

HTML



CSS



Formularios HTML5. Incrustación de fuentes y otras propiedades de CSS3

1. Objetivos

En este cuarto tema se pretenden conseguir los siguientes objetivos:

- ✚ Crear formularios utilizando los nuevos tipos de entrada para formularios HTML5.
 - ✚ Utilizar los nuevos atributos de HTML5 para validación de formularios.
 - ✚ Validar formularios utilizando la nueva API Forms de HTML5.
 - ✚ Aplicar estilos a los formularios con las nuevas pseudo-clases de CSS3.
- Incrustar fuentes en nuestras páginas web.
- ✚ Diseñar webs utilizando múltiples fondos de imagen para un mismo elemento.

2. Introducción

Si en el tema anterior vimos como introducir contenidos multimedia en nuestras páginas web, en este tema hablaremos, principalmente, sobre el uso de formularios.

Veremos los nuevos tipos de entrada que se han introducido en HTML5 (search, email, url, tel, etc.), que nos ayudarán a indicar que tipo de datos se espera que el usuario introduzca en los campos del formulario. Veremos también los nuevos atributos que se han creado para ayudar en la validación de estos campos (required, pattern, etc.), o para ampliar las funcionalidades de los mismos (placeholder, multiple, autofocus, etc.).

Aunque con todas estas nuevas características la validación de los formularios se simplifica bastante, en ocasiones necesitaremos realizar validaciones más complejas, por ejemplo, cuando unos campos dependan de otros. Para conseguir esto, veremos una nueva API JavaScript que incorpora HTML5 (API Forms), que nos permitirá realizar validaciones más complejas de forma sencilla.

En lo que se refiere a la parte de CSS3, hablaremos sobre las nuevas pseudo-clases que se introducen en el estándar para aplicar estilos a los elementos de los formularios según sean requeridos, con datos válidos o con datos inválidos (:required, :valid e :invalid). También veremos cómo mejorar nuestros diseños utilizando varias imágenes de fondo en un mismo elemento y cómo incrustar fuentes en nuestros sitios web para poder utilizarlas sin preocuparnos de que el usuario las tenga instaladas en su equipo.

En el siguiente tema, veremos cómo utilizar el nuevo elemento canvas que incorpora HTML5 para poder dibujar sobre nuestra página, dándonos la

posibilidad de crear animaciones, o incluso, videojuegos dentro de nuestra página utilizando JavaScript. También veremos cómo crear animaciones utilizando CSS3.

3. HTML5. Formularios.

La Web 2.0 está completamente enfocada al usuario, y, cuando el usuario es el centro de atención, todo está relacionado con interfaces: cómo hacerlas más intuitivas, más naturales, más prácticas y, por supuesto, más atractivas. Los formularios web son las interfaces más importantes de todas, permiten a los usuarios insertar datos, tomar decisiones, comunicar información y cambiar el comportamiento de una aplicación.

HTML5 ha introducido nuevos elementos de formulario: tipos de `<input>`, atributos y otras características. Muchas de estas características las hemos estado utilizando en nuestras interfaces durante años: la validación de formularios, cuadros combinados y similares. La diferencia es que, mientras que antes teníamos que recurrir a JavaScript para crear estos comportamientos, ahora muchos de ellos están disponibles directamente en el navegador; todo lo que tenemos que hacer es establecer un atributo en el elemento adecuado para que estén disponibles.

HTML5 no sólo facilita la tarea del desarrollador, también es mejor para el usuario. Manejando la validación del lado del cliente de forma nativa por el navegador, habrá una mayor consistencia a través de los diferentes sitios, y muchas páginas se cargarán más rápido al no tener necesidad de descargar todo el JavaScript de validación.

Habitualmente, los desarrolladores web encuentran tediosa y aburrida la labor de crear formularios. Afortunadamente, HTML5 nos aporta gran variedad de funcionalidades que nos van a posibilitar eliminar gran cantidad de código de validación repetitivo y nos van a facilitar la labor a la hora de crear las interfaces de usuario.


Vamos a ver a continuación un ejemplo de formulario de registro codificado con los elementos antiguos existentes en HTML4/XHTML. Después, a medida que vayamos avanzando, mejoraremos este formulario aplicando los nuevos elementos que vayamos estudiando.

```
<form id="registro" method="post">
  <h1>Regístrate!</h1>
  <ul>
    <li>
      <label for="nombre">Mi nombre es:</label>
      <input type="text" id="nombre" name="nombre">
    </li>
    <li>
      <label for="email">Mi email es:</label>
      <input type="text" id="email" name="email">
    </li>
    <li>
      <label for="recuerdame">Recuérdame en este equipo </label>
      <input type="checkbox" value="true" id="recuerdame">
    </li>
    <li>
      <label for="url">Mi dirección web es:</label>
      <input type="text" id="url" name="url">
    </li>
    <li>
      <label for="password">Me gustaría que mi password fuera:</label>
      <span>(al menos 6 caracteres, sin espacios)</span>
      <input type="password" id="password" name="password">
    </li>
    <li>
      <label for="conocimientos">En una escala de 1 a 10, mis
      conocimientos de HTML5 son:</label>
      <input type="text" id="conocimientos" name="conocimientos">
    </li>
    <li>
      <label for="fechaInicio">Por favor, inicia mi suscripción el
      día:</label>
      <input type="text" id="fechaInicio" name="fechaInicio">
    </li>
    <li>
      <label for="cantidad">Me gustaría recibir <input type="text"
      name="cantidad" id="cantidad" value="1"> copias de <cite>El manual
      de HTML5</cite>.</label>
```

```
        </li>
        <li>
        <label for="tambienCSS3">También regístrame para <cite>El manual de
        CSS3</cite></label>
        <input id="tambienCSS3" name="tambienCSS3" type="checkbox">
        </li>
        <li>
            <input type="submit" id="registro-submit" value="Envíame"/>
        </li>
    </ul>
</form>
```

Este formulario, con los estilos aplicados, se visualizará como se muestra en la siguiente imagen:

El formulario lo podéis encontrar en el fichero `registro.html` dentro de la carpeta `registro` del fichero de recursos4.zip, también podéis encontrar el fichero de estilos que se le ha aplicado y las imágenes necesarias en las carpetas `css` e `imgs` que podéis encontrar en esa misma carpeta.



Regístrate!

Mi nombre es:

.....

Mi email es:

.....

Recuérdame en este equipo

☐

Mi dirección web es:

.....

Me gustaría que mi password fuera:
(al menos 6 caracteres, sin espacios)

.....

Mi password repetida es:

.....

En una escala de 1 a 10, mis conocimientos de HTML5 son:

.....

Por favor, inicia mi suscripción el día:

.....

Me gustaría recibir copias de *El manual de HTML5*.

También regístrate para *El manual de CSS3*

☐

ENVÍAME

Estamos utilizando los elementos de formulario que han estado disponibles desde las primeras versiones de HTML. El usuario sabe qué tipo de datos se espera en cada campo porque se utilizan etiquetas y párrafos para indicárselo. Funciona, pero sin duda puede ser mejorado.

En este tema vamos a mejorar este formulario para incluir características de HTML5. HTML5 ofrece nuevos tipos de entrada específicos para direcciones de correo electrónico, direcciones web (URLs), números, fechas, etc.

Además de los nuevos tipos de entrada, HTML5 también introduce nuevos atributos para estos tipos y los ya existentes. Estos nos van a permitir, entre otras cosas, marcar los campos como obligatorios o declarar qué tipo de datos son aceptables, todo ello sin JavaScript.

Nuevos atributos para formularios

Como se puede ver en el ejemplo anterior la estructura del formulario y sus atributos siguen siendo igual que en especificaciones previas. Sin embargo, existen nuevos atributos para el elemento `<form>`:

- 🌈 **autocomplete:** Este es un viejo atributo que se ha vuelto estándar en esta especificación. Puede tomar dos valores: `on` y `off`. El valor por defecto es `on`. Cuando se configura como `off` los elementos `<input>` pertenecientes al formulario tendrán la función de autocompletar desactivada, sin mostrar entradas previas como posibles valores. Puede ser implementado en el elemento `<form>` o en cualquier elemento `<input>` independientemente. Es posible tenerlo a `on` en el formulario y a `off` en algún `<input>` específico y viceversa.

Es una buena idea desactivar esta funcionalidad en campos sensibles, como pueden ser tarjetas de crédito o números de cuenta, y para aquellos campos cuya información nunca va a ser reutilizada, como los típicos CAPTCHA.

🚦 **novalidate:** Una de las características de formularios en HTML5 es la capacidad propia de validación. Los formularios son automáticamente validados. Para evitar este comportamiento, podemos usar el atributo `novalidate`. Para lograr lo mismo para elementos `<input>` específicos, existe otro atributo llamado *formnovalidate*. Ambos atributos son booleanos, ningún valor tiene que ser especificado (su presencia es suficiente para activar su función). El valor del atributo *formnovalidate* sobrescribe el del atributo `novalidate` del formulario.

Nuevos tipos de `<input>`

Los elementos más importantes en los formularios son los `<input>`. Estos elementos pueden cambiar sus características gracias al atributo `type` (tipo). Este atributo determina qué clase de entrada se espera que introduzca el usuario. Los tipos disponibles hasta el momento eran: `text` (para textos en general) y sólo unos pocos más específicos como `password` o `submit`. HTML5 ha añadido nuevos tipos, incrementando de este modo las posibilidades para este elemento.

En HTML5 estos nuevos tipos no sólo están especificando qué clase de entrada se espera, sino también diciéndole al navegador qué debe hacer con la información recibida. El navegador procesará los datos introducidos de acuerdo al valor del atributo `type` y validará la entrada o no.

Tipo `search`

El tipo `search` (búsqueda) no controla la entrada, es sólo una indicación para los navegadores. Al detectar este tipo de campo algunos navegadores

cambiarán el diseño del elemento para hacerle ver al usuario cuál es el propósito del campo. Algunos navegadores, como por ejemplo Chrome, añaden la posibilidad de vaciar el campo de búsqueda haciendo click en un aspa que aparece a la derecha del campo.

