1. HTML 5

La quinta revisión del lenguaje de programación HTML pretende remplazar al actual (X)HTML, corrigiendo problemas con los que los desarrolladores web se encuentran, así como rediseñar el código y actualizándolo a nuevas necesidades que demanda la web de hoy en día.

Actualmente se encuentra en modo experimental, lo cual indica la misma W3C; aunque ya es usado por múltiples desarrolladores web por sus avances, mejoras y ventajas.

A diferencia de otras versiones de HTML, los cambios en HTML5 comienzan añadiendo semántica y accesibilidad implícitas. Establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Algunos de ellos son técnicamente similares a las etiquetas y , pero tienen un significado semántico, como por ejemplo <nav> (bloque de navegación del sitio web) o <footer>. Otros elementos proporcionan nuevas funcionalidades a través de una interfaz estandarizada, como los elementos <audio> y <video>.

Algunos elementos de HTML 4.01 han quedado obsoletos, incluyendo elementos puramente de presentación, como <font> y , cuyos efectos se deben de realizar utilizando CSS. También hay un renovado énfasis en la importancia del scripting DOM para el comportamiento de la web.

1.1. Navegadores que lo soportan

Actualmente, de los navegadores de escritorio, el que mayor soporte da es Google Chrome, seguido muy de cerca por Mozilla Firefox y Apple Safari. El que menor compatibilidad ofrece es Internet Explorer.

Para comprobar la compatibilidad de un navegador podemos visitar la Web "http://www.html5test.com/" donde se realiza un test de todas las funcionalidades de HTML5.

1.2. Doctype

El doctype es el encargado de indicarle al navegador el tipo de documento que está abriendo, con el fin de renderizar la pagina de manera correcta. Por ejemplo, el doctype de HTML 4 es:

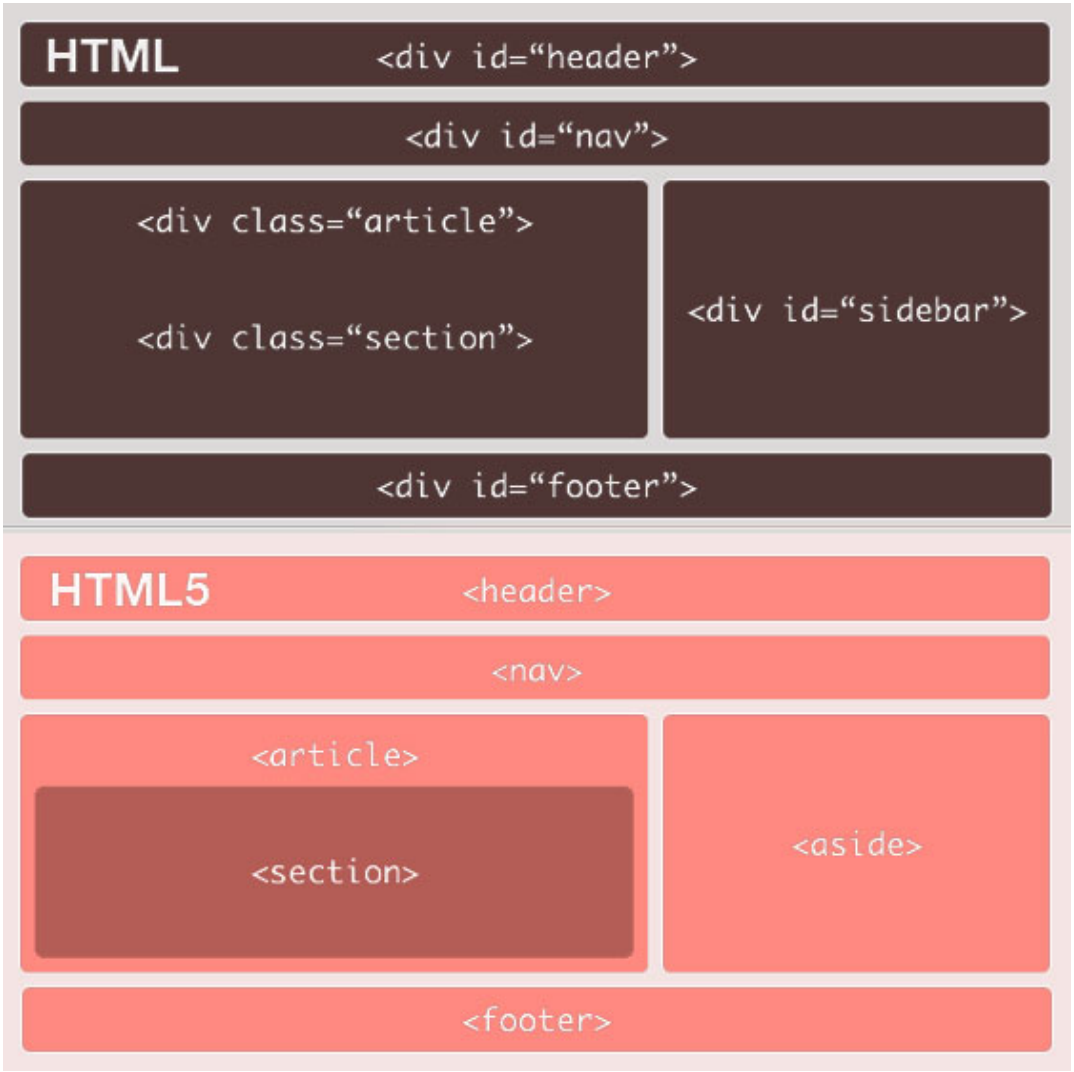
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Para HTML 5 el doctype se ha simplificado mucho y además es compatible con las versiones anteriores de HTML:

```
<!DOCTYPE html>
```

