de cómo se creen los objetos.

Cuando los entornos son dinámicos y no podemos predecir la posición donde se encontrarán los formularios, lo frecuente es acceder a ellos a través de su propiedad **name** o propiedad **id**.

```
var formName = document.forms.nombreFormulario;
var formID = document.forms.idFormulario;
```

Sin embargo, un formulario no solo dispone de los atributos de nombre y de identificación. También tiene atributos para definir la codificación, el método de envío o la dirección de destino.

Estos atributos son, como uno se puede imaginar, exactamente los mismos que los mostrados en el elemento FORM, expuesto anteriormente. La única diferencia que podremos encontrar es que los identificadores de nombre se referencian en modo Lower Camel Case cuando éstos tienen o llevan guion, es decir, que, por ejemplo, el atributo accept-charset del elemento FORM de HTML será equivalente o equivaldrá a acceptCharset.

Por último, cabe mencionar que, los formularios y todos sus elementos también son accesibles a través de las funciones que proporciona el DOM. A continuación, se muestran un par de ejemplos:

```
var formName = document.querySelector("[name='nombreFormulario']");
var formID = document.getElementById("idFormulario");
```

1.2.1 CREACIÓN Y ENVÍO DE FORMULARIOS

Primero definimos el elemento que actuará como formulario y sus propiedades:

```
// Creamos el elemento de formulario
var f = document.createElement("form");

// Establecemos el método de envío y su URL destino
f.setAttribute('method',"post");
f.setAttribute('action',"cambiarcontraseña.html");
```

Ahora, procedemos a crear sus elementos. Primero definimos los campos de entrada.

```
// Añadimos un elemento de formulario
var i = document.createElement("input");
i.setAttribute('type',"email");
i.setAttribute('name',"email");
```

Seguidamente, definimos el botón que actuará como control de envío.

```
// Añadimos el botón para envío
var b = document.createElement("button");
b.setAttribute('type',"submit");
b.setAttribute('value',"Enviar");
```

Finalmente, añadimos los elementos al formulario de forma ordenada y, si lo deseamos, lo añadimos al body.

```
// Añadimos el input y el button al formulario
f.appendChild(i);
f.appendChild(b);

// Finalmente, lo añadimos al body
document.body.appendChild(f);
```

Efectivamente, lo más frecuente es añadir una declaración de formulario en JavaScript al elemento body, pero, no lo hiciésemos y le añadiésemos un valor a través de su propiedad value, podríamos llamar a su método submit para realizar un envío automático directamente desde memoria.

```
// Enviamos el formulario ahora
f.submit();
```

1.2.2 FUNCIONES DE VALIDACIÓN

La validación de formularios se realiza a través de JavaScript y, hasta no hace tanto, no era casi personalizable ni eficiente. Ahora, sin embargo, gracias a la amplia gama de propiedades que poseen los elementos de formulario, junto con los métodos de notificación y manipulación que nos provee HTML5, podemos realizar validaciones de forma bastante rápida y sencilla.

1.2.2.1 La interfaz validitystate

La interfaz VALIDITYSTATE es un objeto que representa todos los posibles esta-dos por los que puede pasar un elemento de formulario. Además, suele indicar la razón o el motivo por el que se encuentra en ese estado.

Entre sus propiedades más frecuentes podemos encontrar:

