

HTML

CSS



1. Objetivos

En este tercer tema se pretenden conseguir los siguientes objetivos:
Conocer los diferentes códecs de vídeo y audio utilizados en la web por los distintos navegadores.

- ✚ Utilizar los nuevos elementos HTML5 para audio y vídeo.
- ✚ Insertar JavaScript en nuestros documentos.
- ✚ Crear manejadores de eventos JavaScript de forma estándar.
- ✚ Seleccionar elementos de la página con los nuevos selectores JavaScript. Utilizar el nuevo api de manejo de elementos multimedia que incorpora HTML5.
- ✚ Crear efectos web utilizando las transformaciones y transiciones de CSS3.

2. Introducción

Ya en el tema 2 vimos algunas de las nuevas características que nos aportan tanto HTML5 como CSS3. En este tercer tema, nos vamos a centrar en tres aspectos fundamentales:

- 🚦 La importancia que tiene JavaScript para la web en general y para HTML5 en particular.
- 🚦 Cómo se manejan los elementos multimedia (principalmente audio y vídeo) con HTML5.
- 🚦 Cómo realizar pequeñas animaciones sin necesidad de utilizar flash, ni JavaScript, ni librerías externas como jQuery.

Evidentemente, por motivos obvios de tiempo, nuestro acercamiento a JavaScript en este curso es introductorio, sólo vamos a repasar las técnicas que necesitamos entender para poder realizar algunos ejemplos y ejercicios, además de ver algunos aspectos nuevos que aparecen con el estándar HTML5.

En lo que se refiere a la parte de contenidos multimedia hablaremos sobre los distintos formatos de audio y vídeo soportados por la web, y cómo trabajar con estos formatos en nuestras páginas, pero no hablaremos sobre temas como conversión de vídeo y cosas de ese estilo.

Finalmente, para seguir avanzando con CSS3, hablaremos de las nuevas características para realizar transformaciones y transiciones, y veremos un ejemplo práctico de uso de las mismas para crear menús desplegables. En el siguiente tema, veremos las novedades que incorpora HTML5 en el manejo de formularios y más funcionalidades nuevas de CSS3, como son las fuentes incrustadas, múltiples imágenes de fondo, etc.

3. HTML5

HTML5 puede ser imaginado como un edificio soportado por tres grandes columnas: HTML, CSS y JavaScript. Ya hemos estudiado los elementos estructurales incorporados en HTML y algunas de las nuevas propiedades que hacen de CSS la herramienta ideal para diseñadores. Ahora es momento de desvelar lo que puede ser considerado como uno de los pilares más fuertes de esta especificación: JavaScript.

La importancia de JavaScript

Para aprovechar esta prometedora plataforma de trabajo ofrecida por los nuevos navegadores, JavaScript se amplió para mejorar la portabilidad e integración. A la vez, se incorporaron interfaces de programación de aplicaciones (APIs) a cada navegador para asistir al lenguaje en funciones elementales. Estas nuevas APIs (como Web Storage, Canvas y otras) son interfaces para librerías incluidas en navegadores. La idea es poner a disposición del desarrollador web poderosas funciones a través de técnicas de programación sencillas y estándares, expandiendo el alcance del lenguaje y facilitando la creación de programas útiles para la web.

En este capítulo, estudiaremos cómo incorporar JavaScript dentro de nuestros documentos HTML y veremos las nociones básicas para trabajar con este lenguaje.

Incorporando JavaScript al documento

Existen tres técnicas para incorporar código JavaScript dentro de HTML. Sin embargo, al igual que ocurre con CSS, en HTML5 se recomienda

usar la inclusión de archivos externos.

En línea

Esta es una técnica simple para insertar JavaScript en nuestro documento que se aprovecha de atributos disponibles en elementos HTML.

Estos atributos son manejadores de eventos que ejecutan código de acuerdo a la acción del usuario. Los manejadores de eventos más usados son, en general, los relacionados con el ratón (onclick, onMouseOver, onMouseOut, etc).

Sin embargo, encontraremos sitios web que implementan eventos de teclado y otros, ejecutando acciones cuando se pulsa una tecla, se carga la página, etc.

`<p onclick="alert('has hecho clic!')">Hacer Clic</p>`

En el ejemplo, usando el manejador de eventos onclick, se ejecuta el código cada vez que el usuario hace clic con el ratón sobre el texto que dice "Hacer Clic". Si cambiáramos el manejador onclick por onMouseOver, por ejemplo, el código se ejecutaría al pasar el puntero del ratón sobre el elemento.

El uso de JavaScript dentro de etiquetas HTML está permitido en HTML5, pero por las mismas razones que en CSS, este proceder no es recomendable. El código HTML se extiende innecesariamente y se hace difícil de mantener y actualizar.

Empotrado

Podemos agrupar el código JavaScript en un mismo lugar entre etiquetas `<script>`. El elemento `<script>` actúa exactamente igual que el elemento `<style>` usado para incorporar estilos CSS. Nos ayuda a organizar el código en un solo lugar, afectando a los elementos HTML por medio de referencias. Para obtener el mismo resultado que en el ejemplo anterior, haríamos lo siguiente:

```
<script>
function mostraralerta(){
    alert('hizo clic!');
}

function hacerclic(){
    document.getElementsByTagName('p')[0].onclick=mostraralerta();
}

window.onload=hacerclic();
</script>
```

Actualmente, existen tres métodos disponibles para referenciar elementos HTML desde JavaScript:

- 🚦 **getElementsByTagName**: referencia uno o varios elementos por su nombre o palabra clave.
- 🚦 **getElementById**: referencia un elemento por el valor de su atributo id.
- 🚦 **getElementsByClassName**: es una nueva incorporación que nos permite referenciar uno o varios elementos por el valor de su atributo class.

Debemos tener en cuenta que el código del documento se lee de forma secuencial por el navegador, y no podemos referenciar un elemento que aún no ha sido creado. Por este motivo, es una buena práctica colocar los scripts al final del documento siempre que sea posible.

En el ejemplo anterior, tenemos una función llamada `mostraralerta()`, y la

referencia al elemento `<p>` junto con el manejador del evento fueron colocados en una segunda función llamada `hacer clic()`. Las funciones son llamadas desde la última línea del código usando otro manejador de eventos (en este caso asociado con la ventana) llamado **onload**. Este manejador ejecutará la función `hacer clic()` cuando el documento esté completamente cargado y todos los elementos creados.

El código del ejemplo se ejecuta de la siguiente forma:

Primero, las funciones JavaScript se cargan (declaran), pero no se ejecutan.

Luego, los elementos HTML, incluidos los elementos `<p>`, se crean.

Finalmente, cuando el documento completo se carga en la ventana del navegador, se dispara el evento `load` y se llama la función `hacer clic()`.

En esta función, el método **getElementsByTagName** referencia todos los elementos `<p>`. Este método devuelve un array que contiene los elementos de la etiqueta especificada encontrados en el documento. Sin embargo, usando el índice `[0]` al final del método, indicamos que sólo trabajamos con el primer elemento devuelto y registramos el manejador de eventos `onclick` para el mismo.

La función `mostrar alerta()` se ejecutará cuando el evento `click` se dispare sobre este elemento, mostrando el mensaje "hizo clic!". Puede parecer mucho código y trabajo para reproducir el mismo efecto logrado por una simple línea en el ejemplo anterior, pero considerando el potencial de HTML5 y la complejidad alcanzada por JavaScript, la concentración del código en un único lugar y la apropiada organización representa una gran ventaja para nuestras futuras implementaciones y para hacer nuestros sitios web y aplicaciones

fáciles de desarrollar y mantener.

Archivos externos

El código JavaScript crece exponencialmente cuando añadimos nuevas funciones y aplicamos algunas de las APIs mencionadas previamente.

El código empotrado incrementa el tamaño de nuestros documentos y los hace repetitivos (cada documento debe volver a incluir el mismo código). Para reducir los tiempos de descarga, incrementar nuestra productividad y poder distribuir y reutilizar nuestro código en cada documento sin comprometer eficiencia, es recomendable trasladar todo el código JavaScript a uno o más archivos externos y llamarlos usando el atributo src:

`<script src="./js/micodigo.js"></script>`

El elemento `<script>` del ejemplo carga el código JavaScript desde un archivo externo llamado `micodigo.js`. Si cogemos el código del ejemplo anterior y lo copiamos a este archivo, veremos que sigue funcionando perfectamente. Normalmente esta línea la pondremos justo antes del fin de la etiqueta del `body`.

Nuevos Selectores

El método `querySelector()`

Este método retorna el primer elemento que concuerda con el grupo de selectores especificados entre paréntesis. Los selectores son declarados usando comillas y la misma sintaxis CSS, como en el siguiente ejemplo:

```
function hacerclic(){
    document.querySelector("body p:firstchild").onclick=mostraralerta;
}

function mostraralerta(){
    alert('hizo clic!');
}

window.onload=hacerclic();
```

En el ejemplo, el método `getElementsByTagName` usado anteriormente ha sido reemplazado por `querySelector()`.

Los selectores para esta consulta en particular están referenciando al primer elemento `<p>` que es hijo del elemento `<body>`.

