5 Interacción con el usuario. Eventos y formularios

OBJETIVOS DEL CAPÍTULO

- ✓ Reconocer las posibilidades de los lenguajes de marcas de capturar y gestionar los eventos producidos.
- ✓ Diferenciar los tipos de eventos que se pueden manejar.
- ✓ Crear código que capture y utilice eventos.
- ✓ Reconocer las capacidades del lenguaje relativas a la gestión de formularios web.
- ✓ Validar formularios web utilizando eventos y expresiones regulares para facilitar los procedimientos.
- ✓ Probar y documentar el código.

La interacción con el usuario, es la relación que establece el usuario con la aplicación web. Para ello en el lado del usuario se utilizan mecanismo de entrada como el teclado o el ratón. En el lado de la maquina se muestra la información a tratar, como por ejemplo un formulario web. El gran número de combinaciones posibles que surgen a la hora de actuar sobre una aplicación web, complica la forma en la que el código debe afrontar estas posibilidades. Los formularios contienen campos en ocasiones muy genéricos que deben ser adaptados al fin del campo en el formulario. Para manejar estas posibilidades existen los eventos que materializan los cambios que el usuario ocasiona en una aplicación web, asociando a los distintos objetos que nos vamos a encontrar una serie de posibles cambios que puede suceder sobre el objeto. A estos cambios se les llama *eventos*.

A continuación vamos a describir lo que es un evento y un formulario para poder asociar los conceptos de programación web a la funcionalidad que llevan asociada:

- **Evento**. Según la RAE un evento es una eventualidad, hecho imprevisto, o que puede acaecer. Cuando hablamos de una aplicación web, un evento es cualquier suceso relacionado con la aplicación web. Por lo tanto, un evento podría ser cerrar la ventana, dar un clic en un botón de radio o situarse con el ratón sobre la página. Es importante tener en cuenta que el conjunto de eventos de una aplicación web son los que pertenecen a la ventana de navegación donde se maneje esta aplicación, pudiéndose extender estos a un *popup* que se abra desde esta ventana.
- **Formulario**. Un formulario web, alojado en una página web, es una agrupación de objetos que tienen una función concreta. Estos objetos permiten al usuario introducir datos para enviarlos a un servidor y que sean procesados.

Figura 5.1. *Vista de un formulario dentro de un navegador*

Además del formulario mostrado en la imagen anterior, existen métodos para recuperar los campos en el lado del servidor. La página de destino del formulario será la encargada de recuperar estos datos.

5.1 MODELO DE GESTIÓN DE EVENTOS

Como hemos visto anteriormente, los eventos son mecanismos que se accionan cuando el usuario realiza un cambio sobre una página web. En una página web existen multitud de elementos. Estos elementos son los objetos que modelan la página web. El encargado de crear la jerarquía de objetos que compone una página web es el DOM. Por lo tanto, es el DOM (*Document Objet Model*) el encargado de gestionar los eventos.

Los eventos se sitúan en componentes de la pagina HTML, si hacemos un clic en un componente de la pagina, podríamos desencadenar una acción. Por un lado deberíamos marcar en el componente cual es el evento que queremos que desencadene la acción. Para poder controlar un evento necesitamos un manejador. Un manejador es una palabra reservada que indica la acción que va a manejar. En el caso del evento *click*, el manejador sería `onClick`. En este caso podemos asociar a un componente el manejador y desencadenar una acción cuando se haga un clic sobre ese componente.

Cuando ha sucedido el evento y este tiene un manejador asociado necesitamos la ayuda de un lenguaje de *script* como JavaScript o Visual Script. El lenguaje más usado es JavaScript. Este es un lenguaje interpretado que tiene la capacidad de actuar sobre los objetos DOM sin que enviemos una nueva petición al servidor. Es decir JavaScript puede modificar las propiedades de los objetos de una página en el mismo navegador.

A continuación vamos a ver un ejemplo de cómo JavaScript muestra un mensaje de alerta cuando hacemos clic sobre una imagen existente en la página web:

```
<IMG SRC="mundo.jpg" onclick="alert('Click en imagen');">
```

En el código anterior vemos que invocamos directamente a la función `alert()` de JavaScript. Si la acción a desencadenar es simple la podemos incluir directamente, aunque normalmente se realiza una llamada a una función. Será esta función la encargada de desencadenar la acción deseada. De esta forma estructuramos mejor el código y evitamos repetir sentencias si vamos a hacer varias veces una llamada para replicar un mismo comportamiento, pudiendo reutilizar una función en otra página si es preciso. A continuación mostramos la misma acción integrada en una página HTML llamando a una función:

```
<html>
  <head>
    <title>Pagina de Evento</title>
    <script>
      function funcl() {
        alert("Click en imagen");
      }
    </script>
  </head>
  <body>
    <IMG SRC="mundo.jpg" onclick="funcl();">
  </body>
</html>
```

En el código superior vemos como al el manejador `onclick` llama a la función, `func1` para que se ejecute. En la parte superior del código, dentro del `head` y entre las etiquetas `<script>...</script>` es donde debemos definir la función `func1` para que muestre la alerta cuando esta función sea llamada.

A continuación vamos a ver que la especificación de DOM define grupos de eventos dividiéndolos según su origen, estos son: eventos del ratón, del teclado, HTML y de DOM:

- **Eventos del ratón**. Estos eventos se originan cuando el usuario hace uso del ratón para realizar una acción. Cualquier movimiento de la flecha del ratón o acción sobre algunos de sus botones, puede desencadenar un evento. Cabe decir que no es necesario que la acción del ratón se realice sobre ningún objeto que interactúe con el usuario como puede ser un botón de radio o un *check*.
- **Eventos del teclado**. Estos eventos se originan cuando el usuario pulsa alguna tecla del teclado.
- **Eventos HTML**. Estos eventos se producen cuando hay algún cambio en la página del navegador. También pueden ocurrir cuando existe alguna interacción entre el cliente y el servidor.
- **Eventos DOM**. Los eventos DOM, o eventos de mutación, son los que se originan cuando existe algún cambio en la estructura DOM de la página.

5.1.1 EVENTOS DEL RATÓN

Los eventos del ratón son los más usuales puesto que la mayor parte de las acciones en una aplicación web las realizamos a través de este. A continuación vamos a ver cuáles son los eventos que define la especificación DOM:

- **Click**. Este evento se produce cuando pulsamos sobre el botón izquierdo del ratón. El manejador de este evento es `onclick`.
- **Dblclick**. Este evento se acciona cuando hacemos un doble clic sobre el botón izquierdo del ratón. El manejador de este evento es `ondblclick`.
- **Mousedown**. Este evento se produce cuando pulsamos un botón del ratón. El manejador de este evento es `onmousedown`.
- **Mouseout**. Este evento se produce cuando el puntero del ratón esta dentro de un elemento y este puntero es desplazado fuera del elemento. El manejador de este evento es `onmouseout`.
- **Mouseover**. Este evento al revés que el anterior se produce cuando el puntero del ratón se encuentra fuera de un elemento y éste se desplaza hacia el interior. El manejador de este evento es `onmouseover`.
- **Mouseup**. Este evento se produce cuando soltamos un botón del ratón que previamente teníamos pulsado. El manejador de este evento es `onmouseup`.
- **Mousemove**. Se produce cuando el puntero del ratón se encuentra dentro de un elemento. Es importante señalar que este evento se producirá continuamente una vez tras otra mientras el puntero del ratón permanezca dentro del elemento. El manejador de este evento es `onmousemove`.

Es importante destacar que el orden de ejecución de los eventos es el siguiente: *mousedown*, *mouseup*, *click*, *mousedown*, *mouseup*, *click*, *dblclick*. Siendo el último evento de la secuencia el doble clic.

out - fuera
over - encima

Cabe señalar que todos los elementos de las páginas web soportan los eventos del ratón, pero no todos los eventos que existen son soportados por todos los elementos de las páginas.

Cuando se produce un evento el objeto `event` se crea automáticamente. Este objeto contiene características adicionales sobre el evento que se ha producido. Estos datos pueden ser, la posición del ratón, el elemento que ha producido el evento, etc.

Para los eventos de ratón, el objeto `event` tiene una serie de propiedades, algunas de ellas son; las coordenadas del ratón (`screenX`, `screenY`), el nombre del evento (`type`), el elemento que origina el evento (`button`), etc.

Algunos navegadores como Internet Explorer permiten acceder al objeto `event` a través del objeto `Windows`. La especificación DOM, en cambio, indica que el único parámetro que se debe pasar a las funciones es el objeto `event`. A continuación mostramos cómo recuperar el parámetro del *array* en el código:

```
func1(){
  var elEvento = arguments[0];
}
```

No es necesario crear una variable que recupere el nombre del *array*, lo podemos indicar de forma explícita, como vemos en el siguiente código:

```
func1(elEvento) {
  ...
}
```

Puede parecer mágico que en la declaración de la función indiquemos un parámetro y en realidad no le pasemos ningún parámetro a la función. Los navegadores que siguen los estándares, crean ese parámetro y se lo pasan automáticamente a la función encargada de manejar ese evento.

5.1.2 EVENTOS DEL TECLADO

Los eventos de teclado son los que suceden cuando pulsamos una tecla. Según la especificación DOM existen los siguientes eventos relacionados con la actividad del teclado.