1.3. Mejor estructura

Hasta ahora se utilizaba de forma abusiva la etiqueta y las tablas para estructurar una web en bloques. El HTML5 nos brinda nuevas etiquetas que perfeccionan esta estructuración. Estas nuevas etiquetas introducen un nuevo nivel semántico que hace que la estructura de la web sea más coherente y fácil de entender. Además los navegadores podrán darle más importancia a determinadas secciones, facilitándole además la tarea a los buscadores, así como cualquier otra aplicación que interprete sitios Web. En la siguiente imagen se puede ver una comparación entre la estructuración realizada con HTML (hasta la versión 4) y HTML 5:



Las Webs se dividirán en los siguientes elementos:

- **<section></section>**: Se utiliza para representar una sección "general" dentro de un documento o aplicación, como un capítulo de un libro. Puede contener subsecciones y si lo acompañamos de las etiquetas <h1>..<h6> podemos estructurar mejor toda la página creando jerarquías del contenido, algo muy favorable para el buen posicionamiento web. Por ejemplo:

```
<section>
   <h1>Introducción al elemento section</h1>
   <p>El elemento section se usa para agrupar contenido relacionado entre
si.</p>
   <p>...</p>
</section>
```

- **<article></article>**: Se usa para definir contenido autónomo e independiente, con la

intención de ser reutilizable de modo aislado. El elemento article puede contener uno o varios elementos section. Si por ejemplo nuestro contenido puede ser redistribuido como RSS y sigue manteniendo íntegro su significado, entonces, probablemente es un elemento article. De hecho, el elemento article está especialmente indicado para sindicación. El elemento article es especialmente útil para posts en blogs, noticias en prensa digital, comentarios y posts en foros.
La especificación de HTML5 añade además que el elemento article debe ser usado por widgets autónomos como; calculadoras, relojes, marcos de clima y cosas por el estilo. Hay que analizar si el contenido de un widget es autónomo, independiente y puede ser reutilizable o incluso sindicado.

- **<aside></aside>**: Representa una sección de la página que abarca un contenido no directamente relacionado con el contenido que lo rodea, por lo que se le puede considerar un contenido independiente. Dentro de este elemento pueden incluirse: elementos publicitarios, barras laterales, grupos de elementos de la navegación, efectos tipográficos, u otro contenido que se considere separado del contenido principal de la página.
- **<header></header>**: Es la cabecera de la página o de una sección. Existe una diferencia clave entre el elemento header y el uso habitual del término header (o cabecera) utilizado comúnmente para situar los elementos del encabezado de un sitio web.
  Una página web debe definir un header principal donde normalmente irá el logo o el nombre del sitio y seguramente un menú de navegación, pero además puede —y debe— definir otros elementos <header> dentro de los elementos <section>:

```
<section>
   <header>
      <h1>Cabecera se sección</h1>
   </header>
   <p>...</p>
</section>
```