En nuestro formulario de registro no tenemos ningún campo de búsqueda, pero un ejemplo típico podría ser el siguiente:

```
<form id="formbusqueda" method="get">
  <input type="search" id="busqueda" name="busqueda">
  <input type="submit" value="buscar">
</form>
```

Este ejemplo sin aplicar ningún tipo de estilo se mostraría en Chrome de la siguiente forma:



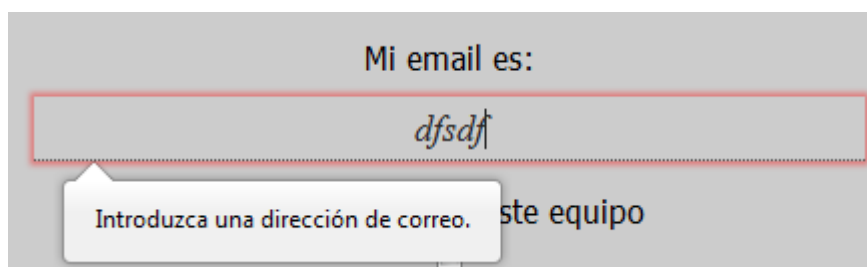
Tipo email

Casi todo formulario en la web ofrece un campo para ingresar una dirección de email, pero hasta ahora el único tipo de campo disponible para esta clase de datos era text. El tipo text representa un texto general, no un dato específico, por lo que teníamos que controlar la entrada con código JavaScript para estar seguros de que el texto ingresado correspondía a un email válido. Ahora el navegador se hace cargo de esto con el nuevo tipo email. Vamos a cambiar el campo email de nuestro formulario de registro para usar el nuevo tipo email:

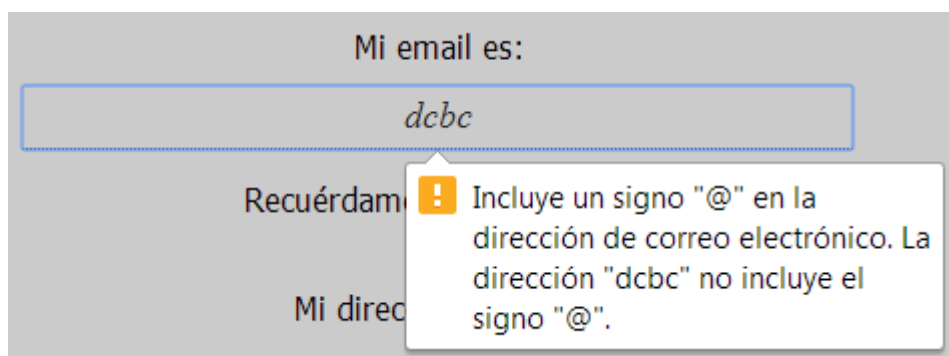
```
<label for="email">Mi email es:</label>
```

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

Cuando cambiamos el tipo a email, aparentemente no se produce ningún cambio en la interfaz del usuario, la entrada sigue esperando un texto. Los cambios los vemos cuando introducimos un valor incorrecto en el campo. Por ejemplo, algunos navegadores como Firefox, Chrome y otros, muestran un mensaje de error cuando se intentan enviar los datos del formulario pulsando el botón submit habiendo una entrada incorrecta en el campo email.



Mensaje de error email no válido en Firefox



Mensaje de error email no válido en Chrome

Más adelante, veremos cómo cambiar el mensaje de error que se muestra.

Tipo url

Este tipo de campo trabaja exactamente igual que el tipo email, pero es

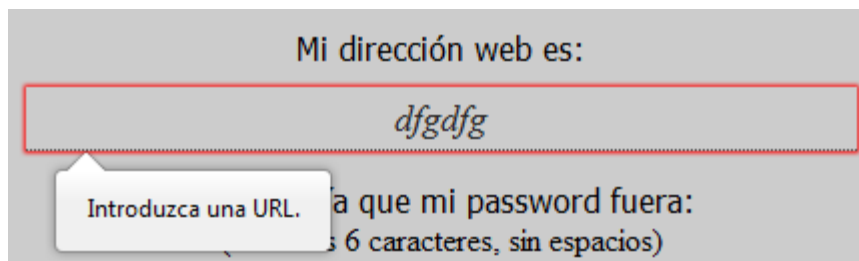
específico para direcciones web. Está destinado a recibir sólo URLs absolutas y retornará un error si el valor es inválido.

Utilizaremos el tipo url para el campo dirección web del formulario de registro:

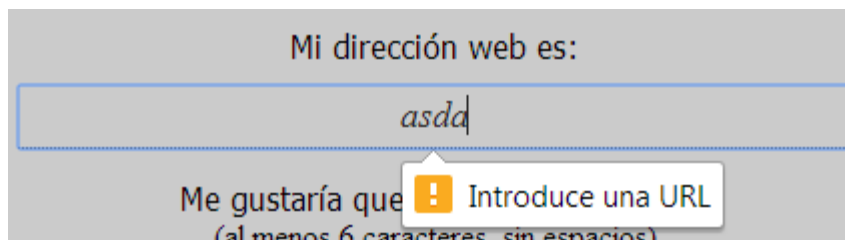
```
<label for="url">Mi dirección web es:</label>
```

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

Los navegadores que soportan el tipo url, informan de que la entrada no es válida si la URL no está correctamente formateada.



Mensaje de error url no válida en Firefox



Mensaje de error url no válida en Chrome

Sólo se valida el formato general de la URL, así, por ejemplo, q://example.xyz será válido, aunque q:// no sea un protocolo real. Más adelante, veremos cómo podemos resolver este problema y validar formatos más específicos.




Tipo tel

Este tipo de campo es para números telefónicos. A diferencia de los tipos email y url, el tipo tel no requiere ninguna sintaxis en particular (cada

país tiene un formato determinado para los números de teléfono). Es sólo una indicación para el navegador en caso de que necesite hacer ajustes de acuerdo al dispositivo en el que la aplicación se ejecuta, por ejemplo, cuando se muestra el teclado en un dispositivo móvil.

Tipo number

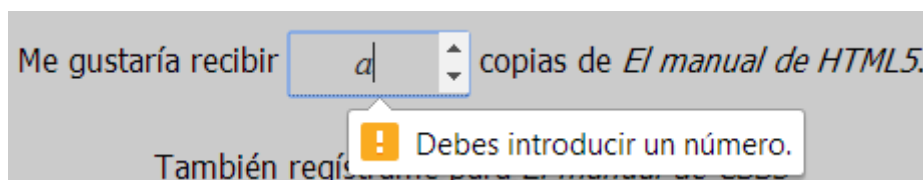
Como su nombre indica, el tipo number se utiliza para pedir entradas numéricas. Algunos navegadores muestran este tipo de campo como un "spinner" con una flecha arriba para incrementar el valor del campo y otra abajo para decrementarlo. Existen algunos atributos nuevos que pueden ser útiles para este campo:

-  **min:** El valor de este atributo determina el mínimo valor aceptado.
-  **max:** El valor de este atributo determina el máximo valor aceptado.
-  **step:** El valor de este atributo determina el incremento o decremento del campo en cada paso. El valor por defecto es 1.

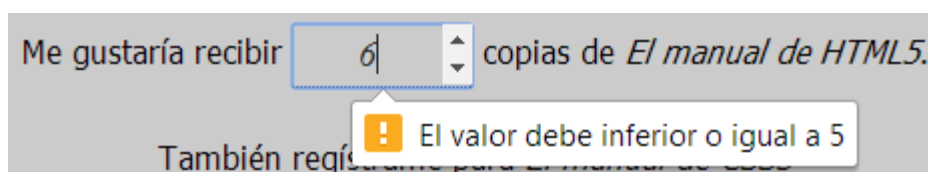
Por ejemplo, si declaramos un valor de 5 para step en un campo que tiene un valor mínimo de 0 y máximo de 10, el navegador no le permitirá especificar valores entre 0 y 5 o entre 5 y 10. Utilizaremos el tipo number para el campo cantidad de nuestro formulario de registro:

```
<label for="cantidad">Me gustaría recibir <input type="number"
name="cantidad" id="cantidad" value="1" min="0" max="5"> copias de
<cite>El manual de HTML5</cite>.</label>
```

Cuando se introduce una entrada inválida el navegador informará del error:



Mensaje de error número erróneo en Chrome



Mensaje de error número fuera de rango en Chrome

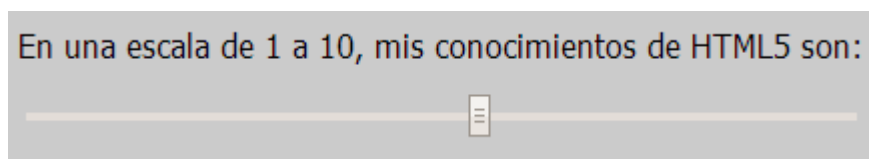
Tipo range

Este tipo de campo hace que el navegador construya una nueva clase de control que no existía previamente. Este nuevo control le permite al usuario seleccionar un valor a partir de una serie de valores o rango. Normalmente, se muestra como una barra deslizable (slider).

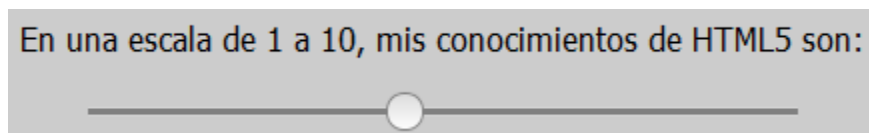
Como en el tipo number, se pueden utilizar los atributos min, max y step. La diferencia entre number y range según las especificaciones, es que con el tipo range el valor exacto del número indicado no es importante. Es ideal para entradas donde esperas un número no necesariamente preciso, por ejemplo, en una encuesta de satisfacción para clientes o cosas por el estilo.

El valor por defecto será el valor medio del rango indicado. Usaremos el tipo range en el campo conocimientos de nuestro formulario de registro:

```
<label for="conocimientos">En una escala de 1 a 10, mis conocimientos de HTML5  
son:</label>  
<input type="range" min="1" max="10" id="conocimientos" name="conocimientos">
```



Tipo range en Chrome



Tipo range en Firefox

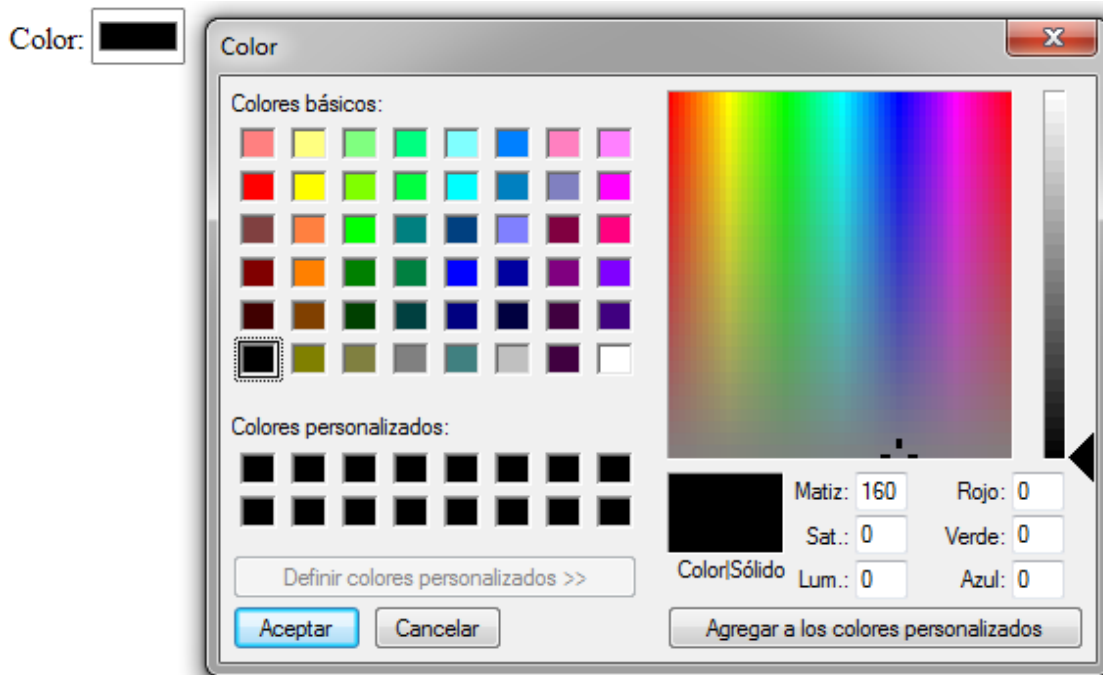
Tipo color

El tipo color nos permitirá introducir colores en formato hexadecimal (#FEFEFE).