- **Keydown**. Este evento se produce cuando pulsamos una tecla del teclado. Si mantenemos pulsada una tecla de forma continua, el evento se produce una y otra vez hasta que soltemos la misma. El manejador de este evento es `onkeydown`.
- **Keypress**. Este evento se produce si pulsamos una tecla de un carácter alfanumérico (el evento no se produce si pulsamos **Enter**, la barra espaciadora, etc.). En el caso de mantener una tecla pulsada, el evento se produce de forma continuada. El manejador de este evento es `onkeypress`.
- **Keyup**. Este evento se produce cuando soltamos una tecla. El manejador de este evento es `onkeyup`.

Las propiedades de `event` para los eventos de teclado son: el código numérico de la tecla pulsada (`keyCode`), el código unicode del carácter correspondiente a la tecla pulsada (`charCode`), el elemento que origina el evento (`target`) y otras para identificar si hemos pulsado **shift** (`shiftKey`), **control** (`ctrlKey`), **alt** (`altKey`) o **meta** (`metaKey`).

Es importante destacar que el orden de secuencia de las teclas no es el mismo para un tipo de teclas que para otras. A continuación veremos cuál es el orden:

- Cuando pulsamos una tecla que corresponda a un carácter alfanumérico, la secuencia de eventos es la siguiente: *keydown*, *keypress*, *keyup*.
- En el caso de pulsar una tecla que no corresponda a un carácter alfanumérico, la secuencia de eventos es: *keydown*, *keyup*.
- Además existe la posibilidad de que dejemos una tecla pulsada. Para el primer caso, se repiten de forma continua los eventos *keydown*, *keypress*. En el segundo caso se repite de forma continuada el evento *keydown* solamente.

5.1.3 EVENTO HTML

Los eventos HTML, como hemos visto anteriormente, son los que actúan cuando hay cambios en la ventana del navegador. También se producen cuando hay ciertas interacciones entre cliente y el servidor. A continuación vamos describir cuáles son estos eventos y cuándo se acciona cada uno de ellos:

- **Load**. El evento *load* hace referencia a la carga de distintas partes de la página. Este se produce en el objeto `Window` cuando la página se ha cargado por completo. En el elemento `<img>` actúa cuando la imagen se ha cargado. En el elemento `<object>` se acciona al cargar el objeto completo. El manejador es `onload`.
- **Unload**. El objeto *unload* actúa sobre el objeto `Window` cuando la pagina ha desaparecido por completo (por ejemplo, si pulsamos el aspa cerrando la ventana del navegador). También se acciona en el elemento `<object>` cuando desaparece el objeto. El manejador es `onunload`.
- **Abort**. Este evento se produce cuando el usuario detiene la descarga de un elemento antes de que haya terminado, actúa sobre un elemento `<object>`. El manejador es `onabort`.
- **Error**. El evento *error* se produce en el objeto `Window` cuando se ha producido un error en JavaScritp. En el elemento `<img>` cuando la imagen no se ha podido cargar por completo y en el elemento `<object>` en el caso de que un elemento no se haya cargado correctamente. El manejador es `onerror`.
- **Select**. Se acciona cuando seleccionamos texto de los cuadros de textos `<input>` y `<textarea>`. El manejador es `onselect`.
- **Change**. Este evento se produce cuando los cuadros de texto `<input>` y `<textarea>` pierden el foco y el contenido que tenían ha variado. También se producen cuando un elemento `<select>` cambia de valor. El manejador es `onchange`.
- **Submit**. Este evento se produce cuando pulsamos sobre un botón de tipo `submit`. El manejador es `onsubmit`.
- **Reset**. Este evento se produce cuando pulsamos sobre un botón de tipo `reset`. El manejador es `onreset`.
- **Resize**. Este evento se produce cuando redimensionamos el navegador, actúa sobre el objeto `Window`. El manejador es `onresize`.
- **Scroll**. Se produce cuando varía la posición de la barra de desplazamiento (*scroll*) en cualquier elemento que la tenga. El manejador es `onscroll`.

- **Focus**. Este evento se produce cuando un elemento obtiene el foco. El manejador es `onfocus`.
- **Blur**. Este evento se produce cuando un elemento pierde el foco. El manejador es `onblur`.

Es importante indicar que el objeto *load* indica que la pagina se ha cargado entera. La especificación DOM requiere que la página y el árbol DOM se hayan cargado por completo. Por ello, este evento es muy importante y muy utilizado.

5.1.4 EVENTO DOM

Estos eventos hacen referencia a la especificación DOM. Se accionan cuando varía el árbol DOM. Estos eventos no están implementados en todos los navegadores, por tanto, habría que comprobar que el navegador soporta estos eventos, aun sabiendo que el navegador cumple con la especificación DOM.

- **DOMSubtreeModified**. Este evento se produce cuando añadimos o eliminamos nodos en el subárbol de un elemento o documento.
- **DOMNodeInserted**. Este evento se produce cuando añadimos un nodo hijo a un nodo padre.
- **DOMNodeRemoved**. Este evento se produce cuando eliminamos un nodo que tiene nodo padre.
- **DOMNodeRemovedFromDocument**. Este evento se produce cuando eliminamos un nodo del documento.
- **DOMNodeInsertedIntoDocument**. Este evento se produce cuando añadimos un nodo al documento.

5.2 UTILIZACIÓN DE FORMULARIOS DESDE CÓDIGO

Un formulario web es un utensilio que sirve para enviar, tratar y recuperar datos que son enviados y recibidos entre un cliente (navegador) y un servidor web. Cada elemento del formulario sirve para almacenar un tipo de dato o para accionar las distintas funcionalidades que tiene el formulario.

Los formularios disponen de una arquitectura, que habitualmente va ligada a otra más general. En nuestro caso los formularios están enmarcados en un lenguaje de marcado llamado HTML.

HTML es un lenguaje de marcado que estructura los contenidos de las páginas web. Los formularios están integrados en este lenguaje de programación y su función es, hacer que los usuarios interactúen con los datos de las páginas web, convirtiendo una página web en una aplicación web. A continuación vamos a ver cómo funciona un formulario web, y como se comunica el cliente con el servidor.

5.2.1 ESTRUCTURA DE UN FORMULARIO

Un formulario está compuesto por muchos elementos, pero para generar una estructura básica de un formulario, basta con utilizar dos de estos elementos. Al igual que HTML los formularios se definen con etiquetas. Dentro de cada etiqueta también podemos acceder a los atributo de la misma. La etiqueta principal de un formulario es `<form>`, esta

En el código anterior vemos como dentro del `body` de una página HTML introducimos una estructura `form`. En el `action` enviamos los datos del formulario (los datos corresponderían a los puntos suspensivos, los veremos en el siguiente punto) a la dirección, *www.web.es/formulario.php*, para que sean procesados en el servidor. En este caso los datos serán enviados de forma oculta puesto que hemos utilizado el método *POST*.

5.2.2 ELEMENTOS DE UN FORMULARIO

Según hemos visto en el punto anterior, un formulario sirve para enviar datos, pero... ¿cómo vamos a enviar esos datos? Y, además, ¿cómo vamos a indicar cuando queremos enviar esos datos? Para poder definir los datos que queremos enviar y cuando los vamos a enviar, tenemos los elementos del formulario. El elemento principal del formulario se denomina con la etiqueta `<input>`. Este elemento tiene una serie de atributos que modifican su funcionalidad. Los tipos de *input* según su funcionalidad se llaman *controles de formulario* y *campos de formulario*. Estos son los encargados de guardar los datos que vamos a enviar en el formulario. Los hay de varios tipos y a continuación vamos a definir cómo guardan y procesan los datos cada uno de ellos. También se encargan de accionar sucesos, por lo general son botones y al ser pulsados realizan la acción para la que están encomendados.

La etiqueta `input` tiene una lista de atributos muy extensa. A continuación vamos a definir cuáles son estos atributos, algunos atributos como el `type`, convierten al *input* en un elemento diferente generando ciertas peculiaridades con respecto a otros tipos de `input`.

5.2.3 ESTRUCTURA DE UNA ETIQUETA INPUT

El formato por defecto de una etiqueta `input` es el siguiente `<input ... />`, como es una etiqueta unaria se cierra a sí misma con la barra (`/`) al final. Esta etiqueta se transforma en distintos elementos que a su vez almacenan un tipo de dato. Para definir el tipo de elemento, la etiqueta `input` se sirve del atributo `type`. De esta forma si igualamos uno de los tipos validos al atributo `type`, modelamos el *input* al tipo asignado. A continuación vamos a ver el atributo `type` y el resto de atributos que puede tener una etiqueta `input`:

- **Type**. Es el que indica el tipo de elemento que vamos a definir. De él dependen el resto de parámetros puesto que el elemento cambia según sea de un tipo o de otro. Los valores posibles que acepta el atributo `type` son; `text` (cuadro de texto), `password` (cuadro de contraseña, los caracteres aparece ocultos tras asteriscos), `checkbox` (casilla de verificación), `radio` (opción de entre dos o más), `submit` (botón de envío del formulario), `reset` (botón de vaciado de campos), `file` (botón para buscar ficheros), `hidden` (campo oculto para, el usuario no lo visualiza en el formulario), `image` (botón de imagen en el formulario), `button` (botón del formulario). Debido a las peculiaridades de cada uno de los tipos procederemos a explicar detalladamente cada tipo en el punto siguiente, indagando en cómo afectan al resto de atributos.
- **Name**. El atributo `name` asigna un nombre al elemento. Si no le asignamos nombre a un elemento, el servidor no podrá identificarlo y, por tanto, no podrá tener acceso al elemento.
- **Value**. El atributo `value` inicializa el valor del elemento. Los valores dependerán del tipo de dato, en ocasiones los posibles valores a tomar serán verdadero o falso.
- **Size**. Este atributo asigna el tamaño inicial del elemento. El tamaño se indica en pixeles. En los campos `text` y `password` hace referencia al número de caracteres.