Propiedad	*Descripción*
badInput	Esta propiedad devuelve true si ha habido algún problema con la conversión del dato introducido.
customError	Esta propiedad devuelve TRUE si el elemento contiene asignado un error definido por el usuario establecido a través del método SETCUSTOMVALIDITY.
patternMismatch	Esta propiedad devuelve true si el elemento no cumple el patrón definido. Si su valor es true, la pseudo-clase :INVALID de CSS también se activará.
rangeOverflow	Esta propiedad devuelve true si el elemento contiene un valor mayor al provisto por la propiedad MAX. Si su valor es TRUE, las pseudo-clases :INVALID y :OUT-OF-RANGE de CSS también se activarán.
rangeUnderflow	Esta propiedad devuelve true si el elemento contiene un valor menor al provisto por la propiedad MIN. Si su valor es TRUE, las pseudo-clases :INVALID y :OUT-OF-RANGE de CSS también se activarán.
stepMismatch	Esta propiedad devuelve TRUE si el elemento contiene un valor que no concuerda con el paso provisto por la propiedad STEP. Si su valor es TRUE, las pseudo-clases :INVALID y :OUT-OF-RANGE de CSS también se activarán.
tooLong	Esta propiedad devuelve TRUE si el elemento tiene una longitud mayor que la provista por el atributo MAXLENGTH. Si su valor es TRUE, las pseudo-clases :INVALID y :OUT-OF-RANGE de CSS también se activarán.
tooShort	Esta propiedad devuelve TRUE si el elemento tiene una longitud menor a la provista por el atributo MINLENGTH. Si su valor es TRUE, las pseudo-clases :INVALID y :OUT-OF-RANGE de CSS también se activarán.

typeMismatch	Esta propiedad devuelve TRUE si el elemento contiene una sintaxis incorrecta. Sólo es válido para los tipos de INPUT EMAIL y URL. Si su valor es TRUE, la pseudo-clase :INVALID de CSS también se activará.
valid	Esta propiedad devuelve TRUE si el elemento cumple todas las restricciones requeridas. Si su valor es TRUE, la pseudo-clase :VALID de CSS también se activará.
valueMissing	Esta propiedad devuelve TRUE si el elemento es requerido y se encuentra vacío. Si su valor es TRUE, la pseudo-clase :INVALID de CSS también se activará.

1.2.2.2 Propiedades y métodos

A continuación, se muestran los principales métodos y propiedades que pue-den ser utilizados en la validación de formularios.

Propiedad	*Descripción*
validity	Esta propiedad resulta ser un objeto que devuelve un conjunto de datos de tipo VALIDITYSTATE y permite conocer el resultado de todos los posibles problemas que se han producido tras realizar una comprobación de validación.
validationMessage	Esta propiedad contiene el mensaje generado tras el proceso de validación. Si el elemento no ha sido validado aún o ha pasado con éxito todo el proceso de validación, el contenido de esta propiedad estará establecido a cadena vacía. Si, por el contrario, existe algún error o problema con la validación, esta propiedad mostrará el mensaje de error o información sobre el problema.
setCustomValidity	Este método permite definir mensajes de error personalizados en los elementos de formulario. El mensaje, proporcionado como parámetro, es guardado en la propiedad VALIDATIONMESSAGE.
checkValidity	Este método comprueba si se cumplen las restricciones que tiene definido el elemento de formulario. Si todo es correcto, es decir, que pasa la validación, devolverá TRUE, en cualquier otro caso, devolverá FALSE.

1.2.2.3 Eventos

A continuación, se muestran los principales eventos que pueden ser utilizados en la validación de formularios.

Propiedad	*Descripción*
invalid	Cada vez que se solicita la acción de enviar un formulario, se realiza una acción de validación previamente. Si el proceso de validación no tuvo éxito, el navegador lanza una especie de excepción que marca al elemento como inválido y le asigna la pseudo-clase de CSS :INVALID. Pues bien, si además de controlar la validación interna queremos o necesitamos gestionarla de forma externa, para esto, tenemos el evento INVALID. El evento INVALID es lanzado cuando el elemento realiza el proceso de validación y no cumple alguna de sus restricciones. Una vez, dentro de este evento podemos, por ejemplo, mostrar los mensajes personalizados que hemos crea-do para nuestra aplicación en el lugar de los predefinidos.

1.2.3 VALIDACIONES DE DATOS EN PÁGINAS WEB

1.2.3.1 Validaciones con casillas de verificación y de tipo radio

Las validaciones de tipo checkbox y radio suelen tener como único objetivo, verificar que el elemento está seleccionado o chequeado.

Para realizar este cometido se suele recurrir a un único método. En concreto, se suele usar la propiedad CHECKED asociada a los elementos INPUT de tipo radio o de casilla de verificación.

```
function isChecked(el){
    if(el.checked){
        return true
    } else {
        return false
    }
}
isChecked(document.getElementById("input1"));
```

1.2.3.2 Validaciones alfanuméricas

Las validaciones alfanuméricas tienen como objetivo, permitir la entrada de datos sin formato. Esto es, permitir la entrada de todo tipo de caracteres, entre los que podemos encontrar números, letras y caracteres o símbolos especiales.