- **<nav></nav>**: Contiene información sobre la navegación por el sitio web, usualmente una lista de enlaces. Este elemento debe de ser utilizado solo para la navegación principal del sitio y no para enlaces externos por ejemplo. Normalmente el elemento nav aparece dentro de un elemento header o footer.
- **<footer></footer>**: Representa el pié de una sección o la parte inferior de una página Web, contiene información acerca de la página/sección que poco tiene que ver con el contenido de la página, como el autor, el copyright, la fecha de última modificación, etc. Igual que con la etiqueta <header>, este elemento también se puede utilizar dentro de una sección para indicar información como: quien lo ha escrito, información de propiedad intelectual, enlaces, etc.

Es muy importante tener en cuenta que estas etiquetas no indican su posición en la página Web, sino su valor semántico. Por ejemplo, las etiquetas header, footer o aside no indican que esos elementos tengan que ir en la parte superior, inferior o lateral del contenido principal, sino que indican su función en esa sección o en esa página.

Además debemos tener en cuenta que estas nuevas etiquetas se comportan igual que una etiqueta de caja por lo que podemos aplicarles los mismos estilos CSS. Podemos redefinir la propia etiqueta o aplicarle una clase, por ejemplo:

```
header { width: 100%; padding: 10px; margin-bottom: 20px; }
.webheader { height: 30px; border: 1px solid gray; background-color:
silver; }
.sectionheader { font-size: 20px; }
```

1.4. Formularios

La estructura de los formularios con HTML 5 no varía con respecto a las anteriores de HTML. Pero sí que se añaden muchos nuevos tipos de campos que podemos utilizar, cada uno específico para cada tipo de dato.

En el caso de que utilicemos estas características y el navegador no sea compatible, simplemente las ignorará sin causarnos mayores problemas. También podemos detectar si el navegador soporta una determinada característica y en caso negativo emularla mediante código JavaScript (para más información ver la sección "Detectar funcionalidades de HTML5").

Los nuevos tipos de campos son:

- **search**: se utiliza para crear cajas de búsqueda. Tiene un aspecto similar a un campo de tipo texto. Además podemos utilizar el atributo *results="num"* para añadir un histórico de búsquedas con "num" resultados. De momento no funciona ni en Firefox ni en Chrome.

```
<label for="busqueda">Búsqueda con histórico: </label>
<input type="search" name="busqueda" id="busqueda" results="5"/>
```

- **number**: campo numérico, incorpora dos botones para para incrementar o decrementar el valor del campo. Además podemos usar atributos para asignar restricciones, como *min=""*, *max=""* o *step=""*. El valor es almacenado en el atributo *value=""*.

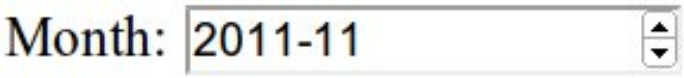
- **range**: campo numérico que permite seleccionar mediante una barra de desplazamiento un valor entre dos valores predeterminados, especificados mediante min="" y max="". El valor actual es almacenado en el atributo *value=""*. Además podemos indicar el incremento mínimo al desplazar la barra con *step=""*.

- **color**: permite seleccionar un color. De momento solo funciona en Opera 11.
- **tel**: es un campo de texto normal pero valida si el valor introducido es un número

telefónico (todavía no funciona).

- **url**: valida direcciones web. De momento requiere “http://” o “http:” simplemente. En algunos navegadores cambia el aspecto del cambio.
- **email**: valida direcciones de email. Funciona en algunos navegadores, mostrando además un aspecto diferenciado. Para iPhone además adapta el teclado.
- **date**: seleccionar un día en un calendario. En algunos navegadores (para móvil) aparece un calendario desplegable (como en Opera).

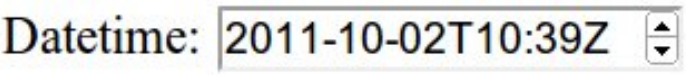
- **month**: selector para meses. En algunos navegadores (para móvil) aparece un calendario desplegable.



- **week**: selector para semanas. En algunos navegadores (para móvil) aparece un calendario desplegable.