- **Maxlength**. Este atributo indica el número máximo de caracteres que pueden contener los elementos `text` y `password`. Es conveniente saber que el tamaño de los campos `text` y `password` es independiente del número de caracteres que acepta el campo.
- **Checked**. Este atributo es exclusivo de los elementos `checkbox` y `radio`. En el definimos que opción por defecto queremos seleccionar.
- **Disable**. Este atributo hace que el elemento aparezca deshabilitado. En este caso el dato no se envía al servidor.
- **Readonly**. Este atributo sirve para bloquear el contenido del control, por tanto, el valor del elemento no se podrá modificar.
- **Src**. Este atributo es exclusivo para asignar una URL a una imagen que ha sido establecida como botón del formulario.
- **Alt**. El atributo `alt` incluye una pequeña descripción del elemento. Habitualmente, y si no lo hemos desactivado cuando posicionamos el ratón (sin pulsar ningún botón) encima del elemento, podemos visualizar la descripción del mismo.

5.2.4 TIPOS DE INPUT

Como ya hemos visto en el punto anterior, la mutación que sufre cada tipo de *input* hace que sean elementos diferentes dentro del formulario. Por esta razón el estudio detallado de cada tipo, es necesario, para comprender las opciones que nos brindan los formularios. El objetivo de los tipos es adecuar los datos para facilitar la recopilación, manejo y envío de datos a través del formulario. También para la recuperación y comprensión de los mismos cuando el usuario recibe un formulario del servidor. A continuación vamos a detallar cada uno de estos tipos viendo un ejemplo de cada uno de ellos.

- **Cuadro de texto**. El cuadro de texto muestra un cuadro de texto vacío en el que el usuario puede introducir un texto. Este es uno de los elementos más usados. La forma de indicar que es un campo de texto es la siguiente `type="text"` La visualización en la página es la siguiente:

Nombre

Figura 5.3. Muestra de un cuadro de texto antecedido del texto nombre

Para que el cuadro de texto pueda ser accedido deberá tener inicializado el atributo `name`, `name="nombreCuadro"`. El valor que indicamos en el atributo `name` es el que emplea el servidor para identificar este campo una vez se ha realizado el envío del formulario. El valor correspondiente a `name` deberá corresponder a un valor de variable como en cualquier otro lenguaje de programación. Respetando el no poner espacios en blanco, etc. El atributo `value` inicializará con un valor el cuadro de texto, `value="Javier"`. En el caso de que no mostremos el tamaño, el navegador mostrará el cuadro de texto con un tamaño predeterminado. De esta forma adaptaremos el tamaño del cuadro de texto al tipo de dato que va a incluir. Con el atributo `maxlength` podemos limitar el número de caracteres que podrá incluir el usuario, así, realizamos una primera limitación a la hora de validar el formulario. Es muy útil a la hora de indicar que un dato como, por ejemplo, un teléfono, no permita introducir más de 9 caracteres. El atributo `readonly` permite que en el formulario visualicemos el

cuadro de texto, pero no nos permite modificarlo. Este atributo es útil cuando realizamos una consulta de datos que residen en el servidor pero existen datos que no podemos modificar, por ejemplo el nombre de usuario a la hora de modificar el resto de datos. `Disabled` como el atributo anterior no permite modificar el campo de texto y además no envía los datos al servidor. A continuación vamos a ver cómo quedaría el código de una etiqueta `input` con los atributos anteriores:

```
<input t,gth="10" readonly />
```

En el código anterior vemos un *input* de tipo `text`, que se llama `nombre`, esta inicializado con la cadena de caracteres `"Javier"` y no se puede modificar por el usuario en el formulario.

- **Cuadro de contraseña**. El cuadro de contraseña es como el cuadro de texto, con la diferencia de que en el de contraseña, los caracteres que escribe el usuario no se ven en pantalla. En su lugar los navegadores muestran asteriscos o puntos con el fin de orientar al usuario del número de caracteres que va escribiendo. Su utilidad es escribir datos privados del usuario, como contraseñas u otros datos sensibles.

Constraseña ••••••••

Figura 5.4. *Cuadro de contraseña en un formulario*

```
<input type="password" name="contrasenia" />
```

En el código anterior mostramos un *input* de tipo `password` que se llama `contrasenia`.

Los atributos indicados en el cuadro de texto pueden ser utilizados de igual forma en el cuadro de contraseña.

- **Casilla de verificación**. Estos elementos permiten al usuario activar o desactivar la selección de cada una de las casillas de forma individual. Su uso habitual es para indicar afirmación o negación sobre la casilla a la que se refiera.

Colores favoritos
☐ Rojo
☐ Azul
☐ Verde

checkbox

Figura 5.5. *Casilla de verificación de un formulario*

```
<p>Colores favoritos</p>
</br><input name="rojo" type="checkbox" value="ro"/> Rojo
</br><input name="azul" type="checkbox" value="az"/> Azul
</br><input name="verde" type="checkbox" value="ve"/> Verde
```

En el código anterior mostramos los 3 *input* de tipo `checkbox`, estos se llaman `rojo`, `azul` y `verde`, los valores que enviarán al servidor en caso de ser verificados son `ro`, `az`, y `ve`, respectivamente.

En el caso de querer mostrar que uno de los *input* de tipo `checkbox` este seleccionado por defecto lo tendríamos que hacer con el atributo `checked`. A continuación mostramos un ejemplo del código correspondiente:

```
<input name="rojo" type="checkbox" value="ro" checked /> Rojo
```

- **Opción de radio**. Este tipo de elemento agrupan una serie de opciones excluyentes entre sí. De esta forma el usuario solo puede coger una opción de entre todas las que tiene establecidas un grupo de botones radio. Al seleccionar una opción el usuario, automáticamente se deselecciona la que estaba seleccionada.

Género
Hombre radio
Mujer

Figura 5.6. Opción de radio de un formulario

A continuación mostramos el código correspondiente la opción de radio que hemos mostrado anteriormente:

```
Género
</br><input type="radio" name="género" value="M"> Hombre
</br><input type="radio" name="género" value="F"> Mujer
```

En el código anterior observamos que el nombre de los *input* es el mismo. El nombre de los elementos que pertenezcan a un mismo grupo de opciones de radio deberá ser el mismo.

Es posible fijar un valor inicial con el atributo `checked`. En ese caso el *input* que deseemos que quede seleccionado por defecto deberá llevar dentro el atributo `checked`, `checked="checked"`.

- **Botón de envío**. Este elemento es el encargado de enviar los datos del formulario al servidor. En este caso el `type` toma el valor `submit`. El valor del atributo `value` se mostrará en este caso en el botón generado.

Enviar

Figura 5.7. Botón de envío de un formulario

```
<input type="submit" name="enviar" value="Enviar">
```

En el código anterior inicializamos el nombre del botón con la cadena `"Enviar"`. El botón enviará los datos del formulario a la dirección que esté dispuesta en el `action`.

- **Botón de reset**. Este elemento es un botón que establece el formulario a su estado original:

Restablecer

Figura 5.8. Botón de restablecimiento de un formulario

En este botón no hemos añadido el atributo `value`. Es el navegador el encargado de asignar un valor por defecto, el valor por defecto en el caso anterior es `"Restablecer"`.

- **Ficheros adjuntos**. Este tipo de *input* permite adjuntar ficheros adjuntos. Las limitaciones sobre cuántos ficheros y el tamaño no están definidas por el elemento. Por lo tanto, deberá controlarse desde otra parte del código para evitar desbordamientos o colapsos en el servidor. El elemento añade de forma automática un cuadro de texto que se dispondrá para almacenar la dirección del fichero adjunto seleccionado. Pulsando un botón

situado a la derecha del cuadro de texto navegaremos por el árbol de directorios en busca del fichero a adjuntar y una vez seleccionado, éste se incluirá en el cuadro que hemos indicado antes.

Fichero adjunto [ ] Examinar...

Figura 5.9. *Opción añadir ficheros adjuntos de un formulario*

```
Fichero adjunto
<input type="file" name="fichero"/>
```

Vemos como en el código anterior no es necesario incluir nada más que el tipo de *input* como `file` para que en la Figura 5.9 nos muestre el campo de texto y el botón de examinar.