Para realizar este cometido podemos recurrir a dos métodos. El primero y más sencillo es recurrir al tipo de dato TEXT que provee el elemento INPUT de HTML.

```
<input type="text" id="input1" name="input1" />
```

Sin embargo, este tipo de validaciones son muy liberales con los datos de entrada, por lo que se suele hacer necesario recurrir a otros métodos algo más restrictivos.

El segundo más sencillo es recurrir a un INPUT de tipo TEXT combinado con el atributo PATTERN de HTML, el cual utiliza expresiones regulares.

```
<input type="text" id="input1" name="input1"
       pattern="[0-9A-Fa-f]+" />
```

Si nos fijamos en el ejemplo anterior, veremos que lo que se está validando es que sea un valor en hexadecimal. Ahora bien, si este método no nos convence por la razón que sea, siempre se puede recurrir a la evaluación mediante el uso de expresiones regulares o mediante el tratamiento de caracteres.

En nuestro caso, si lo que deseamos es realizarlo a través del método de evaluación de expresiones regulares, podemos hacer algo como lo siguiente:

```
function isHexadecimal(value){
   var hex=new RegExp(/^[0-9A-F]+$/, 'ig');

   if(hex.test(value)) return true
   else return false;
}

isHexadecimal(document.getElementById("input1").value);
```

Ahora, si lo que deseamos es realizarlo a través del método de tratamiento de cadenas, podemos hacer algo como lo siguiente:

```
function isHexadecimal(value){
   var hex="0123456789ABCDEFabcdef";

   for(var i = 0; i < value.length; i++){
      if (hex.indexOf(value.charAt(i)) == -1){
         return false;
      }
   }
   return true;
}

isHexadecimal(document.getElementById("input1").value);
```

1.2.3.3 Validaciones numéricas

Las validaciones numéricas tienen como objetivo, impedir al usuario que introduzca valores que no sean de tipo número. Esto es, nada que no sea un valor que no se corresponda con un conjunto de dígitos y un único símbolo para indicar la parte decimal.

Para realizar este cometido podemos recurrir a tres métodos. El más sencillo es utilizar el tipo de dato NUMBER que provee el elemento INPUT de HTML.

```
<input type="number" id="input1" name="input1" />
```

Sin embargo, este tipo de validaciones pueden presentar varios problemas porque, dependiendo del navegador y su versión, puede permitir la introducción de varios símbolos punto e, incluso, varios símbolos de coma. Chrome es buen ejemplo de ello, aunque en sus últimas versiones este problema ya lo haya corregido.

El segundo más sencillo es recurrir a un INPUT de tipo TEXT combinado con la función isNaN de JavaScript.

```
function isValidNumber(valor) {

    if (isNaN(value)) {
        return falso;
    }
}
isValidNumber(document.getElementById("input1").value)
```

Ahora bien, si lo que deseamos es validar números con el símbolo coma en vez de punto para la diferenciar la parte decimal, o necesitamos hacer un tratamiento especial, lo que podemos hacer es recurrir a un método algo más elaborado que se basa en la evaluación de expresiones regulares.

```
function isValidNumber(valor) {
    var rexpr = /^\d*\.?\d*$/;
    if (rexpr.test(valor)) {
        return true;
    } else {
        return false;
    }
}

isValidNumber(document.getElementById("input1").value)
```

1.2.3.4 Validaciones de tipo fecha

Las validaciones de tipo fecha tienen como objetivo, obligar al usuario a introducir valores que se correspondan con uno de los formatos universales de fecha.

Estos formatos son conocidos como Little Endian, que indica que el formato de fecha está establecido como DD-MM-YYYY, Medium Endian, que indica que el formato de fecha está establecido como MM-DD-YYYY o Big Endian, que indica que el formato de fecha está establecido como YYYY-MM-DD.