Como ya hemos comentado, este método sólo retorna el primer elemento encontrado, por lo tanto, la pseudo-clase **first-child** es redundante. El método `querySelector()` de nuestro ejemplo retornará el primer elemento `<p>` dentro del `<body>`, que es, por supuesto, su primer hijo. En cualquier caso, el propósito de este ejemplo es comprobar que `querySelector()` acepta toda clase de selectores CSS válidos, y ahora, del mismo modo que en CSS, JavaScript provee herramientas importantes para referenciar cada elemento en el documento.

Pueden declararse varios grupos de selectores separados por comas. El método `querySelector()` devolverá el primer elemento que concuerde con cualquiera de ellos.

El método `querySelectorAll()`

El método `querySelectorAll()` devuelve todos los elementos que concuerdan con el grupo de selectores declarados entre paréntesis, en lugar de sólo el primero como hacía el método anterior. El valor devuelto es un array que contiene cada elemento encontrado en el orden en el que aparecen en el documento.

```
function hacerclic(){
    var lista=document.querySelectorAll("body p");
    lista[0].onclick=mostraralerta;
}

function mostraralerta(){
    alert('hizo clic!');
}

window.onload=hacerclic;
```

El grupo de selectores especificados en el método `querySelectorAll()` del ejemplo seleccionará cada elemento `<p>` en el documento HTML que sea hijo del elemento `<body>`. El array `lista` contendrá todos los elementos `<p>` del documento. Normalmente, cuando utilizemos este método para obtener un array de elementos, lo recorreremos utilizando para ello un bucle `for`:

```
function hacerclic(){
    var lista=document.querySelectorAll("body p");
    for(var i=0; i<lista.length; i++){
        lista[i].onclick=mostraralerta;
    }
}

function mostraralerta(){
    alert('hizo clic!');
}

window.onload=hacerclic();
```

En este otro ejemplo, en lugar de seleccionar sólo el primer elemento encontrado, registramos el manejador de eventos onclick para cada uno de ellos usando un bucle for. Ahora, todos los elementos <p> del <body> mostrarán una pequeña ventana cuando el usuario haga clic sobre ellos.

El método `querySelectorAll()`, al igual que `querySelector()`, puede contener uno o más grupos de selectores separados por comas.

Manejadores de eventos

Como comentamos anteriormente, el código JavaScript se ejecuta, normalmente, después de que el usuario realiza alguna acción; estas acciones y otros eventos son procesados por manejadores de eventos y funciones JavaScript asociadas con ellos.

Existen tres formas diferentes de registrar un evento para un elemento HTML:

- 🚦 Podemos agregar un nuevo atributo al elemento.
- 🚦 Registrar un manejador de evento como una propiedad del elemento.
- 🚦 Usar el nuevo método estándar **`addEventListener()`**.

Manejadores de eventos en línea

Ya utilizamos esta técnica en el primer ejemplo anterior, incluyendo el atributo onclick en el elemento <p>. Se trata, simplemente, de utilizar los atributos provistos por HTML para registrar eventos sobre un elemento en particular. Esta es una técnica en desuso, pero puede resultar práctica en algunas circunstancias, especialmente cuando necesitamos hacer modificaciones rápidas para testeo.

Manejadores de eventos como propiedades

Para evitar las complicaciones de la técnica en línea (inline), debemos registrar los eventos desde el código JavaScript. Usando selectores JavaScript, podemos referenciar el elemento HTML y asignarle el manejador de eventos que queremos como si fuese una propiedad.

Ya vimos un ejemplo anterior que utilizaba esta técnica cuando registramos el manejador de eventos onload para la ventana usando la construcción window.onload, y el manejador de eventos onclick para el primer elemento <p> encontrado en el documento con la línea de código **document.getElementsByTagName('p')[0].onclick.**

Los nombres de los manejadores de eventos son contruidos agregando el prefijo on al nombre del evento. Por ejemplo, el nombre del manejador de eventos para el evento click es onclick.

Antes de HTML5, esta era la única técnica disponible para usar manejadores de eventos desde JavaScript que funcionaba en todos los navegadores. Algunos fabricantes de navegadores estaban desarrollando sus propios sistemas, pero nada se adoptó por todos hasta que el nuevo estándar

HTML5 se declaró.

El método addEventListener()

El método addEventListener() es la técnica ideal y la que es considerada como estándar por la especificación de HTML5.

```
function mostraralerta() {  
    alert('hizo clic!');  
}
```

```
function hacerclic() {  
    var elemento=document.querySelector("body p:firstchild");  
    elemento.addEventListener('click', mostraralerta, false);  
}
```

```
window.addEventListener('load', hacerclic, false);
```

Este ejemplo, es equivalente a los anteriores, pero ahora se asocian los manejadores de evento usando el método addEventListener(). Para organizar el código en la función hacerclic(), asignamos la referencia al elemento a una variable llamada elemento, y luego agregamos la escucha para el evento click usando esa variable.

El método addEventListener() recibe tres parámetros:

- 🚦 El nombre del evento sin el prefijo on.
- 🚦 La función a ser ejecutada, la cual puede ser una referencia a una función (como en este caso) o una función anónima.
- 🚦 Un valor booleano (falso o verdadero) que indica cómo será disparado un evento en elementos superpuestos. Por ejemplo, si estamos escuchando al evento click en dos elementos que se encuentran anidados (uno dentro de otro), cuando el usuario hace clic sobre estos elementos, dos eventos

click se disparan en un orden que depende de este valor. Si el atributo se declara como true para uno de los elementos, entonces ese evento será considerado primero y el otro luego. Normalmente, el valor false es el más adecuado para la mayoría de las situaciones.

Multimedia en HTML5

En este punto estudiaremos las posibilidades que nos ofrece HTML5 para trabajar con elementos multimedia, en concreto, veremos cómo trabajar con vídeo y con audio.

Reproduciendo vídeo en HTML5

Hoy en día es de lo más común descargar y ver vídeo dentro del navegador web. Conforme el ancho de banda comenzó a hacerse realmente ancho, la popularización del video a través de una conexión de datos se hizo posible; la información dejó de ser sólo texto e imágenes pequeñas o ultra comprimidas y la web se tornó realmente multimedia.

HTML5 introdujo un elemento para insertar y reproducir vídeo en un documento HTML. El elemento `<video>` usa etiquetas de apertura y cierre, además, sólo necesita unos pocos parámetros para lograr su función. La sintaxis es extremadamente sencilla y sólo el atributo `src` es obligatorio:

```
<video src="./video.mp4" controls>  
</video>
```

El código anterior debería ser suficiente, pero, como explicamos anteriormente, las cosas se vuelven un poco más complicadas en la vida real.

"Anatomía" de un Video Digital

Un video digital está compuesto de diversas pistas o tracks, de las cuales una es de vídeo y una o más de audio. Las pistas de audio y vídeo se

encuentran encapsuladas en contenedores que definen el formato del vídeo y son codificadas mediante algoritmos llamados códecs. La palabra códec proviene de la contracción de las palabras coder y decoder (codificador y decodificador), aunque también se denomina códec al software o hardware encargado de realizar la codificación y decodificación de la pista.

Los archivos de video de tipo AVI, MOV, MP4, M4V u otro similar son contenedores de video. Existen diversas aplicaciones reproductoras de video que dan soporte a uno o más de estos formatos (por ejemplo, Windows Media Player da soporte, principalmente, a AVI y QuickTime de Apple es capaz de reproducir videos en MOV, MP4 y otros), sin embargo eso no es suficiente para poder reproducir un vídeo, y para esto, es necesario que la aplicación cuente también con el códec apropiado, es decir, con el mismo con el cual fue codificado.

Contenedores

Como se mencionó antes, un contenedor define el formato y contiene las pistas de vídeo y audio. Los contenedores soportados por HTML5 son los siguientes:

MP4 (MPEG-4 Part 14)

Especificado por el MPEG (Moving Picture Experts Group), este estándar se encuentra presente en diversos medios: desde sistemas de TV por cable, reproductores de discos Blu-ray, smartphones, películas y series de TV de compra y alquiler en la tienda iTunes de Apple, etc. HTML5 soporta este contenedor en varios de los navegadores principales, generalmente, con una extensión .mp4 o .m4v.

OGG

Este estándar abierto está soportado por HTML5 en varios navegadores bajo la extensión .ogv. OGG soporta vídeo codificado con Theora y audio con Vorbis.

WebM

La especificación de este contenedor, apoyada por Google, es abierta y utiliza también Vorbis para codificar sus pistas de audio y el códec VP8 para video. HTML5 soporta este contenedor utilizando la extensión .webm.

Códecs de vídeo para HTML5

Los siguientes códecs de vídeo son válidos en HTML5, aunque no todos ellos son soportados por cada uno de los principales navegadores.

H.264 (MPEG-4 Part 10 o MPEG-4 AVC)

El estándar de codificación de video H.264 fue desarrollado por el MPEG (Moving Picture Experts Group) y es uno de los formatos más utilizados actualmente en diversas aplicaciones, desde video en sitios web, consolas de videojuegos, sistemas de TV por cable, etc.

La utilización de este códec requiere licencia del MPEG LA Group.