Para ello, el navegador nos mostrará una paleta de colores, entre los cuales podremos seleccionar el que nos interese. En nuestro formulario de registro no tenemos ningún campo de este tipo, pero podemos ver el siguiente ejemplo:

```
<label for="campoColor">Color: </label>  
<input id="campoColor" name="campoColor" type="color">
```

El ejemplo, después de pulsar el botón que abre la ventana para seleccionar el color, se muestra en Chrome de la siguiente forma:



Tipos de <input> para fechas y horas

Hay varios nuevos tipos de entrada para fecha y hora: date, datetime, datetime-local, month, time, and week. Todas las entradas de fecha y hora aceptan datos formateados según la norma ISO 8601¹.

Los navegadores que soporten estos tipos de datos los mostrarán con un campo de tipo calendario, mediante el cual podremos desplegar y seleccionar la fecha con la ayuda del ratón.

- 📅 date: comprende la fecha (día, mes, año), pero no la hora. Por ejemplo, 25/03/2014.

¹ http://es.wikipedia.org/wiki/ISO_8601

`<label for="fecha">Fecha: </label>`

`<input id="fecha" name="fecha" type="date">`

Fecha:

marzo de 2014

lun	mar	mié	jue	vie	sáb	dom
24	25	26	27	28	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

🚦 month: Sólo incluye el mes y el año. Por ejemplo, febrero de 2014.

`<label for="mes">Mes: </label>`

`<input id="mes" name="mes" type="month">`

Mes:

marzo de 2014

lun	mar	mié	jue	vie	sáb	dom
24	25	26	27	28	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

🚦 week: Indicaremos el número de la semana (del 1 al 52) y el año. Por ejemplo, Semana 11 de 2014.

`<label for="semana">Semana: </label>`

`<input id="semana" name="semana" type="week">`

Semana:

marzo de 2014

Semana	lun	mar	mié	jue	vie	sáb	dom
9	24	25	26	27	28	1	2
10	3	4	5	6	7	8	9
11	10	11	12	13	14	15	16
12	17	18	19	20	21	22	23
13	24	25	26	27	28	29	30
14	31	1	2	3	4	5	6

- time: Nos permitirá introducir la hora, teniendo en cuenta que la hora irá de 0 a 23. Por ejemplo, 22:05. Se mostrará en el navegador que lo soporte, por ejemplo Chrome, como un spinner que nos permitirá incrementar las horas y/o los minutos o vaciar el campo.

`<label for="hora">Hora: </label>`

`<input id="hora" name="hora" type="time">`

Hora:

- datetime: En este tipo de entrada podremos indicar tanto la fecha como la hora separadas por una "T" y seguidas, o bien por una "Z" para representar UTC (Coordinated Universal Time), o bien por una zona horaria especificada por un carácter "+" o un carácter "-". Por ejemplo, 17/03/2014T10:45+1:00. En los navegadores actuales, este tipo de entrada se muestra como una entrada de texto y no se realiza ninguna validación de los datos introducidos.
- datetime-local: Es idéntico al datetime, excepto, que se omite la zona horaria. Por ejemplo, 17/03/2014T10:45. En este caso, navegadores como el Chrome muestran el campo como un spinner en el que podemos incrementar o decrementar los diferentes valores o vaciar el control.

```
<label for="fecha hora">Fecha-hora: </label>
```

```
<input id="fecha hora" name="fecha hora" type="datetime-local">
```

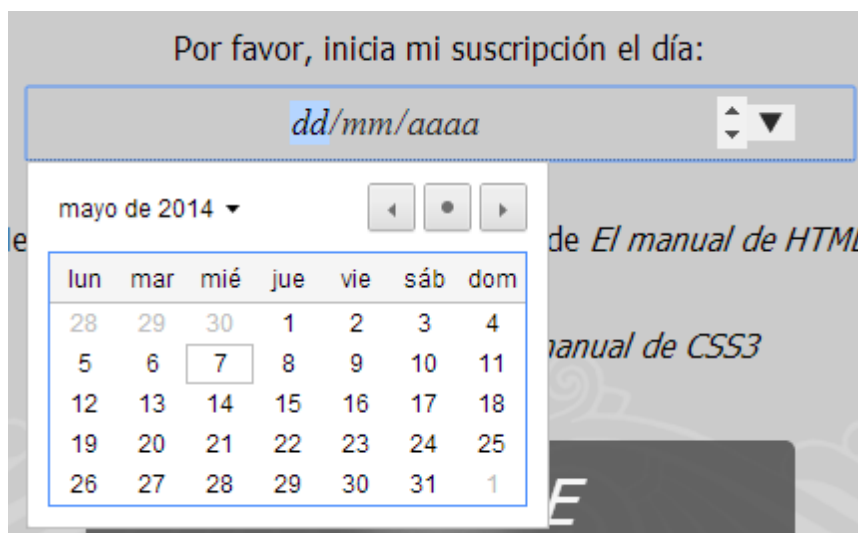
Fecha-hora:

Vamos a cambiar el campo fechaInicio de nuestro formulario de registro al tipo date:

```
<label for="fechaInicio">Por favor, inicia mi suscripción el día:</label>
```

```
<input type="date" id="fechaInicio" name="fechaInicio">
```

En Chrome se mostraría así:



Nuevos atributos para los campos de formulario

Durante años, los desarrolladores han escrito (o copiado y pegado) fragmentos de JavaScript para validar la información que el usuario introducía en los campos de los formularios: qué elementos son obligatorios, qué tipos de datos se aceptan y cosas por el estilo. HTML5 nos proporciona gran cantidad de atributos que nos van a permitir indicar qué valores son aceptables para un campo, e informar al usuario de los errores cometidos, todo ello sin necesidad de usar nada de JavaScript.

Los navegadores que soportan estos atributos HTML5 compararán los datos introducidos por el usuario mediante expresiones regulares que el desarrollador podrá indicar. Luego comprobarán si todos los campos obligatorios están realmente completados, etc.

Además, la inclusión de estos atributos no afectará a los navegadores más antiguos; simplemente ignorarán los atributos que no entienden. A continuación, estudiaremos los nuevos atributos que incorpora HTML5.

El atributo `required`

Este atributo booleano no dejará que el formulario se envíe si el campo se encuentra vacío. Por ejemplo, cuando usamos el tipo `email` para recibir una dirección de email, el navegador comprueba si la entrada es un email válido o no, pero validará la entrada si el campo está vacío. Cuando se incluye el atributo `required`, la entrada será válida sólo si se cumplen las dos condiciones: que el campo no esté vacío y que el valor introducido esté de acuerdo con los requisitos del tipo de campo.

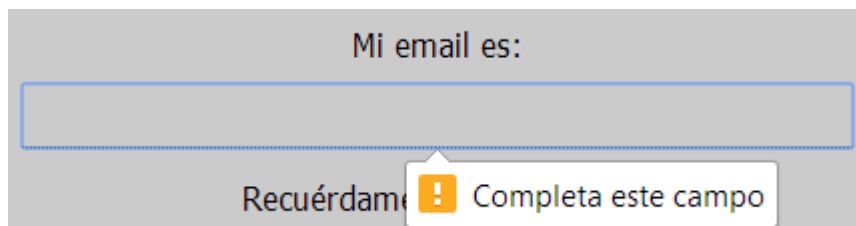
Si un campo obligatorio está vacío o los datos introducidos no son válidos, el envío de los datos del formulario fallará al hacer `submit` y el foco se moverá al primer campo que haya fallado. Además algunos navegadores, como por ejemplo Firefox, Opera o Chrome mostrarán un mensaje de error al usuario.

Este atributo se puede utilizar en cualquier tipo de campo, excepto `button`, `range`, `color` y `hidden`, los cuales, generalmente, tienen un valor por defecto. Al ser un atributo booleano, bastará con indicar la palabra `required` en el elemento correspondiente para que el atributo se active.

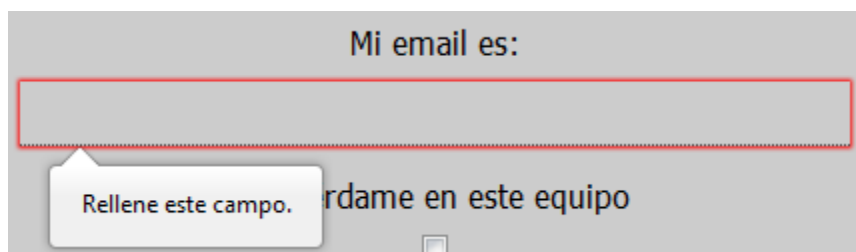
Vamos a usar el atributo `required` dentro de nuestro formulario de registro para los campos `email`, `password1`, `password2` y `fechaInicio`.

```
<label for="email">Mi email es:</label>
```

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



Mensaje de error de campo `required` en Chrome



Mensaje de error de campo `required` en Firefox

El atributo `placeholder`

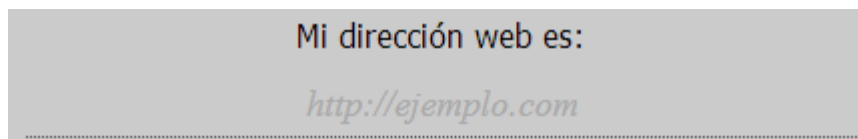
Especialmente en tipos de campo `search`, pero también en entradas de texto normales, el atributo `placeholder` representa una sugerencia corta, una palabra o frase provista para ayudar al usuario a ingresar la información correcta. El valor de este atributo es presentado en pantalla por los navegadores dentro del campo, como una marca de agua que desaparece cuando el elemento obtiene el foco.