- **time**: campo con formato para hora.

- **datetime**: permite seleccionar fecha y hora.



- **datetime-local**: permite seleccionar fechas y hora local.

- **output**: este campo se utiliza para visualizar valores, por ejemplo el valor de un campo “range”. De momento solo funciona en Opera. Se suele utilizar junto con la propiedad “onformchange” para actualizar su valor:

```
<output onformchange="value = rango.value">0</output>
```

Además, junto con estos nuevos tipos de campos, también se han incorporado nuevos tipos de atributos. Estos nuevos atributos son aplicables a la mayoría de los campos:

- **Autocomplete**: La mayoría de los navegadores incorporan la funcionalidad de autocompletar algunos campos de los formularios con valores introducidos anteriormente. Esta funcionalidad no siempre resulta útil, sobre todo si alguien nos roba nuestro portátil o dispositivo móvil. La nueva especificación de HTML5 nos permite desactivar el autocompletado en un formulario completo o solo en campos específicos. El atributo *autocomplete* nos permite definir dos valores: “on“ o “off“.

```
<form action="formaction.php" autocomplete="off">
```

```
  ...
</form>
```

El código anterior desactivaría el autocompletado de todo el formulario. Si por el contrario solo queremos desactivar el autocompletado de un solo campo podemos especificarlo así:

```
<input type="text" name="cuentadelbancosupersecreta" autocomplete="off" />
```

Esta funcionalidad no se puede emular mediante código JavaScript.

- **Placeholder**: El atributo *placeholder="texto"* se utiliza para colocar el valor de su texto dentro del campo a modo de ayuda. Si se focaliza dicho campo, se elimina el *placeholder*. Si abandonamos el campo sin añadir ningún valor, se vuelve a añadir el *placeholder*. Esta funcionalidad siempre ha requerido del uso de JavaScript para ser llevado a cabo, pero con la nueva especificación este comportamiento puede definirse de la forma:

```
<label for="referer">Nombre</label>
<input id="referer" name="referer" type="text"
                        placeholder="Escribe tu nombre completo" />
```

Obteniendo como resultado:

- **Required**: Una de las tareas de validación más extendidas es la de los campos requeridos. La nueva especificación de HTML5 incluye el atributo *required* que nos sirve para definir si un campo es requerido o no. Si un campo requerido está en blanco el formulario no será enviado y además avisará con un mensaje:

```
<label for="username">Su nombre de usuario</label>
<input id="username" name="username" type="text" required/>
```

NOTA: Es un error grave de seguridad validar los formularios únicamente desde el lado del cliente, es imprescindible además realizar la validación en el servidor.

- **Autofoco**: El atributo de autofoco asigna el foco (cursor de escritura) al campo indicado en cuando la página se ha cargado. Sólo se puede asignar a un elemento de la página. De momento este atributo solo lo soportan Safari, Chrome y Opera. Firefox e IE, lo ignoran, pero se puede emular fácilmente mediante código JavaScript (ver la siguiente sección "Detectar funcionalidades de HTML5").

```
<input name="b" autofocus/>
```

- **List**: Usando el atributo list con un elemento <input> podemos especificar una lista de opciones. Esto permite al usuario seleccionar un valor de la lista o escribir uno que no esté en ella (este tipo de elemento se suele llamar *Combo Boxes*). Los elementos de la lista se deben de indicar utilizando otro nuevo elemento de HTML5, el **<datalist>**. El cual simplemente nos permite crear una lista de valores. En algunos navegadores estas funcionalidades todavía no funcionan, como en Chrome.

```
<label for="diasemana">Día de la semana:</label>
```

```
<input type="text" name="diasemana" id="diasemana" list="dias"/>
<datalist id="dias">
   <option value="Lunes" />
   <option value="Martes" />
   <option value="Miércoles" />
   <option value="Jueves" />
   <option value="Viernes" />
   <option value="Sábado" />
   <option value="Domingo" />
</datalist>
```

Con este código obtendríamos un resultado similar al de la siguiente imagen:

- **Pattern (formatting)**: Este atributo se utiliza para validar la entrada del usuario mediante expresiones regulares. En la dirección "http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular" podemos obtener más información sobre las expresiones regulares. Ejemplo de uso (en Firefox y Chrome funciona):

```
<label for="cp">Código Postal</label>
<input id="cp" name="cp" pattern="[\d]{5}(-[\d]{4})" />
```