Para realizar este cometido podemos recurrir a dos métodos. El más sencillo es utilizar el tipo de dato DATE que provee el elemento INPUT de HTML.

```
<input type="date" id="input1" name="input1" />
```

Sin embargo, este tipo de validación es un poco inconsistente porque es el navegador quién define el modo de presentación y porque, además, internamente el valor actual siempre se normaliza al formato Big Endian.

El segundo más sencillo es recurrir a un INPUT de tipo TEXT combinado con el atributo PATTERN.

```
<input type="text" id="input1" name="input1"
       pattern="[0-9]{2}-[0-9]{2}-[0-9]{4}" />
```

Este método, eliminará el formato fecha establecido por el navegador y su disparador, pero con ganaremos en homogeneidad porque, el formato interno y externo se corresponderán y el navegador nos informará que la fecha es incorrecta cuando el formato no se corresponda.

Ahora bien, como alguno podrá haberse dado cuenta, el caso anterior no validará fechas reales, por lo que se vuelve necesario hacer una comprobación en JavaScript. Esta validación se puede realizar de múltiples formas, sin embargo, la más sencilla y eficaz es recurrir al objeto Date y la función isNaN de JavaScript.

```
<input type="text" id="input1" name="input1"
       pattern="[0-9]{2}-[0-9]{2}-[0-9]{4}"
       oninput="isValidDate(this.value)" />
```

```
// Validación en modo Little Endian
function isValidDate(value) {
    var p1 = value.split("-")[0];
    var p2 = value.split("-")[1];
    var p3 = value.split("-")[2];
    var date = !isNaN(new Date(p3 + "-" + p2 + "-" + p1).getTime());

    if(date){
        return true
    } else {
        return false
    }
}
isValidDate(document.getElementById("input1").value);
```

1.2.3.5 Validaciones de tipo tiempo

Las validaciones de tipo tiempo tienen como objetivo, obligar al usuario a introducir valores que se correspondan con los formatos HH:MM:SS, con todas sus posibles omisiones y combinaciones. Es decir, formato completo con horas, minutos y segundos, sólo hora y minuto, sólo hora, etcétera.

Para realizar este cometido podemos recurrir a tres métodos. El más sencillo es utilizar el tipo de dato TIME que provee el elemento INPUT de HTML.

```
<input type="time" id="input1" name="input1" />
```

Sin embargo, aunque este tipo de validación está en líneas generales bastante bien definida, cada navegador la define como mejor le parece, lo que genera en ocasiones algo de desconcierto. Eso, sin contar que Internet Explorer no la soporta.

Por ello, en muchos casos, se hace necesario recurrir a uno de los siguientes métodos. El segundo más sencillo es recurrir a un INPUT de tipo TEXT combinado con el atributo PATTERN.

```
<input type="text" id="input1" name="input1"
pattern="^(0[0-9]|1[0-9]|2[0-3]):(0[0-9]|[1-5][0-9]):(0[0-9]|[1-5][0-9])$" />
```

Al observar la expresión regular propuesta se aprecia una cierta complejidad en su definición, sin embargo, es bastante efectiva ya que sólo valida valores reales.

El tercer método, que resulta ser algo más elaborado, es recurrir al objeto Date y la función isNaN de JavaScript.

```
function isValidTime(value) {
    var hh = value.split(":")[0];
    var mm = value.split(":")[1];
    var ss = value.split(":")[2];
    var dt = new Date();
```

```
        dt = new Date(dt.getFullYear() + "-" +
                      dt.getMonth() + "-" +
                      dt.getDate() + " " +
                      hh + ":" + mm + ":" + ss);

    if(!isNaN(dt.getTime())){
        return true
    } else {
        return false
    }
}
isValidTime(document.getElementById("input1").value);
```

1.2.3.6 Validaciones en desplegables

Las validaciones de tipo lista o desplegable tienen como objetivo, obligar al usuario a seleccionar una opción de las expuestas o, también, comprobar si la opción se corresponde con las circunstancias correctas.

Por ejemplo, podría suceder que tuviésemos varios desplegables que están relacionados entre sí. Es decir, el valor del primero, manipula o cambia los valores del siguiente.