Los desarrolladores han implementado esta funcionalidad durante años utilizando JavaScript, pero ahora está incorporada de forma nativa en el

navegador. Para nuestro formulario de registro usaremos este atributo en los campos url y fechaInicio:

```
<label for="url">Mi dirección web es:</label>
```

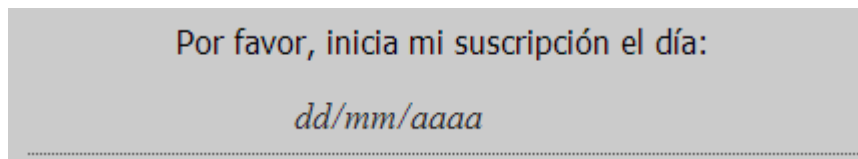
```
<input type="url" id="url" name="url" placeholder="http://ejemplo.com">
```



Visual representation of the first form element: A label "Mi dirección web es:" followed by an input field with the placeholder text "http://ejemplo.com".

```
<label for="fechaInicio">Por favor, inicia mi suscripción el día:</label>
```

```
<input type="date" id="fechaInicio" name="fechaInicio" required  
placeholder="dd/mm/aaaa">
```



Visual representation of the second form element: A label "Por favor, inicia mi suscripción el día:" followed by an input field with the placeholder text "dd/mm/aaaa".

El atributo pattern

El atributo pattern se utiliza para propósitos de validación. Usa expresiones regulares² para personalizar reglas de validación. Algunos de los tipos de campo ya estudiados validan cadenas de texto específicas, como email o url, pero no permiten hacer validaciones personalizadas, como por ejemplo un código postal que consiste en 5 números. No existe ningún tipo de campo predeterminado para esta clase de entrada.

² Expresiones regulares son un tema complejo y no relacionado directamente con HTML5. Para obtener información adicional al respecto podéis visitar la siguiente url:

<http://www.regular-expressions.info/>

El atributo `pattern` nos permite crear nuestro propio tipo de campo para controlar esta clase de valores no ordinarios. Se suele utilizar el atributo `title` para indicar qué tipo de entrada se espera que se introduzca en el campo.

Vamos a utilizar el atributo `pattern` en los campos `password1` y `password2` de nuestro formulario de registro para obligar a que la entrada tenga al menos 6 caracteres sin espacios:

```
<label for="password1">Me gustaría que mi password fuera:</label>
```

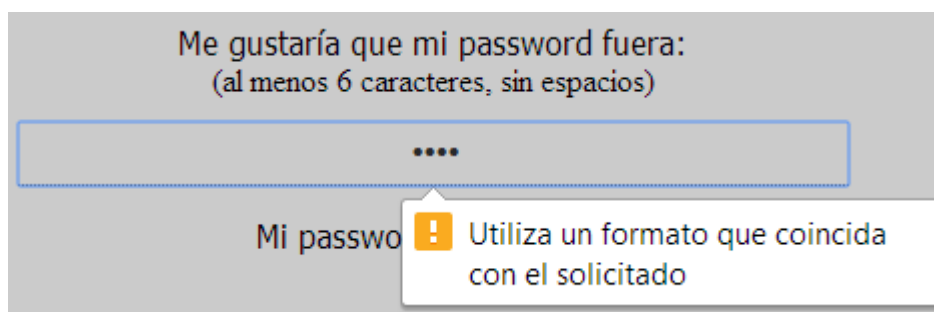
```
<p>(al menos 6 caracteres, sin espacios)</p>
```

```
<input type="password" id="password1" name="password1" required  
pattern="\S{6,}">
```

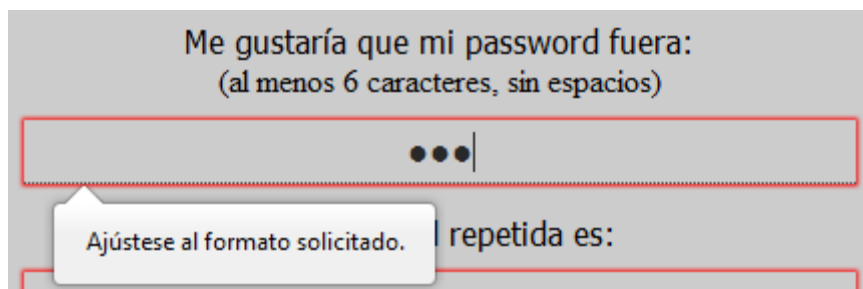
...

```
<label for="password2">Mi password repetida es:</label>
```

```
<input type="password" id="password2" name="password2" required  
pattern="\S{6,}">
```



Mensaje de error de campo `pattern` en Chrome

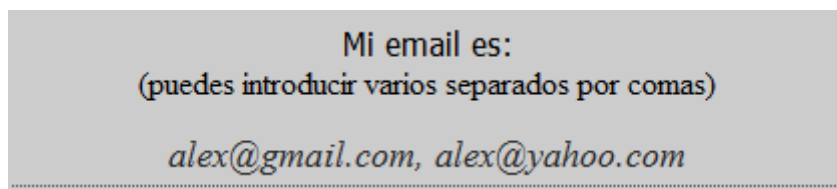


Mensaje de error de campo pattern en Firefox

El atributo multiple

El atributo multiple es otro atributo booleano que puede ser usado en algunos tipos de campo (por ejemplo, email o file) para permitir entradas múltiples en el mismo campo. Los valores insertados deben estar separados por comas para ser válidos. Utilizaremos el atributo multiple para el campo email de nuestro formulario de registro:

```
<label for="email">Mi email es:</label>  
<p>(puedes introducir varios separados por comas)</p>  
<input type="email" id="email" name="email" required multiple>
```



Mi email es:
(puedes introducir varios separados por comas)
alex@gmail.com, alex@yahoo.com

El atributo form

El atributo form nos permite declarar elementos para un formulario fuera del ámbito de las etiquetas <form>, es decir, que no estén anidados en el formulario. Hasta ahora, para construir un formulario teníamos que escribir las etiquetas <form> de apertura y cierre y, luego, declarar cada elemento del formulario entre ellas. En HTML5 podemos insertar los elementos en cualquier parte del código haciendo referencia al formulario al que pertenecen usando el atributo form. En este atributo indicaremos el valor del atributo id del formulario al que referencia. Cuando se envíen los datos del formulario, también se enviarán los campos que no estén dentro de él, pero tengan su id en

el atributo form.

El atributo autofocus

El atributo booleano autofocus indica que un control de formulario debería recibir el foco tan pronto como la página sea cargada. Sólo un campo de formulario puede tener este atributo en una página dada. Utilizaremos el atributo autofocus en el campo nombre de nuestro formulario de registro:

```
<label for="nombre">Mi nombre es:</label>
```

```
<input type="text" id="nombre" name="nombre" autofocus>
```

El atributo readonly

El atributo readonly es similar al atributo disabled existente en CSS2.1: hace que el usuario no pueda editar el campo del formulario. La diferencia con disabled es que con readonly el campo puede recibir el foco y los datos que contenga serán enviados cuando se envíe el formulario.

Otros tipos de campos para los formularios HTML5

Ya hemos visto los nuevos valores para los <input> junto con algunos atributos que se han incorporado a los elementos de formulario. Pero HTML5 tiene más cosas que ofrecernos; existen cuatro nuevos campos de formulario en HTML5: progress, meter, output y datalist.

El elemento datalist y el atributo list

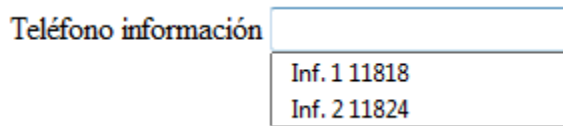
El elemento <datalist> es específico de formularios. Se usa para construir una lista de ítems que luego, con la ayuda del atributo list, será usada como sugerencia en un campo del formulario.

```
<datalist id="informacion">  
<option value="11818" label="Inf. 1 11818">  
<option value="11824" label="Inf. 2 11824">  
</datalist>
```

Este elemento utiliza el elemento `<option>` en su interior para crear la lista de datos a sugerir. Con la lista ya declarada, lo único que resta es referenciarla desde un elemento `<input>` usando el atributo `list`:

```
<label for="telefono">Teléfono información</label>  
<input type="tel" name="telefono" id="telefono" list="informacion">
```

El elemento `informacion` mostrará posibles valores para elegir.



Es importante observar que lo que se muestra como sugerencia es el valor del atributo `label` del elemento `<option>`, pero lo que se carga en el campo al seleccionar la sugerencia es el valor del atributo `value`.

Los elementos `<progress>` y `<meter>`

Estos dos nuevos elementos de HTML5 no son específicos de formularios, pero debido a que representan el progreso en la realización de una tarea, y usualmente estas tareas son comenzadas y procesadas a través de formularios, puede ser incluido dentro del grupo de elementos para formularios. Ambos, permiten el marcado de los datos que están siendo medidos o calibrados de alguna manera. La diferencia entre ellos es bastante sutil:

🚦 **<progress>** se utiliza para describir el estado actual de un proceso que va cambiando hasta llegar a completarse independientemente de si el estado de finalización está definido o no. La barra de progreso de una descarga es un ejemplo de progreso.


🚦 **<meter>** representa un elemento cuyo rango se conoce, lo que significa que tiene definido valores mínimos y máximos. Un ejemplo sería el espacio disponible de un disco.

El elemento **<progress>** puede tener un atributo **max** para indicar el punto en el cual la tarea se completará, y un atributo **value** para indicar el estado actual de la tarea, aunque ambos atributos son opcionales. Ejemplo:

<p>Progreso de la descarga:

<progress value="22" max="100"></progress>

</p>


Progreso de la descarga: 

Normalmente, actualizaremos los valores del elemento **<progress>** mediante JavaScript para cambiar dinámicamente el valor del porcentaje a medida que avanza la tarea. El elemento **<meter>** tiene seis atributos asociados:

Atributo	Descripción
form	Especifica el id del <form> al que pertenece el elemento <meter>
high	Especifica el intervalo que es considerado como un valor alto.
low	Especifica el intervalo que es considerado como un valor bajo.
max	Especifica el valor máximo del intervalo.
min	Especifica el valor mínimo del intervalo.
optimum	Especifica cuál es el valor óptimo.
value	Especifica el valor actual, este atributo es obligatorio.

Aquí tenemos un ejemplo de <meter> que nos indica el porcentaje de uso de un disco:

```
<p>uso de disco actual:<meter value="63" min="0" max="320" low="10"
high="300" title="gigabytes">63 GB</meter>
</p>
```

uso de disco actual: 

El elemento <output>

El propósito del elemento <output> es visualizar el resultado de un cálculo. Se debe utilizar cuando queramos que el usuario pueda ver el valor, pero no modificarlo directamente, y cuando el valor se puede derivar de otros valores introducidos en el formulario. Un ejemplo de uso serían los impuestos calculados de un pedido en un carrito de la compra.