Theora

Basado en VP3, este estándar fue cedido en 2002 a Xiph.org por parte de On2 con licencia de uso libre (royalty-free). Theora está soportado por HTML5 en Mozilla Firefox, Google Chrome y Opera en un contenedor OGG.VP8

Al igual que VP3 (después Theora), VP8 fue desarrollado por On2 Technologies, empresa adquirida por Google en 2010. VP8 es un códec de estándar abierto utilizado para codificar vídeo, normalmente, contenido en el formato WebM.

Códecs de Audio para HTML5

Los siguientes códecs de audio son válidos en HTML5, aunque, como ocurre con los códecs de vídeo, no todos ellos son soportados por todos los navegadores. AAC (Advanced Audio Coding) AAC es un estándar de codificación de audio digital utilizado en una gran variedad de dispositivos: reproductores de música, teléfonos, consolas de videojuegos, etc. Puede ser utilizado como pista de audio en un vídeo en conjunto con el códec H.264 en un contenedor MP4.

AAC requiere licencia para su uso. Vorbis es un códec de estándar libre que se utiliza comúnmente para codificar sonido dentro de vídeos de formato OGG y WebM, aunque es más habitual encontrarlo en el primero. MP3 (MPEG-1 Audio Layer 3) El estándar de codificación de audio MP3 es uno de los más populares dentro del mundo de la música digital. Este códec puede ser utilizado también en conjunto con el H.264 en un contenedor MP4 para reproducir vídeo mediante HTML5 en ciertos navegadores. El estándar requiere licencia de uso.

Soporte de formatos y códecs por parte de los navegadores

La siguiente tabla muestra una relación de los formatos y códecs más importantes, su aplicación y soporte actual en HTML5:

Contenedor	Extensión		Licencia de Contenedor	Codec	Tipo	Licencia de Codec	Soporte Nativo	
	HTML5	Otros					HTML5	Otros
MP4	.mp4	.mov .m4v	Requiere licencia	AAC	Audio	Requiere licencia	Chrome (3.0+) Safari (3.1+) IE (9.0)	Reproductores de discos Blu-ray iTunes
				H.264	Video	Requiere licencia	Android iOS	QuickTime Player TV por cable
				MP3	AUDIO	Requiere licencia	Safari (3.1+)	Apple TV
				H.264	Video	Requiere licencia	IE (9.0)	PS3 Xbox 360
OGG	.ogv	--	Gratuita	Vorbis	Audio	Gratuita	Firefox (3.5+) Opera (10.5+)	VLC XMMS (Linux)
				Theora	Video	Gratuita	Chrome (3.0+)	
WebM	.webm	--	Gratuita	Vorbis	Audio	Gratuita	Firefox (4.0+) Opera (10.6+)	VLC
				VP8	Video	Gratuita	Chrome (3.0+) Android (2.3+)	

Es importante hacer notar que no existe una sola combinación formato/códec de vídeo/códec de audio, a la cual, todos los navegadores modernos le den soporte, por este motivo, es necesario realizar la codificación en varios formatos. Existen numerosas herramientas que nos permiten recodificar un archivo de vídeo, dos ejemplos de aplicaciones gratuitas serían:

- 🔧 HandBrake (<http://handbrake.fr/>): aplicación de escritorio.
- 🔧 Firefogg (<http://www.firefogg.org/>): complemento de Firefox.

El elemento <video>

El elemento <video> ofrece varios atributos para establecer su comportamiento y configuración:

- Los atributos `width` y `height`, al igual que en otros elementos HTML ya conocidos, declaran las dimensiones para el elemento o ventana del reproductor. El tamaño del vídeo será automáticamente ajustado para entrar dentro de estos valores, pero no fueron considerados para redimensionar el vídeo, sino para limitar el área ocupada por el mismo y mantener la consistencia en el diseño.
- El atributo `src` indica la fuente del vídeo. Este atributo puede ser reemplazado por el elemento `<source>` y su propio atributo `src` para declarar varias fuentes con diferentes formatos, como en el siguiente ejemplo:

```
<video id="medio" width="720" height="400" controls>  
<source src="video.mp4">  
<source src="video.ogg">  
</video>
```

En el ejemplo anterior, el elemento `<video>` fue expandido. Ahora, dentro de las etiquetas del elemento hay dos elementos `<source>`. Estos nuevos elementos proveen diferentes fuentes de vídeo para que los navegadores puedan elegir. El navegador leerá la etiqueta `<source>` y decidirá qué archivo reproducir de acuerdo a los formatos que soporte.

Atributos para `<video>`

Podemos ver en los ejemplos anteriores que hemos añadido un atributo `controls`, este atributo muestra controles de vídeo provistos por el navegador.

Cuando el atributo está presente cada navegador activará su propia interface de control del vídeo, permitiendo al usuario comenzar a reproducir el vídeo, pausarlo o saltar hacia un cuadro específico, entre otras funciones.

Junto con controls, también podemos usar los siguientes atributos:

- 🎨 **autoplay**: cuando este atributo está presente, el navegador comenzará a reproducir el vídeo automáticamente, tan pronto como pueda.
- 🎨 **loop**: si este atributo es especificado, el navegador comenzará a reproducir el vídeo nuevamente cuando llegue al final.
- 🎨 **poster**: este atributo es utilizado para proveer una imagen que será mostrada mientras esperamos que el vídeo comience a ser reproducido.
- 🎨 **preload**: este atributo puede recibir tres valores distintos:
 - none: indica que el vídeo no debería ser cacheado, por lo general, con el propósito de minimizar tráfico innecesario.
 - metadata: recomendará al navegador que trate de capturar información acerca de la fuente (por ejemplo, dimensiones, duración, primer cuadro, etc.).
 - auto: es el valor configurado por defecto, que le sugerirá al navegador descargar el archivo tan pronto como sea posible.

```
<video id="reproductor" width="720" height="400" preload controls loop  
poster="./imgs/poster.jpg">  
<source src="./videos/video.mp4">  
<source src="./videos/video.ogg">  
</video>
```

Reproduciendo audio con HTML5

HTML5 provee un nuevo elemento para reproducir audio en un documento HTML. El elemento, por supuesto, es `<audio>` y comparte casi las mismas características del elemento `<video>`.

```
<audio src="cancion.mp3" controls>  
</audio>
```

El elemento `<audio>`

Como ya hemos comentado, el elemento `<audio>` trabaja del mismo modo y comparte varios atributos con el elemento `<video>`:

- 🚦 **src:** Este atributo especifica la URL del archivo a ser reproducido. Al igual que en el elemento `<video>`, normalmente, será reemplazado por el elemento `<source>` para ofrecer diferentes formatos de audio entre los que el navegador pueda elegir.
- 🚦 **controls:** Este atributo activa la interface que cada navegador provee por defecto para controlar la reproducción del audio.
- 🚦 **autoplay:** Cuando este atributo está presente, el audio comenzará a reproducirse automáticamente tan pronto como sea posible.
- 🚦 **loop:** Si se indica este atributo, el navegador reproducirá el audio una y otra vez de forma automática.
- 🚦 **preload:** Este atributo puede tomar tres valores diferentes:
 - **none:** indica que el audio no debería ser cacheado, normalmente, con el propósito de minimizar tráfico innecesario.
 - **metadata:** recomendará al navegador obtener información sobre

el medio (por ejemplo, la duración).

- **auto:** es el valor configurado por defecto y le aconseja al navegador descargar el archivo tan pronto como sea posible.

Una vez más, debemos hablar acerca de codificadores. MP3 está bajo licencia comercial, por lo que, no es soportado por navegadores como Firefox u Opera. Vorbis (el codificador de audio del contenedor OGG) es soportado por estos navegadores, pero no por Safari e Internet Explorer. Por esta razón, nuevamente, debemos aprovechar el elemento `<source>` para proveer al menos dos formatos, entre los cuales el navegador pueda elegir.

```
<audio id="medio" controls>  
  <source src="cancion.mp3">  
  <source src="cancion.ogg">  
</audio>
```

El código anterior reproducirá música en todos los navegadores utilizando los controles por defecto. Aquellos que no puedan reproducir MP3 reproducirán OGG y viceversa. Debemos recordar que MP3, al igual que MP4 para vídeo, tienen uso restringido por licencias comerciales, por lo que sólo podemos usarlos en circunstancias especiales, de acuerdo con lo determinado por cada licencia.

El soporte para los codificadores de audio libres y gratuitos (como Vorbis) se está expandiendo, pero llevará tiempo transformar este formato desconocido en un estándar .

Tipos MIME

Los tipos MIME (Internet Media Types) son una forma de definir formatos de archivos para que nuestro sistema sepa cómo manejarlos. Estos tipos los podemos manejar a dos niveles:

- 🚦 Desde el Servidor: primero nuestro servidor debe ser configurado para servir correctamente los tipos MIME. En el caso de tener un servidor Apache, debemos incluir lo siguiente en el archivo `.htaccess`:

#AddType TYPE/SUBTYPE EXTENSION

AddType audio/aac .aac

AddType audio/mp4 .mp4 .m4a

AddType audio/mpeg .mp1 .mp2 .mp3 .mpg .mpeg

AddType audio/ogg .oga .ogg

AddType audio/wav .wav

AddType audio/webm .webm

AddType video/mp4 .mp4 .m4v

AddType video/ogg .ogv

AddType video/webm .webm

- 🚦 Desde el Cliente: Cuando definimos fuentes en nuestro código, podemos ayudar a nuestro navegador en la identificación del contenido especificando el tipo MIME utilizado.


```
<audio>
  <source src="elvis.mp3" type='audio/mpeg; codecs="mp3"' >
  <source src="elvis.ogg" type='audio/ogg; codecs="vorbis"' >
</audio><video poster="star.png" autoplay loop controls tabindex="0">
  <source src="movie.webm" type='video/webm; codecs="vp8, vorbis"' />
  <source src="movie.ogv" type='video/ogg; codecs="theora, vorbis"' />
</video>
```

Aquí definimos el elemento y las fuentes. El navegador elegirá solamente una de ellas. Al especificar el atributo `type` (no es obligatorio), permitimos que el navegador conozca el tipo MIME y los tipos de codecs que debe utilizar antes de descargar la canción. Si no indicamos dicho atributo, el navegador intentará averiguar, mediante prueba y error, cuál es el tipo adecuado.