Para realizar este cometido podemos recurrir a multitud de métodos, pero en esta ocasión, sólo se mostrará cómo saber si el usuario ha seleccionado o no una opción de un desplegable, fundamentalmente, porque los desplegables dependientes suelen estar manipulados desde la parte del servidor y se suelen actualizar a través de una llamada Ajax que se ejecuta a partir del evento CHANGE.

Dicho esto, para saber si el usuario ha seleccionado o no una opción de un desplegable, lo único que se debe hacer es recurrir a la comprobación de unos valores concretos de la propiedad SELECTEDINDEX de JavaScript asociada al elemento SELECT.

```
function isSelected(el) {
    var index = el.selectedIndex;
    if(index == null || index == 0){
        return false
    } else {
        return true;
    }
}
```

Cabe destacar que, para que este método sea funcional, la primera opción del SELECT debe estar desactivada o tener un carácter meramente informativo. Es decir, debería tener una estructura similar a la siguiente:

```
<select name="select1" id="select1" onchange="isSelected(this)">
    <option value="no-valid">Selccione una opción...</option>
    <option value="0">Exportar</option>
    <option value="1">Importar</option>
</select>
```

1.2.3.7 Validaciones de tipo email

Las validaciones de tipo email tienen como objetivo, obligar al usuario a introducir valores que se correspondan con una expresión de cadena compuesta por un nombre, seguido del símbolo arroba y un identificador de dominio.

Para realizar este cometido podemos recurrir a tres métodos. El más sencillo es utilizar el tipo de dato EMAIL que provee el elemento INPUT de HTML.

```
<input type=”email” id=”input1” name=”input1” />
```

Sin embargo, este tipo de validación puede presentar varios problemas porque, puede dar por válido ciertos valores que, a simple vista, se ven que son falsos. Este es el caso, por ejemplo, del valor “d@d.com”, que este tipo de elementos de formulario lo suelen permitir.

Para corregir esto, se podría establecer una longitud mínima al nombre y/o al dominio, pero al final, nunca se suele usar por la diversidad de nombres que hay o existen en Internet.

Aun así, si se desea realizar esta mejora se puede recurrir al uso de expresiones regulares. Este es, precisamente, el segundo método que consiste en utilizar un INPUT de tipo TEXT combinado con el atributo PATTERN.

```
<input type=”text” id=”input1” name=”input1”
       pattern=”^[a-z0-9!#$%&’*+/=?^_`{|}~-]+(?:\.[a-z0-
9!#$%&’*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-
9](?:[a-z0-9-]*[a-z0-9])?$” />
```

Al observar la expresión regular propuesta puede que se aprecie bastante complejidad en su definición, sin embargo, es posiblemente la más efectiva y la más recomendad ya que resulta ser la implementación del estándar oficial RFC 5322.

El tercer método suele ser el mismo que el anterior, con la diferencia de que se realiza a través de JavaScript.

```
function isValidEmail(value) {
    var re = /^[a-z0-9!#$%&’*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&’*+/=?^_`{|}~-]
+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-
9])?$/;

    if(re.test(value)){
        return true
    } else {
        return false
    }

}
isValidEmail(document.getElementById(“input1”).value);
```

1.2.3.8 Validaciones de tipo teléfono

Las validaciones de tipo teléfono tienen como objetivo, obligar al usuario a introducir valores que se correspondan con un valor numérico con una longitud determinada. Este valor, a menudo, presenta un prefijo y algún tipo de símbolo que sirve como separador o agrupador.

Para realizar este cometido se suele recurrir a tres métodos. El más sencillo es utilizar el tipo de dato TEL que provee el elemento INPUT de HTML.

```
<input type="tel" id="input1" name="input1" />
```

Sin embargo, este tipo de validación casi nunca funciona en navegadores de escritorio y no del todo bien en algunos navegadores móviles. Por esta razón, se hace imprescindible el uso de expresiones regulares, que es lo que utilizaremos en los dos siguientes métodos.