El valor del elemento <output> está contenido entre la etiquetas de apertura y de cierre. Por lo general, se utilizará JavaScript para actualizar este valor. Este elemento tiene un atributo, que se utiliza para hacer referencia a los identificadores de los campos de formulario cuyos valores se

utilizaron para realizar el cálculo del valor del campo. Es importante señalar que el nombre y el valor del elemento `<output>` se envían junto con el formulario.

API Forms

Al igual que la mayoría de nuevas características de HTML5, los formularios cuentan con su propia API para personalizar todos los aspectos de procesamiento y validación.

Existen diferentes formas de aprovechar el proceso de validación en HTML5: podemos usar los tipos de campo para activar el proceso de validación por defecto (por ejemplo, email) o convertir un campo como text (o cualquier otro) en un campo obligatorio usando el atributo `required`. También podemos crear campos con validaciones especiales usando el atributo `pattern` para personalizar requisitos de validación. Sin embargo, cuando se trata de aplicar mecanismos complejos de validación (por ejemplo, combinando campos o comprobando los resultados de un cálculo) deberemos recurrir a nuevos recursos provistos por esta API.

Personalizar los mensajes de error

Los navegadores que soportan HTML5 muestran un mensaje de error cuando el usuario intenta enviar un formulario que contiene un campo inválido. Podemos crear nuestras propias validaciones con mensajes de validación personalizados usando el método `setCustomValidity(mensaje)`.

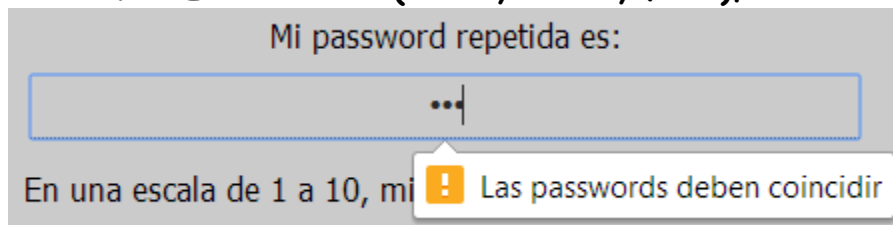
Con este método especificamos un error personalizado que mostrará un mensaje cuando el formulario se envíe. El funcionamiento es muy sencillo, si en

el momento del envío alguno de los campos tiene un mensaje personalizado no vacío, el navegador entenderá que ha habido un error de validación y mostrará el mensaje en el campo indicado. El error se eliminará cuando llamemos a la función **setCustomValidity** pasando un mensaje vacío: **setCustomValidity("")**.

Por ejemplo, para que cuando se pulse el botón enviar en nuestro formulario de registro se verifique que las dos password introducidas son iguales haremos lo siguiente:

```
function iniciar() {  
    registro=document.getElementById("registro");  
    password1=document.getElementById("password1");  
    password2=document.getElementById("password2");  
    /*El evento input se producirá cada vez que se pulse una tecla */  
    password1.addEventListener("input", validacion, false);  
    password2.addEventListener("input", validacion, false);  
}  
  
function validacion() {  
    if(password1.value!=password2.value)  
        password2.setCustomValidity('Las passwords deben coincidir');  
    else  
        password2.setCustomValidity('');  
}
```

```
window.addEventListener("load", iniciar, false);
```



El evento invalid

Cada vez que el usuario envía el formulario, se dispara un evento llamado `invalid` si se detecta un campo que no pasa la validación. El evento es disparado por el elemento que produce el error. Podemos agregar un manejador para este evento y así ofrecer una respuesta personalizada. Por ejemplo, en el siguiente ejemplo se pondrá un color de fondo amarillo a aquellos elementos que no pasen la validación:

```
function iniciar() {  
    registro=document.getElementById("registro");  
    registro.addEventListener("invalid", accionInvalid, true);  
}  
  
function accionInvalid(evento){  
    /* obtenemos el elemento que no ha pasado la validación */  
    var elemento = evento.target;  
    elemento.style.background='yellow';  
}  
  
window.addEventListener("load", iniciar, false);
```

Este código tendría un problema, y es que cuando corriamos el error de validación, el campo seguirá con fondo amarillo. Más adelante veremos cómo hacer algo muy parecido (y más apropiado), evitando este inconveniente.

Controlar la validación

Para tener control absoluto sobre el envío del formulario y el momento de validación, podemos sustituir el `<input type="submit">` por un `<button>`.

Cuando se pulse este botón, el formulario se enviará, pero sólo si todos

sus elementos son válidos. Para ello, debemos agregar un manejador para el evento click del botón.

Usando el método **checkValidity()**, que nos devolverá true si no hay ningún campo que no pase la validación, solicitamos al navegador que realice el proceso de validación, y sólo enviamos el formulario, usando el tradicional método **submit()**, cuando ya no hay más condiciones inválidas.

```
function iniciar() {  
    registro=document.getElementById("registro");  
    registroSubmit=document.getElementById("registro-submit");  
    registroSubmit.addEventListener("click", enviar, false);  
}  
  
function enviar() {  
    if(registro.checkValidity()) {  
        registro.submit();  
    }  
}  
window.addEventListener("load", iniciar, false);
```

Si necesitáramos realizar cualquier otra validación antes del envío, podríamos hacerlo en la función enviar.

Validación en tiempo real

Uno de los problemas que tenemos en los ejemplos anteriores es que la validación se realiza sólo cuando pulsamos el botón enviar. Para poder realizar la validación en tiempo real a medida que el usuario va rellorando los campos, tenemos que aprovechar los atributos provistos por el objeto **ValidityState**, al cual accedemos en todos los campos de formulario a través de su propiedad

validity. Este objeto tendrá las siguientes propiedades booleanas:

- 🚦 **valid**: es true cuando el contenido actual del elemento es válido.
- 🚦 **valueMissing**: es true cuando el atributo `required` fue declarado y el campo está vacío.
- 🚦 **typeMismatch**: es true cuando la sintaxis de la entrada no corresponde con el tipo especificado (por ejemplo, si el texto insertado en un tipo de campo `email` no es una dirección de email válida).
- 🚦 **patternMismatch**: es true cuando la entrada no corresponde con el patrón provisto por el atributo `pattern`.
- 🚦 **tooLong**: es true cuando el atributo `maxlength` fue declarado y la entrada es más extensa que el valor especificado para este atributo.
- 🚦 **rangeUnderflow**: es true cuando el atributo `min` fue declarado y la entrada es menor que el valor especificado para este atributo.
- 🚦 **rangeOverflow**: es true cuando el atributo `max` fue declarado y la entrada es más grande que el valor especificado para este atributo.
- 🚦 **stepMismatch**: es true cuando el atributo `step` fue declarado y su valor no corresponde con los valores de atributos como `min`, `max` y `value`.
- 🚦 **customError**: es true cuando declaramos un error personalizado usando el método `setCustomValidity()` visto anteriormente.

Para controlar estos estados de validación, debemos utilizar la sintaxis `elemento.validity.estado` (donde `estado` es cualquiera de los valores listados arriba). Podemos aprovechar estos atributos para realizar la validación del formulario a medida que se van introduciendo los datos de la siguiente manera:

```
function validarEntrada(evento){
/* obtenemos el elemento sobre el que se está produciendo la entrada */
    var elemento = evento.target;
    if (elemento.validity.valid)
        elemento.style.background = "transparent";
    else
        elemento.style.background = "yellow";
}

function iniciar() {
    registro=document.getElementById("registro");
    /* el evento se producirá cuando se pulsa una tecla sobre cualquier
    campo del formulario */
    registro.addEventListener("input", validarEntrada, false);
}

window.addEventListener("load", iniciar, false);
```

De esta forma, el usuario sabrá si los datos son válidos a medida que los va introduciendo.

4. CSS3

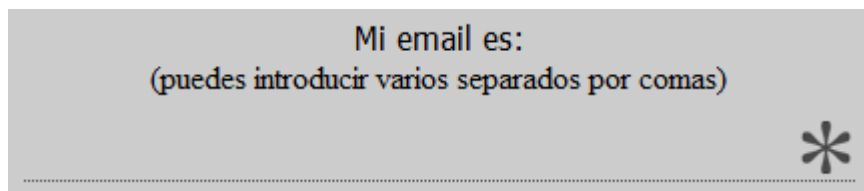
Hemos visto en el punto anterior las nuevas características que incorpora HTML5 para el manejo de formularios. A continuación, veremos una serie de nuevas características que incorpora CSS3 que nos pueden venir muy bien para mejorar el aspecto de nuestros formularios y de nuestras páginas web en general.

Aplicando estilos a campos requeridos o inválidos

Podemos aplicar estilos a elementos de formularios requeridos con la pseudo-clase `:required`. También podemos aplicar estilos a campos válidos o no válidos con las pseudo-classes `:valid` e `:invalid`.

Con estas pseudo-classes, podemos proporcionar indicaciones visuales a los usuarios que muestren qué campos son obligatorios o si la validación es correcta. Por ejemplo, para que en nuestro formulario de registro aparezca un asterisco (*) en los campos que sean requeridos, haremos lo siguiente:

```
input { background: transparent no-repeat top right; }  
input:required { background-image: url('../images/required.png'); }
```

Un formulario de email obligatorio. El campo de texto tiene un fondo gris claro y una línea punteada en la parte inferior. El texto "Mi email es:" está en color azul. Debajo de él, el texto "(puedes introducir varios separados por comas)" está en color azul. A la derecha del campo, hay un icono de un asterisco (*).

Campo email obligatorio

Vemos que el campo email, que es obligatorio, aparece con la imagen del asterisco a la derecha.

Muchas veces, no nos interesará aplicar el mismo estilo a todos los tipos

de input, por ejemplo, los botones suelen tener estilos diferentes. Para solucionar esto, podemos utilizar la pseudo-clase :not.

```
input:not([type=range]):not([type=submit]):not([type=button]):not([type=checkbox box]){  
    background: transparent no-repeat top right;  
}  
input:not([type=range]):not([type=submit]):not([type=button]):not([type=checkbox box]):required{  
    background-image: url('../imgs/required.png');  
}
```


Con estas reglas, evitamos que se inserte la imagen del asterisco, en aquellos tipos de entrada que no nos interesa.

Esto mismo lo podemos hacer para los elementos válidos e inválidos, es decir, aquellos que contengan datos que pasan la validación y aquellos que contengan datos que no la pasan. Al igual que antes, evitaremos que se apliquen estilos en algunos tipos de entrada.

```
input:not([type=range]):not([type=submit]):not([type=button]):not([type=checkbox box]):invalid{  
    background-image: url('../imgs/invalid.png');  
}  
input:not([type=range]):not([type=submit]):not([type=button]):not([type=checkbox box]):valid{  
    background-image: url('../imgs/valid.png');  
}
```

Mi email es:
(puedes introducir varios separados por comas)

alex



Campo email con datos no válidos

Mi email es:
(puedes introducir varios separados por comas)

alex@gmail.com



Campo email con datos válidos

Un problema que nos encontramos es que aparecerá la imagen de campo válido o inválido en todos los campos del formulario a los que les hayamos aplicado los estilos, por lo tanto, la imagen de campo obligatorio no se mostrará.