Es importante saber que para adjuntar archivos debemos indicar el tipo de envío que vamos a realizar. Esto lo indicamos en la estructura del `form`, inicializando el atributo `enctype`. Habitualmente el valor que toma este atributo es `multipart/form-data`. Por lo tanto, el código en la cabecera del formulario quedaría de la siguiente manera.

```
<form action="…" method="post" enctype="multipart/form-data">
  ...
</form>
```

- **Campos ocultos**. Los campos ocultos no son visibles en el formulario por el usuario. Estos elementos son útiles para enviar información de forma oculta que no tenga que ser tratada por el usuario. Son usadas a menudo para que el servidor pueda tratar la información o hacer alguna acción determinada.

```
<input type="hidden" name="campoOculto" value="cambiar"/>
```

En el código anterior vemos como el campo con nombre `campoOculto`, toma el valor `cambiar`. Esto podría usarse para que el servidor realice un cambio con respecto a algo.

- **Botón de imagen**. Este elemento es una personalización de un botón, cambiando el aspecto por defecto que tienen los botones de un formulario por una imagen. El aspecto del botón sería el de la imagen a la que hayamos hecho referencia.

```
<input type="image" name="enviar" src="imagen_mundo.jpg"/>
```

El elemento botón de imagen se tipa igualando a `image` el tipo de *input* como vemos en el código anterior.

- **Botón**. Existe un elemento botón, al que podemos asociar diferentes funcionalidades. De esta forma no nos tenemos que ceñir los botones de *submit* o *reset* que nos ofrecen los formularios.

El botón podrá tener cualquier texto en su interior a través de la inicialización del atributo `value`.

```
<input type="button" name="opcion" value="Opcion validar"/>
```

Como vemos en el código anterior inicializamos el atributo `type` al valor `button`, de esta manera el elemento se convierte en un botón. Estos botones por sí mismos no tienen funcionalidad. Habrá que aplicar una funcionalidad por medio de JavaScript para que se accione alguna tarea.

A continuación vamos a mostrar el código de un formulario con varios componentes para ver cómo se integran los componentes del mismo entre sí.

```
<form action="pagina.php" method="post"
  enctype="multipart/form-data" /> <br/>

  Nombre: <input type="text" name= "nombre" value="" size="42
    maxlength="30" /> <br/>

  Apellidos: <input type="text" name="apellidos" value=""
    size="40" maxlength="80" /> <br/>

  DNI: <input type="text" name="dni" value="" size="10"
    maxlength="9" /> <br/>
  Sexo: <br/>
  <input type="radio" name="sexo" value="hombre"
    checked="checked" />
  Hombre <br/>

  <input type="radio" name="sexo" value="mujer" />
  Mujer  <br/>
  Incluir mi foto: <input type="file" name="foto" /> <br/>

  <input name="publicidad" type="checkbox" value="publicidad"
    checked="checked"/> Enviar publicidad <br/>

  <input type="submit" name="enviar" value="Guardar cambios"
    />
  <input type="reset" name="limpiar" value="Borrar los datos
    introducidos" />
</form>
```

Este sería un formulario combinando la estructura básica vista anteriormente con varios de los elementos que hemos visto en los puntos anteriores. El aspecto del formulario en el navegador es el siguiente:

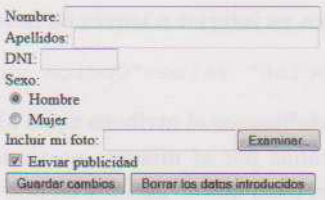

Figura 5.10. *Aspecto de un formulario dentro de un navegador*

5.3 MODIFICACIÓN DE APARIENCIA Y COMPORTAMIENTO

El lenguaje HTML no es muy flexible a la hora personalizar el aspecto de las páginas web. En cambio, habitualmente observamos que las páginas web muestran un aspecto de lo más variado, asemejando los colores, tonalidades y formas al objetivo de la página. Los formularios que hemos visto en los puntos anteriores, cumplen con las necesidades generales que exigen las aplicaciones para interactuar en la página web, pero a menudo surgen necesidades específicas para el desarrollo de una aplicación. Estas necesidades demandan una modificación del aspecto y para eso HTML hace uso de estilos. Las hojas de estilo modifican la apariencia de una página web. En el caso que nos atañe, las aplicaciones web, es especialmente importante poder modificar el aspecto, para conseguir que el usuario final tenga la sensación de manejar la aplicación igual que si lo hiciera en local.

En los siguientes puntos veremos que a través de estos estilos podemos mejorar notablemente el aspecto de los formularios, redondeando más sus formas, haciendo más uniformes los colores y estructurando las partes del formulario para que estos obtengan un aspecto mucho más agradable.

Los formularios tal y como los hemos visto en el punto anterior, pueden llegar a resultar un tanto estáticos. La diversidad de necesidades a la hora de desarrollar una aplicación web, nos hace pensar que necesitamos alguna herramienta que haga más flexible la utilización de formularios. De esta forma la proyección de un formulario en la aplicación web se puede disponer con una visualización u otra dependiendo de ciertas condiciones que vaya introduciendo el usuario.

En los puntos siguientes vamos a analizar cómo abordar el comportamiento de los formularios para que sus acciones tengan un funcionamiento para los que originalmente no estaban preparados. Para ello haremos uso de un lenguaje de *scripts* que nos permita personalizar las acciones y así conseguir los objetivos.

5.3.1 MODIFICACIÓN DE LA APARIENCIA DE UN FORMULARIO

Los formularios, por defecto, tienen unos estilos asignados. Estos estilos tienen asociados unos colores y unos bordes determinados. Desde los estilos por defecto, se pretende que estos se adapten en el entorno web en el que van a ser utilizados. Lamentablemente es difícil generalizar un estilo idóneo. Además, habitualmente, necesitamos personalizar tanto las formas como los colores para que estos se mimeticen con el resto de la aplicación web.

Para modificar el aspecto de un formulario, HTML utiliza estilos. Los estilos indican ciertas características de la apariencia de uno o más elementos del formulario. El lenguaje que maneja los estilos en HTML se llama *Cascading Style Sheets*. Estas siglas significan hojas de estilo en cascada. Habitualmente se denomina a este lenguaje CSS. Las modificaciones que podemos realizar con CSS son casi ilimitadas. A continuación vamos a ver cómo podemos modificar la apariencia de algunos elementos principales de un formulario.

- **Modificar el aspecto de un botón**. En ocasiones puede que necesitemos integrar un botón de un formulario en un texto o simplemente vemos la necesidad de que el enlace que envíe el formulario debería tener otro formato distinto al prefijado por el botón de formulario. A continuación vamos a modificar el aspecto de un botón para que este se muestre con una letra de un color diferente y tenga un borde más ancho de lo normal.

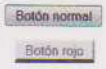

Figura 5.11. *Vista del mismo botón con aspecto diferente*

En la imagen vemos que el primer botón tiene el aspecto habitual para los formularios. En el segundo hemos modificado el color de la letra y el borde, agrandando el contorno del botón y cambiado el color de la línea inferior del mismo.

Para realizar esta modificación hemos aplicado un estilo al botón. Según el estilo aplicado el código del botón quedaría de la siguiente manera:

```
<input class="azul" type="button" value="Botón azul" />
```

Observamos como en el código hemos incluido un atributo llamado `class` al botón. De esta forma el botón buscara el estilo CSS correspondiente a ese nombre. El código CSS puede ser llamado de tres formas. Podemos aplicarlo directamente al elemento, práctica nada recomendable puesto que en el caso de querer modificar el estilo tendríamos que ir elemento por elemento. Otra de las formas es, generar un documento aparte que contenga los estilos, de manera que al comienzo de la página HTML se llame a este fichero y los elementos busquen su respectivo nombre de estilo en él. Y, por último, el modo que vamos a utilizar nosotros que es, incluir en la parte superior de la pagina HTML estos estilos. Los estilos CSS van incluidos dentro de la etiqueta `head`, que traducido al español significa cabecera. En el siguiente código mostramos el código CSS necesario para realizar la modificación del botón que hemos visto en la imagen anterior.

```
<html>
  <head>
    <title>Formulario</title>
    <style type="text/css">
      .azul{
        color: red;
        border-bottom: 4px solid blue;
      }
    </style>
  </head>
  <body>
    ...
  </body>
</html>
```

En el código podemos observar que el estilo `.azul` es el candidato que ha encontrado el atributo `class` del elemento botón para modificar sus colores y formas.

Imaginemos que queremos que el botón se muestre como un texto normal. Bastaría con que modificáramos el estilo indicando cómo queremos que se muestre éste. El código CSS sería el siguiente:

```
<style type="text/css">
  .azul{
    border: 0;
    background-color: transparent;
  }
</style>
```

Podemos observar en el código anterior como el borde del botón lo inicializamos a 0 y el color del botón como trasparente. De esta forma el botón se mostraría como un texto.

- **Suavizar el aspecto de un campo de texto**. Los campos de texto por defecto comienzan a escribir justo al principio del elemento. Al aparecer tan pegado la sensación que da al usuario al visualizarlo no es demasiado agradable. Desde CSS podemos modificar el punto a partir del que queremos que se escriban los caracteres. En la siguiente imagen mostramos la diferencia entre un campo de texto con el fondo amarillo, con el margen (`padding`) por defecto y debajo en el que se ha indicado un poco de margen en la parte izquierda.