Controlar la reproducción del vídeo y el audio mediante JavaScript

Debido a las diferencias de comportamiento entre un navegador y otro, en los ejemplos anteriores algunos atributos estarán habilitados o deshabilitados por defecto, y algunos de ellos, incluso no funcionarán en algunos navegadores o bajo determinadas circunstancias. Para obtener un control absoluto sobre los elementos `<video>` y `<audio>`, deberemos programar nuestro propio reproductor en JavaScript aprovechando los nuevos métodos, propiedades y eventos incorporados en HTML5.

Los métodos

HTML5 incorpora una lista de métodos para procesamiento de medios.

Los siguientes son los más relevantes:

- 🎵 **play()**: comienza a reproducir el medio desde el inicio, a menos que el medio haya sido pausado previamente.
- 🎵 **pause()**: pausa la reproducción.
- 🎵 **load()**: carga el archivo del medio. Es útil en aplicaciones dinámicas para cargar el medio anticipadamente.
- 🎵 **canPlayType(formato)**: Con este método podemos saber si el formato del archivo se soporta por el navegador o no.

Por ejemplo, para iniciar la reproducción de un vídeo después de pulsar un botón haremos lo siguiente:

```
<video id="video" width="720" height="400">
  <source src="video.mp4">
  <source src="video.ogg">
</video>

<div>
  <input type="button" id="boton" value="Reproducir">
</div>
<script>

function iniciar() {
  var boton=document.getElementById('boton');
  boton.addEventListener('click', presionar, false);
}
```

```
function presionar() {  
    var video=document.getElementById('video');  
    video.play();  
}  
  
window.addEventListener('load', iniciar, false); </script>
```

Las propiedades

Con HTML5 también disponemos de algunas propiedades para recabar información sobre el medio, la mayoría de ellas son de lectura/escritura, es decir, también podemos cambiar su valor y el cambio se reflejará en el vídeo.

Las siguientes son las más relevantes:

- 🎨 **paused:** valdrá true (verdadero) si la reproducción del medio está actualmente pausada o no ha comenzado.
- 🎨 **ended:** valdrá true (verdadero) si la reproducción del medio ha finalizado porque se llegó al final.
- 🎨 **duration:** contendrá la duración del medio en segundos.
- 🎨 **currentTime:** Esta propiedad de lectura/escritura contendrá un valor para informar sobre la posición en segundos en la cual el medio está siendo reproducido, o especifica una nueva posición donde continuar reproduciendo.
- 🎨 **error:** contendrá el valor del error ocurrido.
- 🎨 **muted:** propiedad de lectura/escritura que nos permite saber o indicar si queremos desactivar (true) o activar (false) el sonido del vídeo.
- 🎨 **volume:** propiedad de lectura/escritura que nos permite indicar el volumen del vídeo. El valor del volumen debe ser un número entre 0 y 1,

por ejemplo si indicamos `medio.volume = 0.2` estaríamos configurando el volumen al 20%.

🚦 **buffered**: ofrece información sobre la parte del archivo que ya fue cargada en el buffer. Nos permite crear un indicador para mostrar el progreso de la descarga. Se utiliza, normalmente, cuando se dispara el evento `progress`. Debido a que los usuarios pueden forzar al navegador a cargar el medio desde diferentes posiciones en la línea de tiempo, la información retornada por `buffered` es un array que contiene cada parte del medio que ya ha sido descargada, no sólo la que comienza desde el principio. Los elementos del array son accesibles por medio de los atributos `end()` y `start()`. Por ejemplo, el código `buffered.end(0)` devolverá la duración en segundos de la primera porción del medio encontrada en el buffer.

Podemos cambiar la función `presionar()` del ejemplo anterior, para que se inicie la reproducción del vídeo al presionar el botón y se pause la reproducción si se vuelve a presionar:

```
function presionar() {  
    if(!medio.paused && !medio.ended) {  
        medio.pause();  
        reproducir.value= 'Reproducir';  
    }  
    else {  
        medio.play();  
    }  
}
```

```
        reproducir.value= 'Pausa';  
    }  
}
```

Para clarificar la operación, se cambia el texto del botón en función de la operación que realizamos.

Los eventos

HTML5 incorpora nuevos eventos que son específicos para el procesamiento de vídeo y audio. Los eventos fueron incorporados con el objetivo de informar sobre la situación del medio (el progreso de la descarga, si la reproducción del medio finalizó, o si la reproducción del medio es iniciada o pausada, entre otras). Estos son los más relevantes:

- 🚦 **progress**: se dispara periódicamente para informar acerca del progreso de la descarga del medio. La información estará disponible a través del atributo `buffered`.
- 🚦 **canplaythrough**: se dispara cuando el medio completo puede ser reproducido sin interrupción. El estado se establece considerando la actual tasa de descarga y asumiendo que seguirá siendo la misma durante el resto del proceso. Existe otro evento más para este propósito, `canplay`, pero no considera toda la situación y se dispara tan pronto como algunas partes del medio se encuentran disponibles (después de descargar los primeros cuadros de un vídeo, por ejemplo).
- 🚦 **ended**: se dispara cuando el reproductor llega al final del medio.
- 🚦 **pause**: se dispara cuando el reproductor es pausado.
- 🚦 **play**: se dispara cuando el medio comienza a ser reproducido.
- 🚦 **error**: se dispara cuando ocurre un error.

4. CSS3

En este tema vamos a estudiar, las nuevas posibilidades que nos ofrece CSS3 para crear transformaciones y transiciones en los elementos HTML, pero antes veremos que opciones tenemos para posicionar un elemento en nuestra página.

Posicionamiento de los elementos

Las propiedades de posicionamiento CSS nos permiten colocar un elemento en el lugar de la página que más nos interese. Los elementos pueden ser colocados utilizando las propiedades top, bottom, left y right. Sin embargo, estas propiedades no funcionarán a menos que la propiedad position se establezca en primer lugar. En función del método de posicionamiento que utilicemos, estas propiedades funcionarán de una manera u otra. Disponemos de cuatro métodos de posicionamiento diferentes variando el valor de la propiedad position:

- 🚦 **static** (posicionamiento estático): Este es el valor por defecto de la propiedad. Los elementos estáticos se encuentran posicionados de acuerdo al flujo normal de la página, es decir, a continuación del anterior elemento con posicionamiento estático. Los elementos con este posicionamiento no se ven afectados por las propiedades top, bottom, left y right.
- 🚦 **fixed**: Un elemento con la posición fija se posiciona relativo a la ventana del navegador, por lo tanto, no se moverá aunque se haga scroll en la página. Además, estos elementos se eliminan del flujo normal del documento y el resto de elementos se comportarán como si estos

elementos no existieran. Esto hace que los elementos posicionados fijos puedan superponerse a otros elementos o entre sí.

footer { position:fixed; bottom:0px; left:0px; }

En el ejemplo, se especifica un posicionamiento fijo del <footer>, y estará colocado siempre en la parte inferior de la pantalla y en el extremo izquierdo. Así, aunque hagamos más o menos alta la ventana del navegador o exista más o menos contenido, el <footer> siempre estará fijo en la parte inferior de la misma.

🎨 **relative:** La posición de un elemento con posicionamiento relativo se encuentra relativa a la posición que debería ocupar dentro del flujo normal del documento. Así, su contenido puede ser movido y/o superpuesto a otros elementos, pero el espacio reservado para el elemento se mantendrá reservado dentro del flujo normal del documento. A menudo, se utilizan elementos con posicionamiento relativo para contener otros elementos con posicionamiento absoluto.

🎨 **absolute:** Un elemento con posicionamiento absoluto se posiciona de forma relativa al primer elemento padre que tiene un posicionamiento distinto de estático, normalmente relativo y, en ocasiones, absoluto o fijo. Si no se encuentra dentro de ningún elemento con estos posicionamientos, se colocará relativo al elemento <html>. En este caso, será muy probable que cuando movamos horizontalmente la ventana del navegador o -todavía más grave- que según tenga el usuario una anchura u otra del navegador, el elemento este posicionado en un lugar u otro horizontalmente. Para evitar este grave problema, o bien introducimos el elemento con posición absoluta dentro de otro elemento con posición

relativa, o bien le damos una posición relativa al body, que es el elemento que contiene al resto de elementos. Evidentemente, los elementos con posicionamiento absoluto se eliminarán del flujo normal del documento, por lo que el resto de elementos se posicionarán como si estos elementos no existieran. En ocasiones, esto hará que los elementos con posicionamiento absoluto se superpongan sobre otros elementos.