Para solucionar el problema, haremos que sólo se apliquen los estilos de campo válido o inválido cuando el elemento tenga el foco. Esto lo conseguimos utilizando la pseudo-clase :focus.

```
input:not([type=range]):not([type=date]):not([type=submit]):not([type=button]):not([type=checkbox]):not([type=number]):focus:invalid {  
    background-image: url('../imgs/invalid.png');  
}  
  
input:not([type=range]):not([type=date]):not([type=submit]):not([type=button]):not([type=checkbox]):not([type=number]):focus:valid {  
    background-image: url('../imgs/valid.png');  
}
```

Debemos tener en cuenta que algunos navegadores, como por ejemplo Firefox, muestran una sombra roja alrededor de los elementos inválidos. Para evitar esta incongruencia con el resto de navegadores podemos añadir la siguiente regla css:

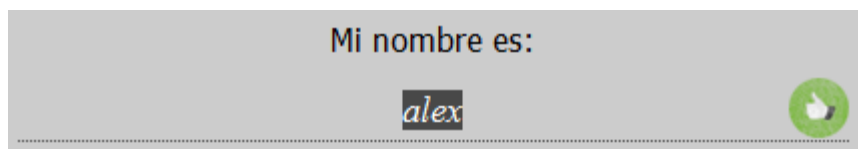
```
:invalid { box-shadow: none; }
```

Estilos para el texto seleccionado

CSS3 incorpora un nuevo pseudo-elemento para el texto seleccionado de la página, de forma que podemos cambiar los estilos del texto que seleccionamos. Este selector es `::selection`. Y para que funcione en Firefox tendremos que utilizar también el prefijo correspondiente (`::-moz-selection`). Por ejemplo, para cambiar los estilos del texto seleccionado de nuestro formulario de registro haremos lo siguiente:

```
::-moz-selection{ background: #484848; color:#fff; text-shadow:  
none; } ::selection { background:#484848; color:#fff; text-shadow: none;  
}
```

Firefox aplicará estos estilos tanto al texto seleccionado de los campos del formulario como al texto seleccionado de la página, sin embargo, otros navegadores, como Chrome, sólo se los aplicarán al texto seleccionado de la página.



Campo con el texto seleccionado

Múltiples imágenes de fondo

Otra de las mejoras que incorpora CSS3 es que nos permite colocar más de una imagen de fondo a un mismo elemento. La sintaxis que utilizaremos para colocar múltiples imágenes de fondo a un elemento será la misma que para colocar una única imagen, simplemente separaremos los valores para cada imagen individual con una coma. Por ejemplo:

background-image:

```
url(primerImagen.jpg),  
url(segundaImagen.gif),  
url(terceraImagen.png);
```

De la misma forma, podemos utilizar el atajo para la propiedad **background:**

background:

```
url(primerImagen.jpg) no-repeat 0 0,  
url(segundaImagen.gif) no-repeat 100% 0,  
url(terceraImagen.png) no-repeat 50% 0;
```

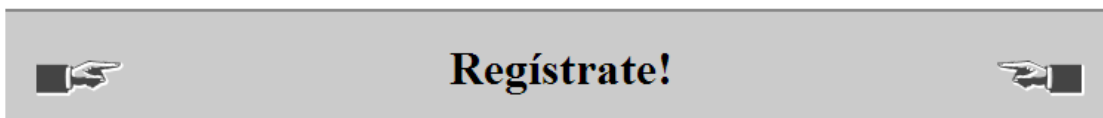
Vamos a utilizar dos imágenes de fondo para el título de nuestro formulario de registro:

form h1 {

```
font-size: 2em;  
padding-bottom: 20px;
```



```
background:
    url(../imgs/bg-formtitle-left.png) left 13px no-repeat,
    url(../imgs/bg-formtitle-right.png) right 13px no-repeat;
}
```



Redimensionar las imágenes de fondo

CSS3 también nos ofrece la posibilidad de redimensionar las imágenes de fondo mediante la propiedad `background-size`. Aunque esta propiedad se puede indicar mediante el atajo `background` a continuación de la posición de la imagen y separada por una barra (/), en muchos navegadores esto no funciona correctamente y es preferible indicarla de forma separada.

Esta propiedad funciona con los siguientes prefijos para los diferentes navegadores:

`-webkit-background-size`

`-moz-background-size`.

`background-size`.

En esta propiedad indicaremos dos valores: el alto y el ancho. Debemos tener cuidado a la hora de indicar estos valores en píxeles, ya que, podemos provocar que la imagen se distorsione si la relación de aspecto entre el ancho y el alto no es adecuada. Para evitar este problema podemos indicar un único valor y el otro se calculará de forma automática para mantener la relación de aspecto. Así, las siguientes declaraciones son equivalentes:

background-size: 100px auto;

background-size: 100px;

Si tuviéramos varias imágenes de fondo podríamos indicar sus tamaños separando los valores por comas, pero si sólo indicamos un tamaño, este se aplicaría para todas ellas:

background-size: 100px auto, auto auto;

background-size: 100px, auto auto;

El tamaño predeterminado de la imagen de fondo es el tamaño real de la imagen. A veces, la imagen es sólo un poco más pequeña o más grande que el elemento que la contiene. En estos casos, nos puede interesar indicar que la imagen se redimensione para cubrir todo el elemento, esto lo haremos utilizando la palabra `cover`:

background-size: cover;

También podemos indicar el tamaño de la imagen en porcentajes del elemento que la contiene. `background-size: 50%`; Vamos a indicar que el tamaño de las imágenes de fondo del título de nuestro formulario de registro es de 100px:

form h1 {

font-size: 2em;

padding-bottom: 20px;

background:

url(../imgs/bg-formtitle-left.png) left 13px no-repeat,

url(../imgs/bg-formtitle-right.png) right 13px no-repeat;

```
-webkit-background-size: 100px auto;  
-moz-background-size: 100px auto;  
background-size: 100px auto;  
}
```



Fuentes incrustadas

El problema con las fuentes o tipos de letra es tan viejo como la propia web. Normalmente, los usuarios de la web tienen un número limitado de fuentes instaladas en sus ordenadores, además, estas fuentes son diferentes de un usuario a otro y, la mayoría de las veces, muchos usuarios tendrán fuentes que otros no. Durante años, los sitios webs sólo pudieron utilizar un grupo limitado de fuentes confiables: Arial, Verdana, Times, Georgia y unas pocas más.

Para solucionar este problema, los diseñadores web recurrían a trucos como crear archivos de imagen que contenían los títulos del sitio, logos, botones, etc. Pero estas soluciones tenían muchos inconvenientes:

- ✚ Cuando estos elementos requerían estados variables, como por ejemplo un botón que se activa o desactiva cuando el ratón pasa sobre él, era necesario crear una imagen diferente para cada estado del elemento.
- ✚ Si el texto cambiaba había que crear de nuevo la imagen que lo contenía.
- ✚ Al tener que utilizar imágenes el peso de la página crecía considerablemente.
- ✚ Los textos que estaban contenidos en las imágenes no eran leídos por los robots de los buscadores, con la correspondiente penalización que esto

suponía para el SEO de nuestra página.

Afortunadamente, ahora tenemos una solución ideal para este problema: `@font-face`. Esta es una solución basada en CSS para incrustar fuentes en nuestras páginas. Además, está soportada por la mayoría de navegadores desde Internet Explorer 6 hacia arriba.

La propiedad `@font-face` permite a los diseñadores vincular un archivo que contiene una fuente específica para mostrar sus textos en la página. Ahora podemos incluir cualquier fuente que necesitemos con solo vincular el archivo adecuado.

Para incluir las fuentes mediante `@font-face`, haremos lo siguiente:

1. Cargaremos el archivo de fuente en nuestro sitio web en una variedad de formatos para que sea soportado por los diferentes navegadores.
2. Nombrar, describir y enlazar dicha fuente en una regla `@font-face`.
3. Incluir el nombre de la fuente en un valor de la propiedad `font-family`, tal como lo haríamos para las fuentes del sistema.

La sintaxis de la regla `@font-face` es la siguiente:

```
@font-face {  
    font-family: 'nombre de fuente';  
    src: 'archivo que contiene la fuente';  
}
```

He aquí un ejemplo de un bloque de `@font-face`:

```
@font-face {  
    font-family: 'MiNuevaFuente';  
    src: url('font.ttf');
```

}

Necesitaremos una regla `@font-face` para cada fuente que queramos incrustar en nuestra página. Además, también necesitaremos reglas diferentes si necesitamos variaciones de la misma fuente: `thin`, `thick`, `italic`, etc. Podemos declarar diferentes `src` para una misma regla `@font-face` separándolas por comas:

```
@font-face {  
    font-family: 'LeagueGothicRegular';  
    src: url('../fonts/League_Gothic-webfont.eot') format('eot'),  
        url('../fonts/League_Gothic-webfont.woff') format('woff'),  
        url('../fonts/League_Gothic-webfont.ttf') format('truetype'),  
        url('../fonts/League_Gothic-webfont.svg') format('svg');  
}
```

Si el navegador no reconoce el primer formato de fuente lo intentará con el segundo y así sucesivamente.

En el ejemplo anterior, hay cuatro formatos de fuentes enumeradas: la primera es una declaración de fuente EOT (el único tipo de archivo entendido por IE4-8), a continuación, se define el WOFF (Web Open Font Format, un estándar emergente), TTF (TrueType) y SVG (Scalable Vector Graphics).

Aunque la mayoría de los navegadores de escritorio usarán una de las tres primeras declaraciones, es recomendable incluir el formato SVG, que originalmente era el único formato soportado por el Iphone, aunque las nuevas versiones ya soportan las fuentes OTF (Open Type Font).

La siguiente tabla muestra un desglose del soporte de los navegadores para los diferentes formatos.