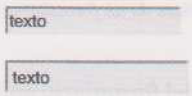

Figura 5.12. *Vista de un campo de entrada sin padding y con él*

El código correspondiente a los elementos *input* de tipo `text` anterior sería el siguiente:

```
<input class="col" type="text" value="texto" /></br></br>

<input class="color" type="text" value="texto" />
```

Para que los *input* recojan de las clases correspondientes a los estilos `col` y `color` necesitamos implementar el código CSS en la cabecera de la pagina HTML. El código es el siguiente:

```
.col{
  background-color: #FFFF00;
}

.color{
  padding: .2em;
  background-color: #FFFF00;
}
```

En el código se visualiza que el primer *input* pinta el fondo del campo de texto en amarillo y no tiene en cuenta el margen izquierdo. En el segundo caso de la llamada `color` se incluye un `padding` de 2 para que el aspecto visual quede más cuidado.

- **Organizar controles de un formulario**. Uno de los principales problemas que nos podemos encontrar a la hora de generar un formulario es cómo colocar los elementos para queden ordenados y bien guiados unos con otros. El aspecto de un formulario depende de que los elementos que lo integren estén bien organizados. A menudo se utilizan saltos de línea y tablas para integrar los elementos en sus celdas. Incluir CSS para guiar los elementos facilita la tarea, aumentando la capacidad de reutilización de código y modificaciones futuras. A continuación vamos a mostrar cómo organizar un formulario utilizando bloques. El código resultante mostraría un formulario con el siguiente aspecto.

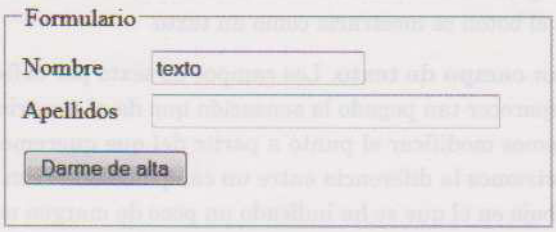

Figura 5.13. Vista de un formulario en un bloque

En la imagen hemos colocado los elementos del formulario aplicando bloques para que el guiado sea equivalente en elementos de todas las filas. El código a nivel de formulario sería el siguiente:

```
<fieldset>
  <legend>Formulario</legend>
  <div>
    <label for="nombre">Nombre</label>
    <input type="text" id="nombre" />
  </div>
  <div>
    <label for="apellidos">Apellidos</label>
    <input type="text" id="apellidos" size="35" />
  </div>
  <input class="btn" type="submit" value="Darme de alta" />
</fieldset>
```

En el código anterior no hemos introducido estilos. Solamente hemos estructurado los elementos dentro de un cuadro (`fieldset`). El *input* realiza una llamada a la clase `btn`. Veamos que a través de los estilos ordenamos los elementos:

```
<style type="text/css">
  div {
    margin: .4em 0;
  }

  div label {
    width: 25%;
    float: left;
  }

  .btn {
    display: block;
    margin: 1em 0;
  }
</style>
```

En el código anterior asignamos un margen superior de 4 y un margen izquierdo de 0 a todos los `div` que existan. En segundo lugar indicamos que los `label` ocuparan un 25% del `div`. El botón final lo agrupamos en forma de bloque, por lo que saldrá en una fila él solo y el margen superior que indicamos es de 1 para que no quede tan junto con la fila superior.

5.3.2 MODIFICACIÓN DEL COMPORTAMIENTO DE UN FORMULARIO

Los formularios tienen unas acciones predeterminadas por defecto. Estas acciones están asociadas a los elementos así como a la estructura principal del formulario. En el caso de que queramos darle otro comportamiento a un elemento del formulario, tenemos que modificar el elemento para que este no realice su función habitual. Imaginemos que tenemos la estructura principal de un formulario pero queremos que los datos se envíen a URL diferentes dependiendo de un dato que introduzca el usuario. En este caso deberíamos modificar el atributo `action` indicado en la estructura principal. Para realizar el envío a la URL podríamos accionar en un *script* la petición al servidor. A continuación mostramos un ejemplo de cómo se enviaría el formulario a una URL diferente:

```
<script language="javascript">
  function enviar(form){
    if (formulario.alta.checked == true){
      formulario.action = "paginas/alta.html";
    }
    if (formulario.alta.checked == false){
      formulario.action = "paginas/baja.html";
    }
    form.submit()
  }
</script>
```

En el código anterior comprobamos si la casilla `alta` ha sido chequeada. En caso de haber sido pulsada el `action` del `form` se inicia a un valor, si no ha sido pulsada se inicia a otro. Al final se envía el formulario por medio de un `submit`. Para que la función `enviar` sea ejecutada, debemos incluir un *input* de tipo `button` que contenga el evento `onclick`. En el manejador de este evento realizaremos la llamada de la siguiente forma: `Onclick="enviar (this.form)"`. El objeto `this` hace que podamos enviar el formulario a través de la función. De esta forma nunca llamamos al `action` de la estructura principal del formulario.

5.4 VALIDACIÓN Y ENVÍO

A la hora de de enviar un formulario, no podemos garantizar que el usuario haya insertado cada dato del formulario como esperamos. Supongamos que en un campo de texto el usuario tiene que incluir un código postal, pero el usuario en ese campo ha incluido una cadena de texto, por ejemplo, Madrid. Si el formulario se envía con ese dato se producirá un error. Para controlar si los campos de un formulario han sido rellenados correctamente haremos uso de las validaciones. Existen varios tipos de validaciones. Podemos realizar validaciones en el servidor, a nivel de servidor y también en la base de datos. Realizar validaciones a nivel de servidor supone que se recibe una petición, el servidor tiene que procesarla y emitir una respuesta. En el caso de que el usuario incluya muchos datos que no son correctos el servidor se puede ver sobrecargado. Para solucionar esta sobrecarga del servidor, realizamos un primer filtrado de los datos en el cliente. Estas validaciones son las que vamos a ver a continuación. En ellas analizamos con *scripts* en el navegador los datos que ha incluido el usuario una vez que pulsa el botón de enviar formulario. La petición no es enviada al servidor y se analiza dentro del navegador si existe algún dato que no se haya introducido correctamente. En el caso de que exista algún dato que no es correcto el *script* asignado muestra un mensaje al usuario indicando que debe realizar alguna modificación en el dato introducido. Si, por el contrario, se validan como correctos los datos, se envían los datos del formulario al servidor.

Este tipo de validaciones se suelen realizar llamando a una función que analice si el dato cumple con las restricciones que se han impuesto como, por ejemplo, en el caso anterior, donde el dato tenía que cumplir con los caracteres que puede tener un código postal. A continuación vamos a ver algunos de los ejemplos de validación más habituales, los tipos de validación que pueden existir son de lo más variado y basta con definirlas dentro de cada una de las funciones que valida un dato.

5.4.1 ESTRUCTURA DEL FORM PARA VALIDAR DATOS

Para que el formulario detecte que tiene que realizar una validación de los datos antes de enviar la petición al servidor, debemos indicárselo en su estructura. Para ello utilizaremos el evento `onsubmit` que ejecuta una acción cuando el `action` del formulario es llamado. Dentro del evento `onsubmit`, debemos llamar a una función, dependiendo de la respuesta de la función, el formulario enviará los datos al servidor (si los datos son correctos) o mostrará los diferentes mensajes de error, para los controles del formulario que se hayan realizado validaciones. El código correspondiente sería el siguiente:

```
<form action="URL" method="post" name="formValidado"
  onsubmit="return validar()">
  ...
</form>
```

En el código anterior observamos como la función `validar()` es llamada dentro del evento `onsubmit` y que dependiendo de la respuesta de validar el servidor envía o no los datos del formulario al servidor. Al realizarse un `submit` desde algún botón del formulario (petición de envío del formulario al servidor), siempre se realiza la llamada a la función `validar()`. Para que los datos del formulario sean validados, debemos tener implementada la función `validar()`. En esta función indicaremos cuáles son los datos del formulario que vamos a validar. A continuación mostramos algunos ejemplos de cómo validar datos de un formulario.

- **Validar un campo como obligatorio**. En ocasiones puede que nos interese que el usuario ingrese un campo como obligatorio en la aplicación web. En esta situación si el usuario no introduce el dato y antes de que el formulario sea enviado debemos comprobar si el campo está vacío. Habiendo llamado en la estructura del `form` a una función, la implementación de la función será la siguiente:

```
<script type="text/javascript">
  function validacion() {
    valor = document.getElementById("campo").value;

    if( valor == null || valor.length == 0 ){
      alert("El campo no puede ser vacío");
      return false;
    }
    return true;
  }
</script>
```

Este código deber ir integrado en la cabecera de la página HTML (`<head>`...código JavaScript...`</head>`). En primer lugar, recogemos el valor a través del objeto `document`, el nombre del *input* es `campo`. En el capítulo posterior veremos con detalle el árbol de estructura de una página web. En la estructura `if` comprobamos que el valor no sea nulo y que su longitud no sea cero. En caso de que el valor del campo `input` sea nulo o igual a cero, mostramos un mensaje de alerta y, posteriormente, devolvemos falso para que sea recuperado por el evento `onsubmit`. En ese caso no se enviaría el formulario.