1.5. Mark

HTML5 también introduce un conjunto nuevo de elementos *inline*, solo que ya no se llaman elementos inline sino *text-level semantics* o semántica a nivel de texto. Uno de ellos es la etiqueta mark. Cuando realizamos una búsqueda en ciertos sitios, los elementos encontrados en la página aparecen remarcados para facilitar su localización. Hasta ahora el estilo se aplicaba con etiquetas , pero esta solución no es semántica. Es ahí donde entra en escena la nueva etiqueta <mark>:

```
<h1>Resultados de la búsqueda de la palabra 'anillo'</h1>
<ol>
   <li>El señor de los <mark>anillo</mark>s...</li>
   <li>el cliente compró este <mark>anillo</mark></li>
</ol>
```

Si queremos podemos redefinir el estilo de esta nueva etiqueta de la misma forma que lo hacíamos con las etiquetas de HTML, por ejemplo, para cambiar el color de fondo a rojo:

```
mark { background-color: red; }
```

1.6. Canvas

El elemento canvas puede definirse como un entorno para crear imágenes dinámicas. Utilizando su API en JavaScript podemos manipular el elemento canvas para dibujar en él y crear gráficos dinámicos de todo tipo (incluidas interfaces de aplicaciones web completas). La API, aunque de momento está en desarrollo, la podemos encontrar en: http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html

Para empezar a usarlo lo único que hay que especificar son sus dimensiones. El texto que escribamos entre la apertura y cierre de la etiqueta canvas solamente será interpretado por navegadores que no soporten esta etiqueta:

```
<canvas id="myCanvas" width="360" height="240">
   <p>Tu navegador no soporta canvas</p>
</canvas>
```

El resto de trabajo con canvas se ha de realizar con código JavaScript. Primero debemos referenciar este elemento y adquirir su contexto (que de momento solo está disponible para 2D):

```
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
```

Una vez adquirimos el contexto podemos empezar a dibujar. La API bidimensional ofrece muchas de las herramientas que podemos encontrar en cualquier aplicación de diseño gráfico: trazos, rellenos, gradientes, sombras, formas y curvas Bézier. Los principales métodos disponibles son:

- **fillRect(x, y, width, height)**: dibuja un rectángulo relleno de color según el estilo activado.
- **strokeRect(x, y, width, height)**: dibuja solo el borde de un rectángulo, el interior será transparente.
- **clearRect(x, y, width, height)**: borra el área indicada.
- **beginPath()**: inicializa el dibujado de un "trazo".
- **closePath()**: cierra la figura creando una línea desde el último punto hasta el primero.
- **moveTo(x, y)**: mueve el puntero del trazo hasta las coordenadas indicadas (para poder seguir dibujando).
- **lineTo(x, y)**: dibuja un trazo desde la posición actual hasta las coordenadas indicadas.
- **stroke()**: dibuja el trazo indicado desde el último "beginPath()".
- **fill()**: cierra el trazo definido desde el último "beginPath()" y lo rellena.
- **arc(x, y, radius, startAngle, endAngle, anticlockwise)**: dibuja un arco con centro en "x, y" y el radio definido. Los ángulos se definen en radianes (radianes = (PI/180)*grados) y el último parámetro es un valor booleano.
- **quadraticCurveTo(controlx, controly, x, y)**: dibuja una curva de bezier cuadrática.

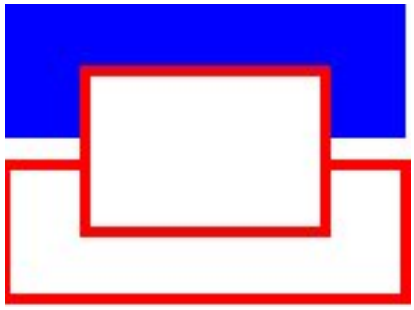
- **bezierCurveTo(control1x, control1y, control2x, control2y, x, y)**: dibuja una curva de bezier cúbica.
- **drawImage(x, y)**: dibuja una imagen (como objeto JavaScript) en el canvas.
- **createImageData(width, height)**: crea un objeto ImageData como un array de píxeles para ser manipulado como un array de enteros.
- **getImageData(x, y, w, h)**: carga un objeto ImageData a partir del dibujo actual para ser manipulado.
- **putImageData(imageData, x, y)**: mapea los valores de un objeto ImageData en el dibujo actual.
- **strokeText(string, x, y)**: dibuja una cadena de texto usando solo su borde.
- **fillText(string, x, y)**: dibuja una cadena de texto.