	IE	Safari	Chrome	Firefox	Opera	iOS
@font-face	4+	3.1+	4+	3.5+	10+	3.2+
WOFF	9+	6+	6+	3.6+	11.1+	
OTF		3.1+	4+	3.5+	10+	4.2+
TTF	9+?	3.1+	4+	3.5+	10+	4.2+
SVG		3.1+	4+		10+	3.2+
EOT	4+					

Como se puede observar, no hay un único formato que se admita en todos los navegadores, por lo que necesitaremos proporcionar la mayor cantidad de formatos posibles. También tenemos la posibilidad de indicar, opcionalmente, descriptores de propiedades en la definición de la fuente (`font-style`, `font-variant`, `font-weight` y otros) para que coincidan con la fuente que se está incrustando.

```
@font-face {  
  font-family: 'LeagueGothicRegular';  
  src: url('../fonts/League_Gothic-webfont.eot') format('eot'),  
       url('../fonts/League_Gothic-webfont.woff') format('woff'),  
       url('../fonts/League_Gothic-webfont.ttf') format('truetype'),  
       url('../fonts/League_Gothic-webfont.svg') format('svg');  
  font-weight: bold;  
  font-style: normal;  
}
```

Con esto no le estamos diciendo al navegador que ponga la fuente en negrita, lo que hacemos es indicar que la fuente que estamos incrustando es la

variedad negrita de la fuente. Por este motivo, si queremos incrustar diferentes variedades de la misma fuente, tendremos que crear una regla `@font-face` para cada una de ellas.

Veamos un ejemplo:

```
@font-face {  
    font-family: 'CoolFont';  
    font-style: normal;  
    src: url(fonts/CoolFontStd.ttf);  
}  
  
@font-face {  
    font-family: 'CoolFont';  
font-style: italic;src: url(fonts/CoolFontItalic.ttf);  
}  
  
.texto {  
    font-family: 'CoolFont';  
}
```

En el ejemplo, vemos que se han creado dos reglas `@font-face` con el mismo nombre, pero con diferente estilo. Cuando indicamos en la regla `.texto` la fuente `CoolFont` sin especificar estilo se utilizará la fuente `CoolFontStd.ttf`, pero si indicáramos el estilo `italic` se utilizaría la fuente `CoolFontItalic.ttf`. Si disponemos de una fuente en un determinado formato, podemos convertirla a otros formatos utilizando multitud de herramientas. Una posibilidad sería utilizar el conversor Font Squirrel que podemos encontrar en la web

<http://www.fontsquirrel.com/tools/webfont-generator>.

Una cosa importante a tener en cuenta es que las fuentes pueden tener licencias de uso, de forma que no deberíamos utilizar fuentes que no sean freeware si no tenemos permiso del autor.

Siguiendo con nuestro ejemplo del formulario de registro, vamos a incrustar dos nuevas fuentes:

```
@font-face {  
    font-family: 'LeagueGothic';  
    src: url('../fonts/League_Gothic-webfont.eot');  
    src: url('../fonts/League_Gothic-webfont.eot?') format('eot'),  
        url('../fonts/League_Gothic-webfont.woff') format('woff'),  
        url('../fonts/League_Gothic-webfont.ttf') format('truetype'),  
        url('../fonts/League_Gothic-webfont.svg') format('svg');  
    font-weight: bold;  
    font-style: normal;  
}  
  
@font-face {  
    font-family: 'AcknowledgementMedium';  
    src: url('../fonts/Acknowledgement-webfont.eot');  
    src: url('../fonts/Acknowledgement-webfont.eot?') format('eot'),  
        url('../fonts/Acknowledgement-webfont.woff') format('woff'),  
        url('../fonts/Acknowledgement-webfont.ttf') format('truetype'),  
        url('../fonts/Acknowledgement-webfont.svg') format('svg');  
    font-weight: normal;  
    font-style: normal;  
}
```



```
}
```

Utilizaremos la primera fuente en nuestros <label> y en el botón de envío:

```
label{  
    font-family: LeagueGothic, Tahoma, Geneva, sans-serif;  
    display: block;  
    margin: auto;  
    text-align: center;  
}
```

```
input[type=submit], button {  
    font-family: LeagueGothic, Tahoma, Geneva, sans-serif;  
    ...  
}
```

Y la segunda fuente en el título del formulario:

```
form h1 {  
    font-family: AcknowledgementMedium;  
    ...  
}
```

Nuestro formulario ahora se verá de la siguiente forma:



REGÍSTRATE!

Mi nombre es:

Mi email es:
(puedes introducir varios separados por comas)

Recuérdame en este equipo ☐

Mi dirección web es:
http://ejemplo.com

Me gustaría que mi password fuera:
(al menos 6 caracteres, sin espacios)

Mi password repetida es:

En una escala de 1 a 10, mis conocimientos de HTML5 son:

Por favor, inicia mi suscripción el día:
dd/mm/aaaa

Me gustaría recibir copias de *El manual de HTML5.*

También regístrate para *El manual de CSS3* ☐

ENVÍAME

Google Web Fonts

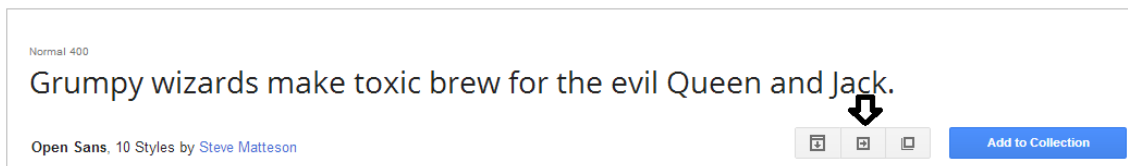
Otra alternativa a la hora de usar fuentes personalizadas en nuestros sitios web es el uso de un servicio que nos ofrece Google (Google Web Fonts). En lugar de descargarnos una fuente, convertirla a los formatos adecuados e

insertarla en nuestro sitio web, podemos utilizar una de las fuentes que nos ofrece Google de las que tiene alojadas en sus servidores.

Lo único que tendremos que hacer es buscar la fuente que nos interesa en el sitio web de Google Web Fonts (<https://www.google.com/fonts/>) y vincularla con nuestra página web.

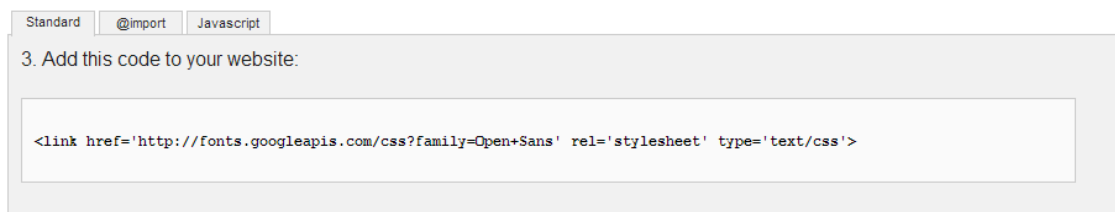
En esta página podremos realizar búsquedas de fuentes por nombre, por estilo, por categorías de fuentes (Serif, Sans Serif, Monospace, etc.), etc.

Una vez hemos localizado la fuente que nos interesa, Google nos indicará como usarla en nuestra web si pulsamos el botón indicado en la siguiente imagen:



Al pulsar este botón nos aparecen tres opciones a la hora de utilizar esta fuente:

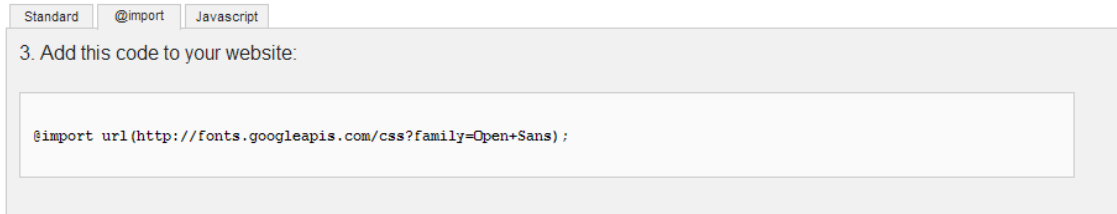
- Utilizando un elemento `<link>` dentro del `<head>` del documento html en el que queramos usar la fuente.



Este método tiene un inconveniente, ya que si queremos utilizar esta fuente en varios documentos html de nuestro sitio, tendremos que enlazarla con todos ellos.

- Importando la fuente a nuestra hoja de estilos, para lo cual utilizaremos el css-rule `@import`. Con este método podremos utilizar la fuente en

todas las páginas que tengan vinculada nuestra hoja de estilos, con lo que, no tendremos el inconveniente del anterior.



🚦 Google nos ofrece una tercera opción utilizando JavaScript, pero nos conformaremos con las dos anteriores.

Una vez que hemos vinculado la fuente con nuestra hoja de estilos podemos utilizarla en nuestras reglas de estilo de la misma forma que utilizamos cualquier otra fuente:

```
p { font-family: 'Open Sans'; }
```

5. Ejercicios

En este tema hemos hablado, principalmente, sobre formularios, por lo tanto, plantearemos a continuación una serie de ejercicios para practicar los nuevos conceptos trabajados.

Ejercicio 1

En este ejercicio vamos a implementar todas las funcionalidades explicadas sobre el formulario de registro que hemos trabajado a lo largo del tema. Las funcionalidades que debe incorporar son las siguientes:

- ✚ Los siguientes campos serán obligatorios: nombre, email, password1, password2, fechaInicio.
- ✚ El campo email será de tipo email y aceptará múltiples valores.
- ✚ Cuando el formulario se cargue, el campo nombre recibirá el foco automáticamente.
- ✚ El campo url será de tipo url y el texto de ejemplo que aparecerá será: `http://ejemplo.com`
- ✚ Tanto el password1 como el password2, tendrán que contener, al menos seis caracteres sin espacios en blanco.
- ✚ El campo conocimientos será de tipo range con un valor mínimo de 1 y un valor máximo de 10.
- ✚ El campo fechaInicio será de tipo date y el texto de ejemplo que aparecerá será: `dd/mm/aaaa`
- ✚ El campo cantidad será de tipo number con un valor por defecto de 1, un valor mínimo de 0 y un valor máximo de 5.
- ✚ El botón de enviar será un button en lugar de un submit.

🚦 Se realizarán las siguientes validaciones adicionales:

- Los campos password1 y password2 deberán ser iguales para que el formulario pueda ser enviado. Esta validación se realizará a medida que se vayan pulsando teclas en el campo password1 o en el campo password2, y el mensaje que se mostrará será el siguiente: "Las passwords deben coincidir".
- El campo fechaInicio, deberá contener una fecha mayor o igual que la actual. Para obtener las fechas en JavaScript haremos lo siguiente:

fechaInicio=document.getElementById("fechaInicio");

fechaActual = new Date();

fechaIni = fechaInicio.valueAsDate;

Una vez tenemos los objetos Date podemos compararlos como si fueran números. Esta validación la realizaremos al pulsar el botón enviar, antes de llamar al método submit.

- 🚦 Los campos obligatorios se marcarán con la imagen required.png.
- 🚦 Cuando el foco esté en un input que cumple la validación se marcará con la imagen valid.png.
- 🚦 Cuando el foco esté en un input que no cumple la validación se marcará con la imagen invalid.png.
- 🚦 El texto seleccionado aparecerá con las siguientes características:
 - Color de fondo: #484848
 - Color de fuente: #fff
 - Sombra del texto: ninguna.
- 🚦 El título del formulario tendrá dos imágenes de fondo, una a la izquierda

(bg-formtitle-left.png) y otra a la derecha (bg-formtitle-right.png).