- **Validar un campo de texto como numérico**. A menudo en los formularios nos encontramos con campos de texto que solo admiten números. Los teléfonos, código postal, DNI (sin letra) y otros muchos datos que obligatoriamente tienen una nomenclatura numérica. Para advertir al usuario de que el dato que ha introducido no es correcto utilizaremos el siguiente código implementado en una función:

```
<script type="text/javascript">
  function validaNum() {
    valor = document.getElementById("telefono").value;

    if( isNaN(valor) ) {
      alert("El campo tiene que ser numérico");
      return false;
    }
    return true;
  }
</script>
```

En el código anterior, tenemos una variable que se llama `valor`. Esta variable tomará el valor del `teléfono` o en su defecto el dato que haya introducido el usuario en la aplicación web. Con la función `isNaN()` comprobamos si el valor es numérico. En caso de que no lo sea devolverá un mensaje de alerta indicando que el dato tiene que ser numérico. Si por el contrario el dato es numérico la función devolverá verdadero.

- **Validar si una fecha es correcta**. Las fechas a menudo suelen ser datos en los formularios difíciles de comprobar, la razón es que hay infinidad de formas de expresarlas. En el ejemplo siguiente mostramos cómo validar de una forma sencilla una fecha, en la que se han ingresado el año, mes y día de forma independiente cada uno a través de tres campos diferentes:

```
<script type="text/javascript">
  function validaFecha() {
    var dia = document.getElementById("dia").value;
    var mes = document.getElementById("mes").value;
    var ano = document.getElementById("ano").value;

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

    if( !isNaN(fecha) ) {
      return false;
    }
    return true;
  }
</script>
```

En el código anterior recogemos tres valores uno por el día otro por la fecha y el último por el año. Con la función `Date()` generamos una fecha introduciendo los valores. Posteriormente, comprobamos si la fecha que hemos creado es correcta, en caso de que el resultado sea que la fecha no es válida la función devuelve `false`.

- **Validar un checkbox**. Otra de las validaciones que podemos necesitar habitualmente, es comprobar si un *checkbox* ha sido seleccionado. A menudo en la aceptación de ciertas condiciones para acceder a servicios nos encontramos con esta comprobación, sin la que el formulario no tiene mucho sentido que continúe. En el siguiente código vamos a ver cómo se implementa la comprobación de si se ha activado un *checkbox*:

```
<script type="text/javascript">
  function validaCheck() {
    elemento = document.getElementById("campoCondiciones");

    if( !elemento.checked ) {
      return false;
    }
    return true;
  }
</script>
```

Si elemento checkbox no ha sido seleccionado

En el código anterior recuperamos el valor que ha tomado el elemento *checkbox* del formulario. En la siguiente condición comprobamos si éste ha sido chequeado, en caso de que no se haya pulsado la función devolverá falso. Si se ha chequeado la función devuelve verdadero.

Conviene tener en cuenta que si somos demasiado estrictos a la hora de validar o damos poca información, podemos bloquear al usuario de la aplicación web. Es posible que el usuario no sepa la manera de continuar en el proceso de manejo debido a un error en el tipo y validación de los datos.

5.5 EXPRESIONES REGULARES

Las expresiones regulares describen un conjunto de elementos que siguen un patrón. Dicho de otra forma, es una regla que identifica a una serie de elementos que tiene algo en común. Un ejemplo de expresión regular podrían ser todas las palabras que comienzan por la letra "a" minúscula. La expresión regular para este patrón debe asegurarse de que la palabra comienza por "a" minúscula, el resto de palabras que precedan a la cadena podrán ser cualquier carácter.

La mayoría de los lenguajes de programación incluyen la implementación de expresiones regulares. En el caso que nos atañe, JavaScript implementa una configuración para implementar expresiones regulares y así facilitar las comprobaciones de ciertos datos que deben seguir una estructura concreta. Este tipo de expresiones es muy útil a la hora de evaluar algunos tipos de datos que normalmente utilizamos en los formularios.

5.5.1 CARACTERES ESPECIALES DE LAS EXPRESIONES REGULARES

A continuación vamos describir algunos de los elementos que utiliza JavaScript para crear expresiones regulares. En el siguiente apartado vamos a representar la simbología que nos presta la expresión regular y, posteriormente, significado o interpretación:

- ✓ **^ Principio de entrada o línea**. Este carácter indica que las cadenas deberán comenzar por el siguiente carácter. Si éste fuera una "a" minúscula, como indicamos en el punto anterior, la expresión regular seria sería, `^a`.
- ✓ **$ Fin de entrada o línea**. Indica que la cadena debe terminar por el elemento precedido al dólar.
- ✓ *** El carácter anterior 0 o más veces**. El asterisco indica que el carácter anterior se puede repetir en la cadena 0 o más veces.
- ✓ **+ El carácter anterior 1 o más veces**. El símbolo más indica que el carácter anterior se puede repetir en la cadena una o más veces.
- ✓ **? El carácter anterior una vez como máximo**. El símbolo interrogación indica que el carácter anterior se puede repetir en la cadena cero o una vez.
- ✓ **. Cualquier carácter individual**. El símbolo punto indica que puede haber cualquier carácter individual salvo el de salto de línea.
- ✓ **x|y x ó y**. La barra vertical indica que puede ser el carácter *x* o el *y*.
- ✓ **{n} n veces el carácter anterior**. El carácter anterior a las llaves tiene que aparecer exactamente *n* veces.
- ✓ **{n,m} Entre n y m veces el carácter anterior**. El carácter anterior a las llaves tiene que aparecer como mínimo *n* y como máximo *m* veces.
- ✓ **[abc] Cualquier carácter de los corchetes**. En la cadena puede aparecer cualquier carácter que esté incluido en los corchetes. Además, podemos especificar rangos de caracteres que sigan un orden. Si se especifica el rango [a-z] equivaldría a incluir todas las letras minúsculas del abecedario.

- **[^abc] Un carácter que no esté en los corchetes**. En la cadena pueden aparecer todos los caracteres que no estén incluidos en los corchetes. También podemos especificar rangos de caracteres como en el punto anterior.
- **\b Fin de palabra**. El símbolo `\b` indica que tiene que haber un fin de palabra o retorno de carro.
- **\B No fin de palabra**. El símbolo `\B` indica cualquiera que no sea un límite de palabra.
- **\d Cualquier carácter dígito**. El símbolo `\d` indica que puede haber cualquier carácter numérico, de 0 a 9.
- **\D Carácter que no es dígito**. El símbolo `\D` indica que puede haber cualquier carácter siempre que no sea numérico.
- **\f Salto de página**. El símbolo `\f` indica que tiene que haber un salto de página.
- **\n Salto de línea**. El símbolo `\n` indica que tiene que haber un salto de línea.
- **\r Retorno de carro**. El símbolo `\r` indica que tiene que haber un retorno de carro.
- **\s Cualquier espacio en blanco**. El símbolo `\s` indica que tiene que haber un carácter individual de espacio en blanco: espacios, tabulaciones, saltos de página o saltos de línea.
- **\S Carácter que no sea blanco**. El símbolo `\S` indica que tiene que haber cualquier carácter individual que no sea un espacio en blanco.
- **\t Tabulación**. El símbolo `\t` indica que tiene que haber cualquier tabulación.
- **\w Carácter alfanumérico**. El símbolo `\w` indica que puede haber cualquier carácter alfanumérico, incluido el de subrayado.
- **\W Carácter que no sea alfanumérico**. El símbolo `\W` indica que puede haber cualquier carácter que no sea alfanumérico.

5.5.2 VALIDAR UN FORMULARIO CON EXPRESIONES REGULARES

Con la combinación de estas expresiones podemos abordar infinidad de patrones para validar datos en la mayoría de los formularios. Combinando cada una de las condiciones obtenemos patrones como pueden ser, una dirección de correo electrónico, un teléfono, un código postal, un DNI, etc. A continuación, vamos a ver cómo realizar algunas de las configuraciones de expresiones regulares más utilizadas para validad datos de un formulario.

- **Validar una dirección de correo electrónico**. En el caso de que el usuario de la aplicación web introduzca una dirección de correo electrónico, es casi imprescindible que esta sea correcta. Utilizando expresiones regulares vamos a comprobar que la estructura de la dirección sea correcta. En primer lugar, comprobaremos que comienza por una cadena de texto, que sigue por una @ y que termina por otra cadena de texto un punto y otra cadena de texto. De esta forma aseguraremos que, al menos, la estructura de la dirección es correcta. Combinando las funciones con las expresiones regulares, obtenemos un código que valida una dirección de correo electrónico.

  La llamada en la estructura principal del `form` sigue siendo la misma que en los casos anteriores y recogemos en el evento `onsubmit` la respuesta de la llamada a la función que valida, en nuestro caso la dirección de correo electrónico. El código correspondiente a la función JavaScript que valida un correo electrónico sería el siguiente:

```
<script type="text/javascript">
  function validaEmail() {
    valor = document.getElementById("campo").value;

    if(!(/\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
      .]\w+)/.test(valor)) ) {
      return false;
    }
    return true;
  }
</script>
```

- **Validar un DNI**. Un dato muy habitual en los formularios es el documento nacional de identidad. A la hora de que el usuario interactúe con la aplicación web, es necesario comprobar que el DNI que introduce es correcto. No podemos asegurar que el DNI sea el de la persona que se está identificando, pero sí podemos asegurar que el DNI que introduce el usuario es correcto. Para validad un DNI introducido podemos utilizar el siguiente código.

```
<script type="text/javascript">
  function validaDNI(){
    valor = document.getElementById("dni").value;

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

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

    if(valor.charAt(8) != letras[(valor.substring(0, 8))%23])
    {
      return false;
    }
    return true;
}
</script>
```

En el código anterior validamos un DNI en varios pasos. Primero recogemos en la variable `valor` el campo del DNI que ha introducido el usuario. Generamos un *array* con las letras validas para el DNI. En la primera condición comprobamos a través de expresiones regulares que el patrón del campo introducido tiene una longitud de 8 números y una letra. Si no es así devuelve falso la función.