A continuación mostramos un ejemplo de dibujado en un objeto canvas una vez capturado su contexto:

```
// Primero definimos las propiedades con las que vamos a dibujar
context.fillStyle   = '#0000ff'; // color de relleno azul
context.strokeStyle = '#ff0000'; // color de borde rojo
context.lineWidth   = 4;         // grosor de línea

// Y a continuación dibujar algunas figuras
context.fillRect  (0,   0, 150, 50);   // rectángulo relleno
context.strokeRect(0,  60, 150, 50);   // rectángulo solo borde
context.clearRect (30, 25,  90, 60);   // borrar área del canvas
context.strokeRect(30, 25,  90, 60);   // Orden de coordenadas: izqda,
arriba, ancho, largo
```

Obteniendo finalmente un resultado similar a:



Webs muy importantes están cambiando sus contenidos a canvas y dejando de usar Flash, como Slideshare (ver http://www.slideshare.net/AmitRanjan/slideshare-is-html5-now).

1.7. Audio

El nuevo elemento audio permite insertar archivos sonoros en diferentes formatos, incluyendo mp3 y ogg. Además provee de una interfaz de control sobre la reproducción del mismo con una API en JavaScript sin necesidad de plugins de ningún tipo (como Flash). Añadir un reproductor de audio en HTML5 es muy simple:

```
<audio src="archivo.mp3" controls>
   <p>Tu navegador no soporta el elemento audio</p>
</audio>
```

En Firefox obtendríamos un resultado similar a:

El texto que se encuentra entre las etiquetas audio solo es tenido en cuenta por navegadores que no soporten la nueva etiqueta. El atributo “controls” indica al navegador que muestre los controles de reproducción. En caso de no activarlo no se visualizaría nada, pero podríamos controlar la reproducción mediante funciones JavaScript, de la forma:

```
<audio id="player" src="archivo.mp3"></audio>
<button
onclick="document.getElementById('player').play();">Reproducir</button>
<button
onclick="document.getElementById('player').pause();">Pausa</button>
<button onclick="document.getElementById('player').volume += 0.1;">Subir
Volumen</button>
<button onclick="document.getElementById('player').volume -= 0.1;">Bajar
Volumen</button>
```

También podemos usar los atributos “autoplay” y “loop” para que se auto-reproduzca y para que se cree un bucle de reproducción infinito:

```
<audio src="archivo.mp3" autoplay loop></audio>
```

El formato de audio a utilizar vendrá impuesto por el navegador usado y no por el estándar:

Códec	IE>=9	Firefox	Chrome	Safari	Opera
Ogg Vorbis	no	sí	sí	no	sí
WAV PCM	no	sí	sí	sí	sí
MP3	sí	no	sí	sí	sí
AAC	sí	no	sí	sí	sí
Speex	no	no	sí	no	no

Como puede verse, combinando Vorbis y MP3 podremos ofrecer audio a todos los navegadores mayoritarios. Existe una forma de definir más de un archivo de audio para la etiqueta audio, en lugar de usar el atributo “src”, utilizaremos la etiqueta “source” para poder definir múltiples archivos. Estas etiquetas se irán leyendo de arriba a abajo hasta

que el navegador encuentre un formato soportado. De esta manera podremos complacer las necesidades de todos los usuarios sin discriminar a ningún navegador.

```
<audio controls>
   <source src="archivo.ogg" type="audio/ogg" />
   <source src="archivo.mp3" type="audio/mpeg" />
   <object type="application/x-shockwave-flash"
data="player.swf?soundFile=archivo.mp3">
      <param name="movie" value="player.swf?soundFile=archivo.mp3" />
      <a href="archivo.mp3">Descarga el archivo de audio</a>
   </object>
</audio>
```

En este ejemplo hemos añadido además una tercera línea con un reproductor Flash por si no fuesen soportados ninguno de los formatos anteriores, y un link directo de descarga para aquellos que tampoco soporten Flash. Así estaremos ofreciendo nuestro contenido a todos los navegadores y dispositivos manteniendo unas buenas prácticas en cuanto a accesibilidad del contenido se refiere.