Elementos superpuestos

Cuando los elementos se posicionan fuera del flujo normal del documento, puede ser que se superpongan. En estas circunstancias, debemos indicar de alguna forma qué elementos se colocan delante y cuáles se colocan detrás. Para esta tarea disponemos de la propiedad z-index.

La propiedad z-index indica el orden de apilamiento de un elemento (qué elemento debe situarse delante o detrás del resto). Un elemento con un orden de apilamiento mayor, está siempre delante de un elemento con un orden menor. Si dos elementos se solapan sin tener especificada la propiedad z-index, el que haya sido insertado más tarde en el código html estará sobre el otro.

Veamos el siguiente ejemplo:

```
<header>
  <div id="chincheta1"></div>
  <div id="chincheta2"></div>
  <h1>Título de la página</h1>
</header>
<style>
```



```
header { position: relative; padding: 20px 100px; }
```

```
#chincheta1 {  
    position: absolute;  
    left: 320px;  
    top: -5px;  
    width: 35px;  
    height: 49px;  
    background-image:  
    url("imgs/chincheta.png");  
    z-index: 2;  
    border: 1px solid black;  
}
```

```
#chincheta2 {  
    position: absolute;  
    left: 350px;  
    top: -5px;  
    width: 35px;  
    height: 49px;  
    background-image: url("imgs/chincheta.png");  
    z-index: 1;  
}  
</style>
```



Título de la página

En el ejemplo tenemos un elemento `<header>` que contiene dos `<div>` vacíos y un `<h1>`. Si nos fijamos en los estilos, el `<header>` tiene posicionamiento relativo, pero como no hemos indicado las propiedades `top`, `bottom`, `left` ni `right`, estas tomarán los valores por defecto, es decir `0px`, por lo tanto, el elemento se colocará como si tuviera posicionamiento estático.

La ventaja de tener el `<header>` con posicionamiento relativo es que, al colocar los dos `<div>` con posicionamiento absoluto, su posición será relativa a la posición del elemento `<header>`, ya que este será el primer elemento padre que los contiene con un posicionamiento distinto de estático.

Además, como se aprecia en la imagen, los dos elementos con posicionamiento absoluto están superpuestos, el primero (que tiene borde) tiene la propiedad `z-index` a 2, mientras que el segundo (que no tiene borde) tiene la propiedad a 1, por lo tanto, el primero estará colocado sobre el segundo.

Transformaciones (transform)

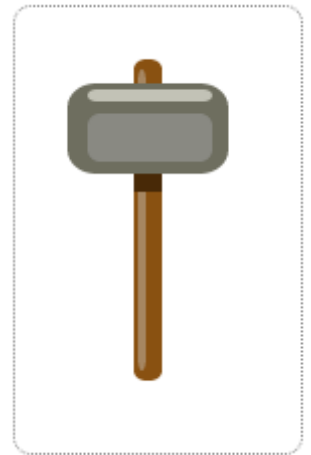
Con HTML5 y CSS3 podemos utilizar la propiedad `transform` para modificar características de un elemento como la posición (`translate`), la rotación (`rotation`), la escala (`scale`) o el sesgo (`skew`).

Este cambio de posición, tamaño o rotación se produce al instante (no hay animación en este cambio, para ello habría que utilizar otras propiedades que veremos más adelante). Para poder comprobar estas propiedades vamos a utilizar un `<div>` que contiene el dibujo sencillo de un martillo creado con SVG.

```

<div id="martillo1">
  <svg width="250" height="250" xmlns="http://www.w3.org/2000/svg">
    <!-- Mango (marrón oscuro) -->
    <rect x="43" y="10" width="14" height="160" rx="6" ry="5" fill="#895111"/>
    <!-- Brillo mango (marrón claro) -->
    <rect x="45" y="15" width="4" height="150" rx="6" ry="6" fill="#A68661"/>
    <!-- Cabeza hierro -->
    <rect x="10" y="22" width="80" height="45" rx="14" ry="12" fill="#6E6E5F"/>
    <!-- Brillo cabeza -->
    <rect x="20" y="37" width="62" height="24" rx="6" ry="6" fill="#898982"/>
    <rect x="20" y="25" width="62" height="6" rx="4" ry="4" fill="#C2C2B6"/>
    <!-- Sombra de la cabeza en el mango -->
    <rect x="43" y="67" width="14" height="9" fill="#482907"/>
  </svg>
</div>
#martillo
{
  position:relative;
  width:110px;
  height:190px;
  padding:16px;
  border:1px dotted gray;
  border-radius:9px;
  float:left;
}

```



A este <div> le vamos a aplicar todas las transformaciones de posición, rotación, escala y sesgo, partiendo de esta posición, tamaño, escala y apariencia original.

Movimiento (translate)

El movimiento se basa en la suma o resta de píxeles en su posición horizontal y vertical respecto a la posición en la que se encuentra el objeto en este momento. Con números positivos se desplaza el objeto hacia la derecha y hacia abajo y con números negativos se desplaza hacia arriba y hacia la izquierda. Así, en este ejemplo, el martillo se desplaza 200 píxeles hacia la

derecha y 5 píxeles hacia abajo respecto a su posición original.

```
#martillo { transform:translate(200px,5px); }
```

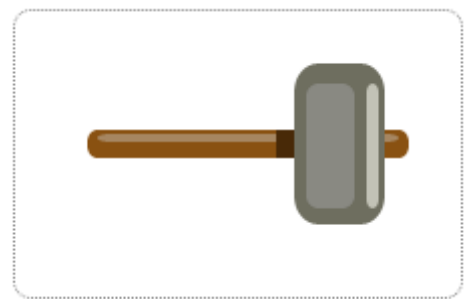
También podemos utilizar funciones independientes para cada dimensión: `translateX` y `translateY`.

También es posible utilizar porcentajes, si queremos colocar este martillo en la mitad horizontal de la pantalla, es decir, centrar el martillo horizontalmente, podemos indicar un porcentaje (50%), en lugar de utilizar píxeles.

```
#martillo { transform:translate(50%,0px); }
```

Rotación (rotate)

La rotación se basa en los grados de rotación (de 0º a 360º). El eje de rotación siempre está situado en el centro del elemento y los grados se especifican añadiendo **deg** después del valor (por ejemplo: 90deg). Si se indican valores positivos en los grados de rotación, el objeto rota hacia la derecha. Con los grados especificados en números negativos, el objeto rota hacia la izquierda. (Así, teniendo en cuenta que una vuelta entera equivale a 360º, se aprecia el mismo resultado con 350deg, que con -10deg).



```
#martillo{ transform:rotate(90deg); }
```

En este caso, el elemento del martillo rota 90º hacia la derecha tomando como eje de rotación el centro del <div> donde se encuentra el martillo (que no siempre coincide con el contenido de dicho <div>).

Escala (scale)

Esta propiedad se basa en una escala horizontal, seguida de otra vertical, siendo 1 la escala original. Para obtener un porcentaje se puede multiplicar el valor indicado por 100, así un valor de '2' ($2 \times 100 = 200$) representaría un 200%, es decir, el doble de tamaño. El elemento aumenta o disminuye de tamaño tomando como eje su centro.

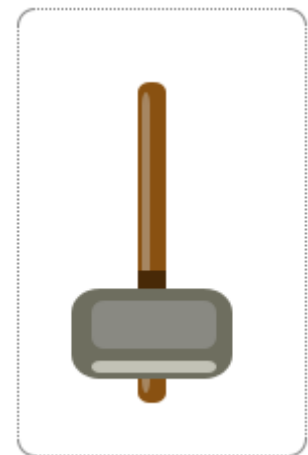
```
#martillo {  
transform:scale(2, 1.75);  
}
```



En este caso, el elemento del martillo no aumenta de tamaño de forma proporcional. Aumenta el doble de anchura ($2 \times 100 = 200\%$) y sólo un 175% de altura ($1.75 \times 100 = 175\%$). Podemos conseguir el efecto del espejo utilizando números negativos para la escala.

```
#martillo { transform:scale(1, -1); }
```

En el ejemplo, se mantiene la escala en el eje x, ya que estamos utilizando el valor 1 que mantiene la proporción original para la dimensión horizontal. Sin embargo, el valor -1 en el eje y, aunque también mantiene la proporción original en esta dimensión, invierte el elemento verticalmente para producir el efecto espejo, como se puede apreciar en la imagen. Existen también otras dos funciones similares a



scale, pero restringidas a la dimensión horizontal o vertical: **scaleX** y **scaleY**. Estas funciones, por supuesto, utilizan un único parámetro.

Sesgo (skew)

Podemos entender el sesgo como la rotación del elemento en los ejes x e y. Para definirlo se indica primero la rotación en el eje x en grados, y posteriormente, la rotación en el eje y, también en grados. Se pueden utilizar valores positivos o negativos, según se requiera para la deformación.

```
#martillo { transform:skew(10deg, 25deg); }
```

Si sólo indicamos el primer parámetro, únicamente se rotará el elemento sobre el eje x. Al igual que ocurría con la función rotate podemos utilizar funciones diferentes para cada dimensión: **skewX** y **skewY**.