En la última condición realizamos la división entre 23 y tomamos el resto. Comprobamos que la posición del *array* `letras`, perteneciente al resto de la operación, se corresponde con el valor de la letra del DNI introducido. En caso de que sea igual devuelve verdadero (al final de la función). Si no es igual la letra del valor introducido a la calculada, entra en la condición y devuelve falso.

- **Validar un número de teléfono**. Otro dato que es importante que sea correcto para interactuar en una aplicación web con el usuario son los números de teléfono. Al igual que en casos anteriores, no podemos comprobar que el numero pertenezca al usuarios que lo introduce, pero si podemos asegurar que el número de teléfono tiene un formato correcto. Las expresiones regulares son un recurso muy útil para validar un teléfono. A continuación vamos a ver cómo validaremos un número de teléfono:

```
function validaTelefono(){
  valor = document.getElementById("telefono").value;

  if( !(/^\d{9}$/.test(valor)) ) {
    return false;
  }
  return true;
}
```

En el código anterior estamos validando que el número que introdujo el usuario este formado por dígitos y tenga una longitud de 9. A la hora de validar un teléfono podemos ser más estrictos. Si queremos que el campo solo corresponda a un número de teléfono móvil, podemos especificar que el número solo pueda comenzar por 6, puesto que en España los teléfonos móviles comienzan todos por 6. La expresión regular integrada en la condición anterior para comprobar esto sería la siguiente:

```
if( !(/^[6]\d{8}$/.test(valor)) ) {
```

Esta expresión indica que el dato debe comenzar por el número 6 y que los siguientes 8 caracteres deben ser numéricos. En el caso de que queramos validar un número de teléfono fijo, el código sería el que mostramos a continuación:

```
if( !(/^[89]\d{8}$/.test(valor)) ) {
```

En el código los números solo pueden comenzar por 8 o por 9, que son los números actuales válidos en España.

En el caso de que la nomenclatura de teléfonos cambiara por alguna razón tecnológica, solamente hay que cambiar la función que valida el dato, para que ésta permita devolver verdadero en los casos de los teléfonos que cumplan con las expresiones regulares.

Existen otros muchos datos que podemos validar con expresiones regulares, como pueden ser números de cuentas bancarias, CIF de empresas y cualquier dato que siga un patrón determinado. Validar los datos para que el usuario no envíe el formulario si estos no son correctos es algo que aumenta el rendimiento del servidor, al disminuir las peticiones. También es importante mostrar una información amplia de como el usuario tiene que introducir los datos que se validen, de lo contrario, si el usuario no conoce cómo debe ingresar el dato puede quedarse bloqueado ante la aplicación. Ser demasiado estrictos en las validaciones a veces es contraproducente puesto que la curva de aprendizaje de la aplicación web puede aumentar considerablemente.

En los casos de validaciones estrictas tenemos que tener en cuenta que el servicio de atención al usuario debe tener medios para facilitar el manejo de la aplicación web al usuario, bien sea telefónico, presencial o por correo electrónico, dependiendo de la infraestructura en la que se enmarque nuestra aplicación.

5.6 UTILIZACIÓN DE COOKIES

Para poder hablar de las *cookies* antes es necesario conocer cómo funciona el protocolo de comunicaciones HTTP (*HyperText Transfer Protocol*), puesto que las *cookies* surgieron como necesidad ante algunas ausencias tecnológicas de este protocolo. El protocolo HTTP fue desarrollado por W3C (*Consorcio World Wide Web*), entre el año 1999 y 2000. Este protocolo sigue el esquema siguiente.

Petición cliente >>>>>> Respuesta del servidor

Este protocolo no tiene estado o conexión, lo que quiere decir que no guarda información de las conexiones anteriores. Realiza la petición el cliente, el servidor responde y ahí finaliza la comunicación. En los comienzos del uso de páginas web, apenas había necesidad de mantener una comunicación con el cliente en una página. La información se mostraba en una página. Para ir a otra página que enlazaba con la anterior, simplemente se pulsaba en el enlace y se mostraba la nueva página. Con la evolución de las páginas web se comenzó a ver la necesidad de identificar al usuario que realizaba la petición al servidor. En una página que muestre el correo electrónico identificamos al usuario mediante un nombre y una contraseña pero, y en la siguiente página... ¿cómo sabe el servidor que envía los datos al cliente que se *logueó*? Podría realizar una petición de esa URL otro navegador y visualizar sin ningún tipo de validación, la interfaz de entrada del correo electrónico. Para resolver este problema nacieron las *cookies*.

Una *cookie* es un fichero de texto que se almacena en el ordenador del cliente. Cada navegador especifica una ruta en la que guarda sus *cookies*. Por lo tanto, las *cookies* son un fichero propio de cada navegador. El servidor solicita al navegador que este guarde las *cookies*. En las sucesivas peticiones del cliente (navegador) al servidor, el navegador envía la petición junto con la *cookie*. De esta forma, el servidor sabe que quien le realiza la petición es el cliente anterior y no otro.

Las *cookies* surgieron con el fin de mantener la información de los carritos de la compra virtuales, para la compra a través de la Web. Por medio de las *cookies* el usuario podía navegar entre las páginas sin perder los elementos que había seleccionado para comprar, pudiendo modificar esta cesta virtual a través de la misma aplicación web. Debido a las carencias del protocolo HTTP en seguida se comenzaron a usar las *cookies* para otros fines de identificación de usuarios.

5.6.1 USOS DE LAS COOKIES

El principal objetivo de las *cookies* es que el servidor pueda conocer alguna características del usuario (navegador), para así poder tomar una decisión con respecto si envía una respuesta, cómo la envía y qué datos van asociados. Partiendo de este objetivo común los fines que puede tener un servidor para solicitar al usuario que almacene *cookies* en su ordenador son muy diversos. A continuación vamos a ver algunos de los principales usos que se da a las *cookies*. En los puntos siguientes vamos a ver que todos los usos de las *cookies* no son igual de lícitos y que si el navegador no está bien configurado puede ser un agujero de seguridad. Al navegar por páginas que no ofrezcan confianza podemos tener una vulnerabilidad en el sistema, el servidor de la página de dudosa confianza podría acceder al sistema accediendo al disco para fines a los que no le hemos autorizado. Para solucionar estos problemas podemos limitar o anular el uso de las *cookies* en nuestro navegador.

- **Mantener opciones de visualización**. Las *cookies* son utilizadas en ocasiones para mantener unas preferencias de visualización. Algunas páginas como Google, permiten que el usuario haga una configuración de su página de entrada en el buscador, a través de las *cookies* el servidor reconoce ciertos aspectos que el usuario configuró y conserva el aspecto.
- **Almacenar variables**. El servidor puede utilizar las *cookies* para almacenar variables que se necesiten utilizar en el navegador, puede almacenar los campos de un formulario que se va visualizando en páginas diferentes. Un ejemplo sería una página en la que nos solicitan unos datos, en la siguiente nos solicitan otros datos y así hasta la página final. Los datos de las páginas anteriores se irán almacenando en las *cookies* hasta que se finaliza el ciclo del formulario y el usuario envía los datos al servidor. Antes del envío al servidor se recuperarán todos los campos del formulario que están guardados en las *cookies*.
- **Realizar un seguimiento de la actividad de los usuarios**. En ocasiones los servidores hacen uso de las *cookies* para almacenar ciertas preferencias y hábitos que el usuario tiene a la hora de navegar. Con esta información, el servidor recoge información para poder personalizar sus servicios y publicidad orientándolo a cada cliente en particular. Estos fines no son del todo lícitos si la entidad que realiza estas actividades no avisa al usuario de que está realizando estas acciones.
- **Autenticación**. Es uno de los usos más habituales de las *cookies*, autenticar a los usuarios. A través de las *cookies*, el navegador guarda los datos del usuario, al realizar una petición al servidor, el navegador envía las *cookies* junto con la petición. De esta forma el servidor sabe que el usuario que realiza la petición es el que se autentico la primera vez. Las *cookies* tiene una caducidad, cuando pasa un período de tiempo establecido, estas desaparecen junto con el fichero de texto que guarda el navegador. En el caso de la autenticación las *cookies* suelen caducar cuando se finaliza la navegación por la página, cerrando la ventana. Habitualmente, como mecanismo de seguridad, las aplicaciones web aplican un tiempo máximo de inactividad, por ejemplo de 15 minutos, tras el cual las *cookies* caducan si no se produjo movimiento en la navegación de la aplicación web.