Estas imágenes estarán redimensionadas a 100px de ancho y alto automático.

- 🎨 Los <label> y el botón del formulario tendrán la fuente LeagueGothic.
- 🎨 El título del formulario usará la fuente AcknowledgementMedium.
- 🎨 Ambas fuentes deben ser incrustadas a la hoja de estilos.

El formulario resultante se verá como se muestra en la imagen de la página 50. Las imágenes y las fuentes necesarias, las podéis encontrar dentro de las carpetas imgs y fonts del fichero de recursos. El ejercicio se guardará en una carpeta llamada registro que contendrá el documento registro.html, una carpeta css para los estilos, una carpeta imgs para las imágenes, una carpeta fonts para las fuentes y una carpeta scripts para el JavaScript.

Ejercicio 2

En este segundo ejercicio seguiremos trabajando sobre nuestra página web del curso. En concreto, vamos a mejorar algo el aspecto de la página cambiando algunas de las fuentes, ya que las que estamos utilizando hasta ahora son bastante sosas.

Para la cabecera de la página y para la barra de navegación usaremos una fuente llamada 'Luckiest Guy'. Para los contenidos de la página usaremos una fuente llamada 'Abel'. Ambas fuentes están disponibles en Google Fonts, sólo tenéis que buscarlas e incorporarlas a vuestra hoja de estilos. Esto lo haremos en el fichero estilos.css para que afecte a toda la página.

Al cambiar la fuente en la barra de navegación, el efecto que queda en

los textos hace que no se aprecie bien lo que hay escrito, por este motivo vamos a cambiar la sombra de los textos que pusimos en los enlaces de la barra de navegación. Pondremos una sombra con distancia horizontal y vertical de 2px y color #f06529 para los enlaces en estado normal, y una sombra con distancia horizontal y vertical de 0px, radio de difuminación de 5px y color #ff7e42, para los enlaces en estado :hover.

Una vez cambiadas las fuentes, la página principal quedará como aparece en la siguiente imagen:



Podéis ver esta imagen con mejor resolución dentro del fichero de recursos (resultado.jpg).

Ejercicio 3

Para terminar, vamos a implementar la sección de contacto de nuestra página, que contendrá un formulario. Este formulario lo crearemos en un documento que llamaremos contacto.html y cuyo resultado será el que se muestra en la siguiente imagen:



Podéis ver esta imagen con mejor resolución dentro del fichero de recursos (contacto.jpg). Lo primero que haremos será modificar las páginas index.html y videos.html, de forma que el enlace contacto apunte al documento contacto.html. Este documento será exactamente igual a los de las secciones de contenidos y vídeos, salvo dos excepciones: en la barra de navegación no habrá enlace en contacto y el contenido de la sección main será diferente.

Dentro de la sección main tendremos lo siguiente:

🚦 Un `<div>` que contendrá el formulario y que tendrá el id **contenedorFormulario**. Dentro del `<div>` tendremos lo siguiente:

- Un `<h1>` con el texto Contáctenos.
- Un formulario con los siguientes campos:
 1. Un campo para solicitar el nombre del usuario que quiere contactar. Este campo será obligatorio y contendrá el texto de ejemplo siguiente: "Escriba su nombre".
 2. Un campo para solicitar el correo electrónico del usuario. Este campo será obligatorio y contendrá el texto de ejemplo siguiente: "Escriba su email".
 3. Un `<textarea>` para que el usuario escriba el mensaje que quiere enviar. Este campo será obligatorio y contendrá el texto de ejemplo siguiente: "Su mensaje...".
 4. Un botón submit para enviar los datos del formulario.

Para que el formulario tome el aspecto que se muestra en la imagen, tendremos que aplicar una serie de estilos que crearemos en un fichero llamado `contacto.css` dentro de la carpeta `css`. Dentro de este fichero crearemos los siguientes estilos:

🚦 Sección main:

- Pondremos la imagen de fondo `fondoForm.jpg`, indicando que la imagen abarque todo el espacio de la sección.
- Sus contenidos estarán centrados.

🚦 Contenedor del formulario:

- La posición será relativa.

- Se mostrará como inline-block.
- Márgenes de 50px.
- Ancho 600px.
- Margen interno superior e inferior 20px, izquierdo y derecho 50px.
- Color de fondo rgb: 200, 200, 200 con una opacidad de 0.7.
- Contenidos alineados a la izquierda.
- Sombra de la caja con los siguientes valores: desplazamiento horizontal 3px, desplazamiento vertical 0px, distancia de difuminación 8px y color rgb: 0, 0, 0 con una opacidad de 0.4.

🚦 Título del formulario:

- Posición absoluta con una distancia al borde superior de -16px y al borde izquierdo de 490px.
- Color de fondo rgb: 255, 255, 255 con una opacidad de 0.7.
- Fuente *Aguafina Script* en cursiva y color #333. Esta fuente es una fuente de google, por lo que, debemos buscarla y enlazarla correctamente mediante un @import.
- Espacio entre letras 5px.
- Sombra del texto con distancia horizontal y vertical de 0px, radio de difuminación de 2px y color #777.
- Márgenes internos superior e inferior de 0px e izquierdo y derecho de 10px.

🚦 Estilos comunes para todos los campos del formulario, excepto el botón:

- Se mostrarán como inline-block.
- Borde de 1px, estilo sólido y color #e5e5e5.

- Tendrán dos sombras de caja³:
 1. La primera con las siguientes características: Desplazamiento horizontal 0px, desplazamiento vertical 1px, distancia de difuminación 4px y color rgb: 0, 0, 0 con una opacidad de 0.3.
 2. La segunda con las siguientes características: Desplazamiento horizontal 0px, desplazamiento vertical 0px, distancia de difuminación 40px y color rgb: 0, 0, 0 con una opacidad de 0.1. Esta sombra será interna (inset).
 - Color de fuente negro.
 - Fuente en negrita.
 - Márgenes internos superior e inferior de 1px, izquierdo 45px y derecho de 4px.
 - Margen inferior 10px.
 - Configuraremos una transición para el cambio de todas las propiedades de estos elementos de la siguiente forma: duración 0.4s, propiedad all, efecto ease-in.
- 🎨 Estilos comunes para los campos nombre y correo:
- Alto 32px.

³ Para insertar más de una sombra de caja al mismo elemento, basta con que pongamos una a continuación de otra separándolas con comas. Por ejemplo: `box-shadow: 1px 1px 5px #cecece, 0px 0px 40px #fefefe;`

- Ancho 330px.
- Al obtener el foco estos campos, se cambiará el ancho a 370px y la sombra de la caja que tendrá las siguientes características: Desplazamiento horizontal 0px, desplazamiento vertical 0px, distancia de difuminación 12px y color rgb: 255, 255, 255. (Como habíamos creado una transición para todos los campos, estos cambios serán animados).
- El campo nombre tendrá la imagen de fondo icon-nombre.png y el campo correo icon-correo.png. No se repetirán y estarán colocadas a una distancia de 10px del borde izquierdo.
- Al obtener el foco en uno de los dos campos pueden pasar dos cosas:
 - Que la validación del campo sea correcta, con lo que, además de la imagen indicada anteriormente, tendrá también la imagen valid.png como fondo, tampoco se repetirá y estará colocada a 390px del borde izquierdo.
 - Que la validación del campo no sea correcta, con lo que, además de la imagen indicada anteriormente, tendrá también la imagen invalid.png como fondo, tampoco se repetirá y estará colocada a 390px del borde izquierdo.
- Si nos fijamos en la imagen, tanto el campo nombre como el campo correo tienen un asterisco '*' a la derecha que indica que ambos

campos son requeridos. Para hacer esto haremos lo siguiente:

- En el documento html añadiremos un `<label>` vacío a continuación de cada `<input>`.
- En la hoja de estilos indicaremos una regla para seleccionar los input que sean requeridos y tengan un hermano adyacente de tipo `<label>`. En esta regla indicaremos que se añada después del elemento (`::after`) el contenido `' *'` (`content: ' *'`).

Campo mensaje:

- Fuente arial.
- Tamaño de fuente 1em.
- Ancho 90%.
- Alto 150px.
- Imagen de fondo icon-mensaje.png. No se repetirá y estará colocada a una distancia de 10px del borde izquierdo.
- Al obtener el foco se cambiará la sombra de la caja como sigue: desplazamiento horizontal 0px, desplazamiento vertical 0px, distancia de difuminación 12px y color rgb: 255, 255, 255. (Como habíamos creado una transición para todos los campos, este cambio será animado).

Botón enviar:

- Márgenes superior e inferior de 10px, izquierdo y derecho de 0px.
- Márgenes internos de 10px.
- Borde de 1px, estilo sólido y color #434343.

- Color de fuente blanco.
- Como fondo le pondremos un gradiente de arriba abajo iniciando en el color #5e5e5e y acabando en el color #2a2a2a.
- La sombra de la caja será así: desplazamiento horizontal 0px,
- desplazamiento vertical 1px, distancia de difuminación 1px, color #353535 y tipo de sombra inset.
- Cursor del ratón pointer.
- Al poner el ratón sobre el botón se pondrá el mismo gradiente de fondo, pero al contrario, es decir, en lugar de arriba abajo, de abajo arriba.

Además de todo esto, el formulario tiene que validarse de la siguiente forma:

- 🚦 El campo nombre, tiene que ser uno de los siguientes: alex, juan, ana o pedro. Para ayudar a la entrada de usuario crearemos un <datalist> que contendrá estos nombres y asociaremos al campo. A medida que el usuario vaya pulsando teclas estando en el campo, se validará si el usuario que hay en el campo es uno de los válidos. Si al pulsar enviar el usuario no es válido se mostrará el mensaje de error "El usuario introducido no existe".
- 🚦 El campo correo debe contener una dirección de correo de un dominio español, es decir, debe acabar en ".es". A medida que el usuario vaya pulsando teclas estando en el campo, se validará si el correo que hay en el campo es válido. Si al pulsar enviar el correo no es válido se mostrará el mensaje de error "Debes introducir una dirección de correo española (acabada en .es)".

Estas validaciones las realizaremos en un fichero JavaScript llamado `contacto.js`. Todos los ficheros (imágenes, fuentes, etc.) necesarios para la realización del ejercicio, los podéis encontrar en el fichero de recursos. Con esto habremos terminado los ejercicios del tema. Recuerda que para realizar la entrega del ejercicio, debes comprimir las carpetas `curso` y `registro` dentro de un fichero llamado:

`nombre-alumno-tema4.zip`

No olvidéis pasar el validador tanto a los `html` como a los `css` antes de efectuar la entrega.