Múltiples transformaciones simultáneas

Se pueden indicar, de manera simultánea, tantas transformaciones como se deseen, separando cada una de ellas por un espacio.





```
#martillo { transform:translate(200px, 0px) rotate(33deg) scale(0.5,0.5); }
```

En este ejemplo, al <div> del martillo se le aplica un desplazamiento hacia la derecha de 200 píxeles, seguidamente una rotación hacia la derecha de 33 grados y, finalmente, una reducción de tamaño proporcional del 50%.



Compatibilidad con todos los navegadores

Es importante tener en cuenta que para que transform funcione en todos los navegadores deberemos repetir la propiedad anteponiendo los prefijos de cada uno de los navegadores:

-  -webkit- (para Chrome y Safari)
-  -moz- (para Firefox)
-  -o- (para Opera)
-  -ms- (para Internet Explorer)

#martillo{

```
transform:translate(200px, 0px) rotate(33deg) scale(0.5,0.5);  
-webkit-transform:translate(200px, 0px) rotate(33deg) scale(0.5,0.5);  
-moz-transform:translate(200px, 0px) rotate(33deg) scale(0.5,0.5);  
-o-transform:translate(200px, 0px) rotate(33deg) scale(0.5,0.5);  
-ms-transform:translate(200px, 0px) rotate(33deg) scale(0.5,0.5);  
}
```

Veamos la tabla de compatibilidad que nos ofrecía Can I use? para esta propiedad en el momento de crear estos apuntes. (URL: <http://caniuse.com/#feat=transforms2d>):

► Show options = Supported = Not supported = Partially supported = Support unknown

Show all tables

CSS3 Transforms - **Working Draft** *Usage stats: Global Support: 83.72%

Method of transforming an element including rotating, scaling, etc.

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	BlackBerry Browser	IE Mobile
								2.1 ~webkit-		
								2.2 ~webkit-		
						3.2 ~webkit-		2.3 ~webkit-		
						4.0-4.1 ~webkit-		3.0 ~webkit-		
	8.0 ~ms-		31.0 ~webkit-			4.2-4.3 ~webkit-		4.0 ~webkit-		
	9.0		32.0 ~webkit-			5.0-5.1 ~webkit-		4.1 ~webkit-		
	10.0	27.0	33.0 ~webkit-			6.0-6.1 ~webkit-		4.2-4.3 ~webkit-	7.0 ~webkit-	
Current	11.0	28.0	34.0 ~webkit-	7.0 ~webkit-	20.0 ~webkit-	7.0 ~webkit-	5.0-7.0	4.4 ~webkit-	10.0 ~webkit-	10.0
Near future		29.0	35.0 ~webkit-		21.0 ~webkit-					
Farther future		30.0	36.0 ~webkit-		22.0 ~webkit-					
3 versions ahead		31.0	37.0 ~webkit-							

Notes Known issues (3) Resources (8) Feedback Edit on GitHub

The scale transform can be emulated in IE < 9 using Microsoft's "zoom" extension, others are (not easily) possible using the MS Matrix filter

Transformaciones dinámicas

Las transformaciones vistas hasta ahora cambiarán la forma de la web, pero la mantendrán tan estática como siempre. Sin embargo, podemos aprovecharnos de la combinación de transformaciones y pseudo-clases para convertir nuestra página en una aplicación dinámica:

```
#martillo {
  position: relative;
  width: 110px;
  height: 190px;
  padding: 16px;
  border: 1px dotted gray;
  border-radius: 9px;
  float: left;
}
#martillo: hover { transform: scale(1, -1); }
```


En este ejemplo, la regla original del elemento martillo se ha conservado intacta, pero hemos añadido una nueva regla para aplicar efectos de transformación usando la pseudo-clase :hover. El resultado obtenido es que cada vez que el puntero del ratón pase sobre este elemento, la propiedad transform lo invertirá sobre el eje vertical, y cuando el puntero salga del elemento, volverá a rotar para volver a su posición original.

Esta técnica ya se utilizaba con versiones anteriores de CSS para crear **rollovers**, por ejemplo, pero con la nueva propiedad transform se pueden conseguir efectos más logrados.

Transiciones (transition)

Las transiciones (transition) funcionan como pequeñas animaciones que siguen las mismas características que las transformaciones (cambio de posición, tamaño, rotación y/o muchas otras propiedades CSS), aunque con la capacidad de poder especificar un tiempo para que se produzca dicha transformación, y así poder utilizarla como una animación sencilla.

Aunque no es obligatorio, tenemos la posibilidad de especificar el tipo de movimiento y si debe existir algún retardo para que empiece la animación.

Además, es interesante poder especificar todos estos parámetros en una sola línea.

transition:[propiedad a modificar] [Duración] [Tipo de animación] [Retardo];

transition: width 2s ease-out 0.5s;

🎨 [Propiedades a modificar]: Las propiedades que podemos modificar utilizando transiciones son las que se muestran en la siguiente lista:

all	bottom	margin	text-shadow
background-color	left	opacity	vertical-align
border	right	word-spacing	visibility
border-radius	box-shadow	letter-spacing	z-index
color	width	fill	
top	height	padding	
	line-height	stroke	

🎨 [Duración en segundos]: Se debe especificar el número de segundos que va a durar la animación. Primero se escribe un valor numérico (los segundos) y seguido la "s" (ejemplo: 3s).

🎨 [Tipo de animación]: Esta propiedad es opcional. Aquí tenemos la posibilidad de indicar un tipo de easing, es decir, si la animación empezará suavemente e irá acelerándose progresivamente, si pasará todo lo contrario o si el movimiento tendrá una velocidad uniforme durante todo su recorrido entre otras. Algunas de las posibilidades son las siguientes:

- **linear**: La velocidad de la animación es uniforme en todo el recorrido.
- **ease**: La velocidad se acelera al inicio, luego se retarda un poco y se acelera al final de nuevo.
- **ease-in**: La animación empieza lentamente y va aumentando progresivamente.

- **ease-out**: La animación empieza muy rápida y va descendiendo progresivamente.
- **ease-in-out**: La animación empieza y acaba lentamente, y es en la parte central del recorrido donde la velocidad es más rápida.

🚦 [Retardo]: En este apartado podemos especificar un tiempo (en



segundos) que el navegador esperará antes de poner en marcha la animación. Se especifica el número de segundos a esperar seguido de la "s" (ejemplo: 1s).

Compatibilidad con todos los navegadores

Al igual que ocurría con las transformaciones, para que transition funcione en todos los navegadores deberemos repetir la propiedad anteponiendo los prefijos de cada uno de los navegadores:

- 🚦 -webkit- (para Chrome y Safari)
- 🚦 -moz- (para Firefox)
- 🚦 -o- (para Opera)
- 🚦 -ms- (para Internet Explorer)

Transición con una única propiedad

Vamos a hacer que el elemento martillo con el que hemos trabajado anteriormente, cambie su color de fondo (background-color) utilizando para ello una transición. La transición tendrá una duración de 1 segundo, el tipo de animación será ease-in-out y tendrá un retardo de medio segundo.

En primer lugar, indicaremos en la regla css del martillo como se realizará la transición de la propiedad background-color si esta fuera cambiada.

```
#martillo {  
    position:relative;  
    width:110px;  
    height:190px;  
    padding:16px;  
    border:1px dotted gray;  
    border-radius:9px;  
    float:left;  
    transition: background-color 1s ease-in-out 0.5s;  
}
```

A continuación, utilizaremos la pseudo-clase :hover para cambiar el color de fondo del elemento:

```
#martillo:hover { background-color: black; }
```

El efecto obtenido será que al pasar con el ratón sobre el martillo se producirá la transición.

Transición con múltiples propiedades

Para especificar el cambio de varias propiedades con diferentes tiempos para cada una de ellas, únicamente tenemos que separar las diferentes propiedades por "," y, posteriormente, modificarlas en el lugar del código necesario.

Así, en el siguiente ejemplo, al pasar el ratón por encima, se producirán 4 cambios de propiedades al mismo tiempo:

- 🔧 El color de fondo (background-color) del martillo se convertirá en negro en 1 segundo.
- 🔧 La anchura se modificará hasta llegar a los 225 píxeles en 1 segundo.
- 🔧 La altura se modificará hasta llegar a los 200 píxeles durante 1 segundo.
- 🔧 El ancho del borde del martillo (border-width) se expandirá hasta llegar a los 10 píxeles en 1 segundo.

```
#martillo {  
    transition: background-color 1s, width 1s, height 1s, border-width 1s;  
}
```

```
#martillo:hover {  
    background-color: black;  
    width:225px;  
    height:200px;  
    border-width:10px;  
}
```

Transiciones con propiedades CSS3

Debemos tener en cuenta que cuando realizamos transiciones utilizando propiedades css3 que requieren el uso de prefijos, estos debemos indicarlos, tanto en la propiedad transition como en la propiedad a modificar. Por ejemplo, para realizar la transformación del espejo creada anteriormente utilizando una transición, haremos lo siguiente:

```
#martillo {  
    position: relative;  
    width: 110px;  
    height: 190px;  
    padding: 16px;  
    border: 1px dotted gray;  
    border-radius: 9px;  
    float: left; transition:  
    transform 1s ease-in-out 0.5s;  
    -moz-transition: -moz-transform 1s ease-in-out 0.5s;  
    -webkit-transition: -webkit-transform 1s ease-in-out 0.5s;  
    -o-transition: -o-transform 1s ease-in-out 0.5s;  
    -ms-transition: -ms-transform 1s ease-in-out 0.5s;  
}  
  
#martillo: hover {  
    transform: scale(1, -1);  
    -moz-transform: scale(1, -1);
```

```
-webkit-transform:scale(1,-1);  
-o-transform:scale(1,-1);  
-ms-transform:scale(1,-1);  
}
```

Recordad que si queréis reproducir los ejemplos que se han indicado, debéis añadir las líneas correspondientes para los navegadores que utilizéis.