5.6.2 LECTURA Y ESCRITURA DE LAS COOKIES

Hasta ahora hemos hablado de que son y para qué sirven las *cookies* pero, ¿cómo se implementan? Al principio las *cookies* solo podían ser almacenadas por la petición de un CGI (*Common Gateway Interface*) desde el servidor. Un CGI es, según su traducción, una interfaz de entrada común. Es una tecnología que permite a un cliente (navegador) solicitar datos de un programa que se ejecuta en un servidor.

La compañía de software Netscape implementó en su lenguaje JavaScript la capacidad de introducir las *cookies* directamente desde el cliente (navegador), sin necesidad de CGI. Al principio se produjeron ciertas vulnerabilidades de seguridad, por lo que es muy importante configurar convenientemente el navegador, puesto que las *cookies* pueden ser creadas, borradas, aceptadas o bloqueadas directamente en el navegador. A continuación vamos a ver cómo se escribe y se lee una *cookie* del navegador.

- **Implementación de lectura y escritura de una cookie**. Los dos procesos de implementación principales de una *cookie* son la escritura y la lectura de la misma. A continuación vamos a ver el código de cómo se escribe una *cookie* y cómo se lee:

Retorna el valor de la cookie especificada por c_name

```
<html>
 <head>
  <script type="text/javascript">
   function getCookie(c_name){
    var i,x,y,ARRcookies=document.cookie.split(";");
    for (i=0;i<ARRcookies.length;i++){
     x=ARRcookies[i].substr(0,ARRcookies[i].indexOf("="));
     y=ARRcookies[i].substr(ARRcookies[i].indexOf("=")+1);
     x=x.replace(/^\s+|\s+$/g,"");
     if (x==c_name){
      return unescape(y);
     }
    }
   }
```

Muestra valor de la cookie

nombre de la cookie

valor de la cookie

fecha expiración

```
   function setCookie(c_name,value,exdays){
    var exdate=new Date();
    exdate.setDate(exdate.getDate() + exdays);
    var c_value=escape(value) + ((exdays==null) ? "" : ";
    expires="+exdate.toUTCString());
    document.cookie=c_name + "=" + c_value;
   }
```

Crea o actualiza una Cookie

```
   function checkCookie(){
    var username=getCookie("username");
    if (username!=null && username!=""){
     alert("Bienvenido " + username);
    }else{
     username=prompt("Por favor, Introduzca su usuario:","");

     if (username!=null && username!=""){
     setCookie("username",username,365);
     }
    }
   }
  </script>
 </head>
 <body>
  <input type="button" name="chequeaCookie" value="Chequear
   las cookies" onclick="checkCookie();">
 </body>
</html>
```

nombre cookie

valor de la cookie

En el código anterior tenemos tres funciones JavaScript. La primera función devuelve el valor de una *cookie* de la cual hemos pasado el nombre a la función. Para introducir el valor de la *cookie* accedemos a través del árbol del documento (páginas HTML), llamado `document`.

En la segunda función escribimos una *cookie* pasando el nombre de la variable, el valor y la fecha de caducidad. Se comprueba la fecha actual y se le suman los días pasados por parámetro.

En la tercera función comprobamos si existe un valor para la *cookie*, en el caso de que no exista un valor se llama la función `setCookie()`. Ésta se encarga de solicitar al usuario el nombre e introducirlo en la *cookie*. La próxima vez que chequeemos la *cookie* el navegador contendrá ya el valor del nombre, por tanto, mostrará el nombre que introdujimos antes. Para volver a repetir la operación de insertar un nombre que guardemos en la *cookie*, debemos borrar las *cookies* del navegador.

Podemos probar que si desactivamos las *cookies* del navegador, este código que hemos visto anteriormente no funciona. Por lo tanto, las aplicaciones web que hagan uso de las *cookies* requieren que éstas estén activadas en el navegador. De lo contrario, por ejemplo, si se usan las *cookies* para logarse en la aplicación, el usuario no podría acceder a la misma.

ACTIVIDADES 5.1

- Se propone al alumno realizar una plantilla con los eventos que existen y dónde puede utilizar cada uno de ellos.
- La arquitectura de una aplicación web está compuesta por diferentes tecnologías (HTML, CSS, JavaScript). Defina en un esquema sobre cómo se unen y cuáles son los elementos principales que tiene cada una de ellas.
- En el tema hemos visto varios tipos de validaciones. Si tenemos que abordar la validación de un número de cuenta bancaria, un CIF, que un campo de entrada tenga como mínimo 6 caracteres y que un campo para poder ser rellenado dependa de si uno anterior se rellenó, ¿cómo podríamos implementar estas validaciones?
- Realice un estudio de cuáles pueden ser los problemas que generan las *cookies* en una aplicación web. Busque en Internet qué alternativas existen para abordar la funcionalidad de las *cookies*. Concluya si la tendencia es utilizar las *cookies* o si son otras tecnologías las que se están tendiendo a utilizar.
- Busque en Internet qué es el protocolo HTTPS. Comente qué aporta este protocolo con respecto al uso de las *cookies*.

RESUMEN DEL CAPÍTULO

Los eventos ofrecen mecanismos para que cuando hay movimientos en una aplicación web se desencadenen acciones implementadas por el programador. Los eventos están asociados al teclado, al ratón, a la página del navegador o al árbol DOM.

Los formularios se utilizan para crear, modificar y manipular información. También sirven para ayudar a emular los comportamientos de cualquier aplicación en local que queramos transformar en la aplicación web.

Los estilos CSS modifican la apariencia de las paginas HTML y también de los formularios. De esta forma adaptamos la aplicación web a fin destinado. Se puede asemejar el aspecto de una aplicación web a la misma aplicación en local tanto que nos costaría indicar cuál es cada una.

La validación en el cliente sirve para disminuir las peticiones al servidor, dar más rapidez en la respuesta al cliente y comprobar los datos antes de ser enviados. Las expresiones regulares nos ayudan a validar los datos que tienen patrones de comportamiento.

Las *cookies* nos ayudan a mantener el estado entre las páginas web puesto que éstas carecen de esa característica. Además, nos permiten guardar información del usuario en el caso de que éste visite de nuevo la página para poder atenderle de forma más personalizada.

EJERCICIOS PROPUESTOS

1. Realice una pequeña aplicación en la que exista un formulario con los distintos tipos vistos en el tema. Valide cada campo para que el usuario no pueda introducir un tipo de datos diferente al requerido. Valide también para que no pueda dejar campos en blanco. Cuando el usuario introduzca los datos correctamente en el formulario introduzca estos datos en *cookies* y proporcione al formulario un botón para que estos puedan ser leídos.

TEST DE CONOCIMIENTOS

1 Los eventos son mecanismos que se accionan:
a) En el servidor.
b) En el cliente.
c) En el servidor y en el cliente.
d) Todas las anteriores.

2 Los eventos, según su origen, se dividen en:
a) Eventos de ratón y teclado.
b) Eventos de acción y reacción.
c) Eventos HTML y DOM.
d) La respuesta *b* no es correcta.

3 El orden de ejecución de los eventos:
a) Se ejecutan en el orden en el que se pusieron.
b) Tienen un orden de ejecución en cada caso.
c) Se ejecutan todos a la vez si se refieren al mismo elemento.
d) Hay que asignar la prioridad de ejecución a cada evento si queremos que cambie el orden de ejecución.

4 Los eventos HTML:
a) Son todos los eventos que existen.
b) Son los eventos que actúan cuando hay cambios en la ventana del navegador.
c) Estos eventos no se producen en el objeto `Window`.
d) `Submit` no es un evento HTML.

5 Para poder utilizar un formulario:
a) Es necesario tener un atributo `action`.
b) Es necesario tener un atributo `action` y uno `method`.
c) Es necesario tener un atributo `action`, un `method` y un `name`.
d) Es necesario tener un atributo `action` y uno `name`.

6 En un formulario:
a) Tenemos el atributo `type` que indica el tipo de elemento.
b) El atributo `name` identifica de forma única cada elemento.
c) El atributo `checked` está definido para todos los elementos.
d) Las respuestas *a* y *b* son correctas.

7 El tipo de elemento *contraseña*, que se define con `password`:
a) Cifra los datos introducidos en el campo para que al enviarlos estos no puedan ser interceptados.
b) Sustituye los campos por asteriscos o círculos para que no los vean las miradas indiscretas.
c) Modula la contraseña para que ésta no sea interceptada al enviarse.
d) No se usa casi nunca porque es muy poco seguro.

8 Para modificar el aspecto de un elemento del formulario:
a) Utilizamos hojas de estilos en la cabecera u otro fichero.
b) Aplicamos los estilos al elemento directamente con CSS, es lo más recomendable.
c) Aplicamos atributos de HTML para modificar el aspecto.
d) Usamos JavaScript para modificar el aspecto de los elementos.

9 Los datos se validan en el navegador para:
a) Dar seguridad a la aplicación y evitar fallos.
b) Evitar demasiadas peticiones al servidor, mejorar la experiencia del usuario y asegurar que la información enviada es correcta.
c) No tener que validar los datos en el servidor puesto que se sobrecarga más.
d) No tener que comprobar en la base de datos que los datos son correctos.

10 Las *cookies* son:
a) Almacenadas en el servidor.
b) Almacenadas en un directorio del ordenador cliente.
c) Almacenadas entre en servidor y el navegador.
d) Todas las anteriores son correctas.