1.8. Vídeo

La nueva especificación de HTML5 soporta la inclusión de vídeo empotrado en las páginas web de forma nativa. El elemento *video* no especifica el formato del mismo sino que el uso de uno u otro vendrá impuesto por el fabricante del navegador:

Códec	IE>=9	Firefox	Chrome	Safari	Opera
Ogg Theora	no	sí	sí	no	sí
H.264	sí	no	no	sí	no
VP8	no	sí	sí	no	sí

El elemento *video* dispone de los atributos “autoplay”, “loop” y “preload”, para activar la auto-reproducción, para indicar que se reproduzca en bucle y para activar/desactivar la precarga del vídeo. Asimismo puedes utilizar los controles que te ofrece el navegador de forma nativa utilizando el atributo *controls* o bien puedes ofrecer tus propios controles en JavaScript. Dado que el vídeo ocupa un espacio, también podremos definir sus dimensiones con los atributos “width” y “height”. E incluso podemos indicar una imagen para que se muestre antes de la reproducción mediante el atributo “poster”:

```
<video src="archivo.mp4" controls width="360" height="240"
poster="poster.jpg"> </video>
```

Con lo que obtendríamos un resultado similar a:

Para dar soporte a todos los navegadores, podemos especificar diferentes archivos en diferentes formatos. Además podemos usar el mismo truco que usábamos con el elemento audio para seguir dando soporte al *plugin* de Flash a través de la etiqueta *object*, e incluso incluir un link de descarga:

```
<video controls width="360" height="240" poster="poster.jpg">
   <source src="archivo.ogv" type="video/ogg" />
   <source src="archivo.mp4" type="video/mp4" />
   <object type="application/x-shockwave-flash" width="360" height="240"
data="player.swf?file=archivo.mp4">
      <param name="movie" value="player.swf?file=archivo.mp4" />
      <a href="archivo.mp4">Descarga la película</a>
   </object>
</video>
```

1.9. Geolocalización

La geolocalización es la forma de obtener tu posición en el mundo y si quieres, compartir esta información. Existen muchas maneras de descubrir donde te encuentras, por tu dirección IP, la conexión de red inalámbrica, la torre de telefonía móvil por la que se conecta tu móvil, o usando directamente el posicionador GPS.

HTML5 incorpora una nueva funcionalidad para facilitar esta tarea, que dependerá de que el navegador le de soporte. Está disponible a partir de las versiones de Opera 10.6, Firefox 3.5, Chrome 5, Safari 5 e Internet Explorer 9.

```
if (navigator.geolocation)
{
```

```
   navigator.geolocation.getCurrentPosition(showPosition);
}

function showPosition( position )
{
      var lat = position.coords.latitude;
      var lng = position.coords.longitude;

      alert( "Latitud: " + lat + ", longitud: " + lng );
}
```

1.10. Almacenamiento Offline

El almacenamiento web está ampliamente soportado por los navegadores modernos, tanto en plataforma escritorio como en plataforma móvil, Android 2.1+, iPhone 3.1+, iPad 4.2+, Opera Mobile 11.00+, Palm WebOS 1.4+ y BlackBerry 6.0+, Crome 4.0+, Firefox 3.5+, IE 8.0+, Opera 10.5+ y Safari 4.0+.

Tipos de almacenamiento

El almacenamiento web ofrece dos áreas de almacenamiento diferentes, el almacenamiento local (localStorage) y el almacenamiento por sesión (sessionStorage), que difieren en alcance y tiempo de vida. Los datos alojados en un almacenamiento local es solo accesible por dominio y persiste aún cuando se cierre el navegador. El almacenamiento por sesión es por ventana y su tiempo de vida está limitado a lo que dure la ventana (o pestaña).