Un ejemplo práctico. Menús desplegables con transiciones.

(este ejemplo lo necesitaréis para vuestra página web de ejercicio)

Hasta la aparición de CSS3 los menús desplegables se basaban, principalmente, en JavaScript o flash, pero desde la aparición de las transiciones y otros efectos de esta nueva versión de CSS, podemos crear múltiples efectos sin necesidad de programar nada ni utilizar ningún plug-in externo.

Vamos a ver un ejemplo de menú desplegable utilizando transiciones y algún truco CSS para conseguir un aspecto más atractivo.

En primer lugar, el código HTML que necesitaremos será el siguiente:

```
<nav>  
<ul>  
  <li><a href="#">Opción 1</a>  
    <ul>  
      <li><a href="#">Subopción 1</a></li>  
      <li><a href="#">Subopción 2</a></li>  
      <li><a href="#">Subopción 3</a></li>  
    </ul>  
  </li>  
  <li><a href="#">Opción 2</a>  
    <ul>  
      <li><a href="#">Subopción 4</a></li>
```

- [Opción 1](#)
 - [Subopción 1](#)
 - [Subopción 2](#)
 - [Subopción 3](#)
- [Opción 2](#)
 - [Subopción 4](#)
 - [Subopción 5](#)
 - [Subopción 6](#)
 - [Subopción 7](#)

```
<li><a href="#">Subopción 5</a></li>
<li><a href="#">Subopción 6</a></li>
<li><a href="#">Subopción 7</a></li>
</ul>
</li>
</ul>
</nav>
```

Con el código anterior tendremos dos opciones de menú, la primera desplegará un menú con tres subopciones y la segunda desplegará un menú con cuatro subopciones. Antes de aplicarle los estilos, la página se verá como se muestra en la imagen.

A continuación, veremos los estilos que necesitamos.

En primer lugar, inicializaremos los márgenes internos y externos de todos los elementos a 0px mediante el selector universal:

```
* { margin: 0; padding: 0; }
```

Para los enlaces del menú principal bastará con quitar el subrayado y cambiar el color de la fuente cuando el ratón está sobre ellos:

```
nav>ul>li>a { text-decoration:none; }
```

```
nav>ul>li>a:hover { color: red; }
```

Para convertir la lista desordenada en un menú utilizaremos los siguientes estilos:

```
nav>ul>li {
    position:relative;
    list-style:none;
    display:inline-block;
    padding: 20px 0px;
```



```
margin: 0px 80px;  
}
```

Cabe destacar la propiedad `position: relative;` que utilizamos para poder poner el menú desplegable con posición absoluta. De esta forma, la posición del menú desplegable será relativa a la opción que lo contiene.

Los estilos que utilizaremos para el menú desplegable serán los siguientes:

```
nav li ul {
```

```
/* sacamos el submenú del flujo normal de los contenidos poniéndole posición absoluta */
```

```
position: absolute;
```

```
/* Le ponemos un z-index de 2 para que en el caso que el submenú se solape con otros elementos quede por encima de ellos */
```

```
z-index: 2;
```

```
list-style: none;
```

```
/* El ancho que le pondremos dependerá de lo que contengan los submenús. Lo que pongamos en la propiedad margin-left será la mitad de lo que hayamos puesto aquí */
```

```
width: 100px;
```

```
text-align: center;
```

`/* Para centrar el submenú con respecto a la opción de menú, pondremos su parte izquierda en la mitad del ancho y le restaremos a los márgenes la mitad del ancho que hayamos puesto anteriormente */`

`left:50%;`

`margin-left:-50px;`

`/* el valor de la propiedad top dependerá de lo que contengan las opciones de menú principales, pondremos un valor adecuado para que el submenú se muestre debajo. */`

`top:25px;`

`font:Verdana;`

`font-size: 0.7em;`

`/* ponemos las propiedades height: 0px y overflow: hidden para que el submenú quede oculto. La propiedad height será la que utilizaremos para animar la transición */`

`height:0px;`

`overflow:hidden;`

`/* aquí configuramos la transición para cuando se cambie el valor de la propiedad height. Esto hará que la aparición y desaparición del submenú esté animada */`

`-webkit-transition:height 0.4s linear;`

`-moz-transition:height 0.4s linear;`

```
-o-transition:height 0.4s linear;  
transition:height 0.4s linear;  
}
```

El siguiente paso será hacer que cambie la altura del submenú cuando pasamos el ratón sobre los elementos del menú:

```
nav li:hover ul { height:200px; }
```

Como se puede observar, utilizamos la pseudo-clase :hover en el elemento li, pero las propiedades que cambiamos son las del elemento ul que está dentro.

Con esto hemos conseguido que el submenú desaparezca y vuelva a aparecer cuando pasamos el ratón sobre la opción correspondiente, pero el efecto de despliegue no se aprecia demasiado, porque no hemos aplicado estilos a las opciones del submenú. A continuación, aplicaremos una serie de estilos que mejorarán el aspecto de nuestro menú desplegable.

```
/* Con esto pondremos un color de fondo para los elementos del menú que sean pares y otro para los impares. También cambiaremos el color de fondo cuando pasemos el ratón sobre ellos. */
```

```
nav li ul li { background-color: #313131; }  
nav li ul li:nth-child(odd) { background-color: #363636; }  
nav li ul li:hover { background-color: #444; }
```

```
/* con esto redondearemos las esquinas del submenú, además de colocarlo debajo de la opción de menú correspondiente */
```

```
nav li ul li:first-child {  
border-radius:3px 3px 0 0;
```

`/* Lo que pongamos como margen superior, deberá coincidir con lo que hemos puesto en el top del ul */`

`margin-top:25px;`

`/* indicamos posición relativa para poder introducir luego el triángulo del menú desplegable con posición absoluta */`

`position:relative;`

`}`

`nav li ul li:last-child {`

`border-bottom-left-radius:3px;`

`border-bottom-right-radius:3px;`

`}`

`/* Con esto conseguiremos que aparezca el triángulo superior que aparece en el menú desplegable */`

`nav li ul li:first-child:before{`

`content:'';`

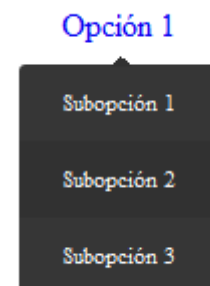
`position:absolute;`

`width:1px;`

`height:1px;`

`border:5px solid transparent;`

`border-bottom-color: #313131;`



```
    left:50%;  
    top:-10px;  
    margin-left:-5px;  
}
```

/* Aquí configuraremos la visualización de los enlaces del menú desplegable */

```
nav li ul li a{  
    padding:12px;  
    color:#fff;  
    text-decoration:none;  
    display:block;  
}
```

Como se puede apreciar en la imagen, la apariencia del menú está bastante lograda y, si lo probamos, veremos que el efecto que se obtiene está bastante conseguido.

5. Ejercicios

Para practicar todo lo expuesto en este tema, vamos a realizar una serie de mejoras en la página web del tema anterior. Comenzaremos por la parte de CSS.

Ejercicio 1

Ya dijimos en temas anteriores que a la hora de decidir si una imagen la introducíamos mediante HTML o mediante CSS, teníamos que pensar si la imagen formaba parte de los contenidos de la web (HTML) o si, por el contrario, se trataba de una imagen de decoración (CSS). En la cabecera de nuestra página tenemos una imagen con el logo de HTML5, CSS3 y JavaScript. Esta imagen la introducimos en el primer tema por medio de CSS, pero al representar el logo del que tratan los contenidos de nuestra página, quizás fuera mejor introducirla dentro del HTML.

Lo que tenemos que hacer en este ejercicio es eliminar la imagen del logo de la cabecera de nuestra hoja de estilos e introducirla mediante HTML. Para ello introduciremos un nuevo elemento `` dentro del elemento `<h1>` de nuestro `<header>`.

El aspecto de la página debe continuar igual que cuando teníamos la imagen dentro del CSS, por lo que tendremos que aplicar los estilos que sean necesarios. En concreto, pondremos el elemento `<h1>` que contendrá el `` con posicionamiento relativo y el elemento `` con posicionamiento absoluto y posición izquierda y superior a 15px.