El segundo métodos es utilizar el tipo de dato TEXT que provee el elemento INPUT de HTML combinado con el atributo PATTERN.

```
<input type="text" id="input1" name="input1"
       pattern="^\d{3}-\d{3}-\d{3}$" />
```

Si observamos el ejemplo anterior, veremos que el patrón o formato de entrada se corresponde a uno de los tantos y tantos formatos posibles que puede admitir un tipo de dato referente a un número de teléfono. Por ello, a continuación, se muestra una tabla con algunos de los formatos más extendidos en Internet.

Propiedad	*Descripción*	*Descripción*
900900900	/^[0-9]{9}$/	Nueve dígitos todos seguidos.
999 999 999	/^[0-9]{3}\s[0-9]{3}\s[0-9]{3}$/	Nueve dígitos agrupados de tres en tres y separados por espacio.
999-999-999	/^[0-9]{3}-[0-9]{3}-[0-9]{3}$/	Nueve dígitos agrupados de tres en tres y separados por guion.
999.99.99.99	/^\d{3}\.\d{2}\.\d{2}\.\d{2}$/	Nueve dígitos separados por punto en formato de tres, dos, dos y dos.
(+34) 999 999 999	/^\(\+\d{2,3}\)\s\d{3}\s\d{3}\s\d{3}$/	Prefijo internacional con el símbolo + seguido de dos o tres dígitos, espacio en blanco y nueve dígitos agrupados de tres en tres separados por espacio.

El tercer método, como hemos dicho antes, resulta ser el mismo que el anterior, con la diferencia de que se realiza a través de JavaScript.

```
function isValidPhone(value) {
    var re = /^\(\+\d{2,3}\)\s\d{3}\s\d{3}\s\d{3}$/;

    if(re.test(value)){
        return true
    } else {
```

```
        return false
    }

}
isValidPhone(document.getElementById("input1").value);
```

1.2.4 IMPLEMENTACIÓN DE CÓDIGO DE VALIDACIÓN

Imaginemos que deseamos comprobar que el valor introducido en un INPUT denominado STATUS concuerde con uno de los tres posibles valores: "Asignado", "En Progreso" o "Finalizado".

Para ello, lo primero que necesitaremos es declarar el código HTML con un LABEL, un campo de entrada de texto y dos elementos adicionales para poder establecer los posibles mensajes de error.

```
<h1>Prueba de validación</h2>
<form name="frm" enctype="multipart/form-data" method="get">
    <label>
       Estado:
       <input id="status" type="text" oninput="check(this)" />
    </label>

    <h2>Mensaje tras validación</h2>
    <div id="validateMsg"></div>

    <h2>Listado de errores de validity</h2>
    <div id="validity"></div>
</form>
```

Una vez insertado el HTML, necesitamos declarar la funcionalidad de JavaScript que realizará la validación. Para ello, podríamos hacer algo como:

```
function check(input) {
    // Recuperamos el valor introducido y los elementos a manipular
    var val = input.value;
    var vm  = document.getElementById("validateMsg");
    var va  = document.getElementById("validity");

    // Comprobamos si el valor es válido
    if (input.value != "Asignado" &&
        input.value != "En Progreso" &&
        input.value != "Finalizado") {

        // Como no es válido, establecemos un mensaje de error entendible
        // y se lo asignamos a la función setCustomValidity
        var msg = '"' + val + '" no es un estado.';
        input.setCustomValidity(msg);
```

```
        // Asignamos el mensaje de error a nuestro objeto validityMsg
        vm.innerHTML = input.validationMessage;
        va.innerHTML = ‘’;

        // Mostramos el listado de las restricciones incumplidas
        for(var key in input.validity){
            var status = input.validity[key];
            if(status){
                va.innerHTML += key + “: “ + status.toString();
                va.innerHTML += “<br/>”;
            }
        }

    } else {
        // Todo correcto.
        // Eliminamos el mensaje y el listado.
        input.setCustomValidity(‘’);
        vm.innerHTML = “”;
        va.innerHTML = “”;
    }
}

// Recuperamos el elemento con ID status y le asignamos un evento para
// comprobar su validez
var s = document.getElementById(“status”);
s.addEventListener(“invalid”, check, true);
```