Los datos se almacenan de forma muy sencilla, por pares clave/valor, de la forma:

```
// Para almacenamiento persistente en local:
localStorage.setItem("miValor", valor);

// Para almacenamiento por sesión:
sessionStorage.setItem("miValor", valor);
```

Para recuperarlos posteriormente solo tenemos que hacer:

```
var miValor = localStorage.getItem("miValor");
var miValor = sessionStorage.getItem("miValor");
```

Las variables guardadas con sessionStorage sólo se mantendrían en caso de que cambiemos de página o que el navegador se refresque, mientras que localStorage guardaría los datos aunque el navegador sea cerrado.

También podemos borrar los valores almacenados, indicando un valor en concreto o todos ellos:

```
localStorage.remove("miValor");
localStorage.clear();
```

Offline Application Cache (appCache)

Esta nueva característica de HTML5 permite ejecutar aplicaciones Web aun cuando no estamos conectados a Internet. Al visitar por primera vez una página web (que use appCache) el navegador descarga y guarda todos los archivos necesarios para esa página. La siguiente vez que la visitemos el navegador usará directamente los archivos descargados (a no ser que estemos conectados y se compruebe que hay una versión más actual de la Web).

El principal componente del appCache es el archivo de manifiesto (manifest file), un archivo de texto con la lista de archivos que el navegador cliente debe almacenar. En primer lugar, para usar esta característica debemos de indicar el archivo de manifiesto en la etiqueta de apertura HTML:

```
<html manifest="app.manifest">
```

Este fichero debe de empezar con el texto CACHE MANIFEST. A continuación en cada nueva línea indicaremos un recurso a almacenar (usando URLs absolutas o relativas), además podemos poner comentarios anteponiendo el símbolo “#”.

```
CACHE MANIFEST
# Esto es un comentario
index.html
js/scripts.js
css/estilos.css
imgs/logo.gif
imgs/image1.jpg
```

Una vez cargada la página, la única petición que realizará el navegador será por el fichero de *manifiest*. Aunque solo haya cambiado un letra del fichero, se descargarán todos los recursos de nuevo. Para asegurarnos que servimos la última versión de nuestra página cuando realizamos cambios, la forma más sencilla y segura es actualizar el fichero de manifiesto con un comentario indicando la fecha de la última actualización (o un número de versión, etc.), de la forma:

```
CACHE MANIFEST
# Actualizado el 2011-10-12
```

Para más información podéis consultar las fuentes:

- http://www.w3.org/TR/offline-webapps/
- http://www.w3.org/TR/html5/offline.html

1.11. Detectar funcionalidades de HTML5

Modernizr es una librería de JavaScript con licencia MIT de código abierto que detecta si son compatibles elementos de HTML5 y CSS3. Podemos descargar la librería desde “http://www.modernizr.com/”. Para utilizarla solo hay que incluir en el <head> de tu página de la forma:

```
<head>
   <script src="modernizr.min.js"></script>
</head>
```

Modernizr se ejecuta automáticamente, no es necesario llamar a ninguna función. Cuando se ejecuta, se crea una objeto global llamado *Modernizr*, que contiene un *set* de propiedades *Boleanas* para cada elemento que detecta. Por ejemplo si su navegador soporta elementos *canvas*, la propiedad de la librería "*Modernizr.canvas*" será "*true*". Si tu navegador no soporta los elementos *canvas*, la propiedad será "*false*", de la forma:

```
if (Modernizr.canvas) {
   // sí que hay soporte
} else {
   // no hay soporte para canvas
}
```

Para comprobar elementos de un formulario también podemos crearnos dos simples funciones que validan el soporte para diferentes tipos de inputs y atributos:

Comprobar si un input es soportado

Con la siguiente función podemos comprobar si un navegador soporta o no los nuevos tipos de inputs:

```
function inputSupports(tipo) {
   var input = document.createElement('input');
   input.setAttribute('type', tipo);
   if (input.type == 'text') {
      return false;
   } else {
      return true;
   }
}
```

Por lo que podemos usarlo de la siguiente forma:

```
if (!inputSupports('range')) {
   // Input tipo range no soportado
}
```

Comprobar si un atributo es soportado

Para comprobar si hay soporte para un atributo

```
function attrSupports(el, attr) {
   var telement = document.createElement(el);
   if (attr in telement) {
      return true;
   } else {
      return false;
   }
}
```

Por lo que podemos usarlo para comprobar, por ejemplo, los atributos autofocus,