Ejercicio 2

En ocasiones, para conseguir un diseño específico no tenemos más remedio que introducir un elemento que no tendrá semántica ni aportará nada a los contenidos, pero que nos ayudará como elemento contenedor para, por ejemplo, poner una imagen de fondo y ubicarla en el lugar que nos interese. En estos casos, utilizaremos siempre elementos `<div>` o ``, dependiendo de si necesitamos un elemento en bloque o en línea. El resultado que queremos obtener en este ejercicio es el siguiente:



La imagen de la chincheta se encuentra en el fichero recursos3.zip. Para colocar esta imagen sobre la cabecera de nuestros móviles, necesitaremos introducir un elemento `<div>` en cada uno de los `<header>` de nuestros elementos `<article>`.

```
<div class="chincheta"></div>
```

Una vez modificado el html tendremos que colocar la chincheta en el lugar adecuado, para lo cual, le pondremos posición absoluta (evidentemente, los elementos <header> de los <article> tendrán posición relativa). Los elementos chincheta tendrán las siguientes propiedades: ancho 35px, alto 49px, posición superior -91px, imagen de fondo chincheta.png. Además, habrá que controlar la superposición con los móviles. La chincheta del primer <article> tendrá la posición izquierda en 199px, la del segundo en 120px y la del tercero en 30px.

Para evitar que las chinchetas se solapen con la barra de navegación, le pondremos un margen superior a los <article> de 30px. Una vez hemos colocado las chinchetas, para que parezca que los móviles están colgando de ellas, rotaremos un poco los <article> primero y tercero. Para ello utilizaremos una transformación de rotación de -2deg en el primero y 2deg en el tercero.

Ejercicio 3

En este ejercicio vamos a aplicar un efecto sobre los enlaces de la barra de navegación, de forma que al colocar el cursor sobre ellos se produzca una transición que cambie su aspecto.

En primer lugar, le aplicaremos al texto de estos enlaces una sombra con las siguientes características: distancia horizontal y vertical 0px, radio de difuminación 5px y color blanco. El enlace quedará así:

Contacto

A continuación, indicaremos que al poner el ratón sobre el enlace se aplique una sombra con las siguientes características: distancia horizontal y vertical 2px, radio de difuminación 0px y color rojo. El enlace con el ratón sobre él quedará así:

Contacto

Finalmente, indicaremos que la transición de una sombra a la otra tarde 0.5 segundos.

Ejercicio 4

En este ejercicio, vamos a reproducir el ejemplo práctico que se ha presentado en el último punto del tema (El menú desplegable).

Crearemos un menú desplegable que se mostrará cuando coloquemos el ratón en la opción de menú vídeos. El menú contendrá tres opciones: vídeos1, vídeos2 y vídeos3. Estas tres opciones, así como el enlace principal a vídeos, enlazarán con el documento videos.html que crearemos en el siguiente ejercicio.

Los colores del menú que debes utilizar son los siguientes:

```
#page>nav li ul li { background-color:#e34c26; }
```

```
#page>nav li ul li:nth-child(odd) { background-color:#f06529; }
```

```
#page>nav li ul li:hover { background-color:#ff7e42; }
```

El color de borde de la regla `#page>nav li ul li:first-child:before`, será `#e34c26`.

La página con el menú desplegado se verá de la siguiente forma:



Ten en cuenta, que parte de la opción principal vídeos estará solapada con la parte superior del menú, por lo tanto, será normal que sólo se resalte la opción vídeos cuando estamos en la parte de arriba del enlace. En el fichero de recursos tenéis la imagen resultado.jpg en la que se puede ver esta imagen con mejor resolución.

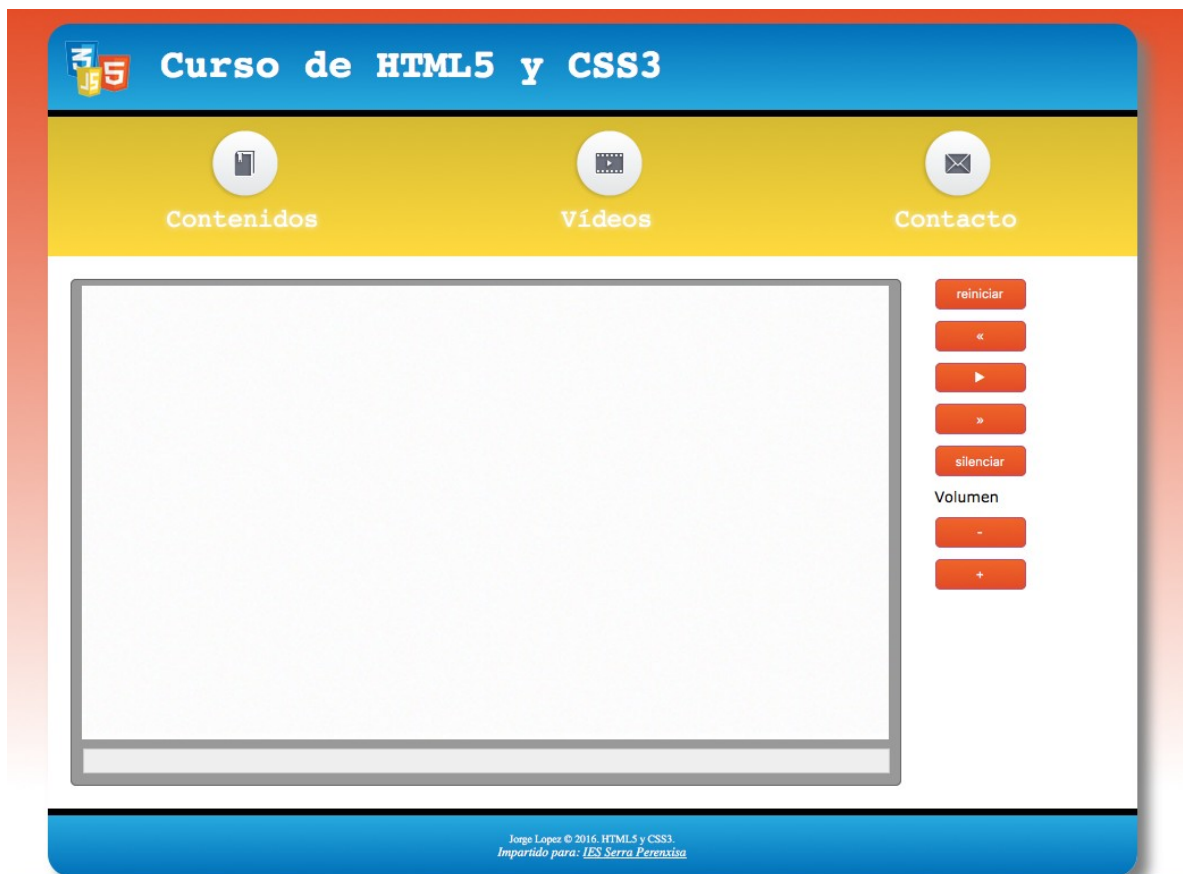
Ejercicio 5

En este último ejercicio vamos a crear una nueva sección dentro de nuestra página web, en concreto, la sección de vídeos. Accederemos a esta sección a través de los enlaces del menú desplegable creado en el ejercicio

anterior y de la opción principal vídeos. Esta sección la crearemos dentro de un documento llamado videos.html. Este documento será exactamente igual que el documento index.html, salvo dos excepciones:

- ✚ En el menú de navegación, el enlace de contenidos apuntará a la página index.html.
- ✚ Los contenidos del documento (elemento <main>) serán diferentes a los del documento index.html y se explicarán a continuación.

El aspecto de la página de vídeos será el siguiente:



En el fichero de recursos puedes encontrar la imagen videos.jpg que muestra este resultado con mejor resolución.

El código de nuestra sección <main>, será el siguiente:

```
<main>
  <div id="reproductor">
    <video id="medio" width="720" height="400">
      <source src="media/trailer.mp4">
      <source src="media/trailer.ogg">
    </video>
    <div id="barra">
      <div id="progreso"></div>
    </div>
  </div>
  <nav>
    <input type="button" id="reiniciar" value="reiniciar">
    <input type="button" id="retrasar" value="&laquo;">
    <input type="button" id="play" value="⏵">
    <input type="button" id="adelantar" value="&raquo;">
    <input type="button" id="silenciar" value="silenciar">
    <label>Volumen</label>
    <input type="button" id="menosVolumen" value="-">
    <input type="button" id="masVolumen" value="+">
  </nav>
</main>
```

Como se puede observar en la imagen, tenemos dos partes diferenciadas, por un lado el reproductor de vídeo con una barra inferior que nos servirá para ir marcando el porcentaje de reproducción y, por otro lado, la barra de navegación que contiene los botones que nos servirán para controlar la reproducción del vídeo. Los estilos que utilizaremos para conseguir el aspecto de la imagen los tenemos en el fichero reproductor.css que podéis encontrar también en el fichero de recursos. Debéis guardar este fichero en la carpeta css de vuestra web y enlazarlo con el fichero videos.html.

En este fichero encontraremos los siguientes estilos:

```
#reproductor {
    float:left;
    width: 720px;
    margin: 20px;
    padding: 5px;
    background: #999999;
    border: 1px solid #666666;
    -moz-border-radius: 5px;
    -webkit-border-radius: 5px;
    -o-border-radius: 5px;
    border-radius: 5px;
}

main nav {
    float: left;
    margin: 10px auto;
    width: 100px;
}

main nav label {
    margin: 10px;
}

input[type="button"] {
    width: 80px;
    margin: 10px;
    display: block;

    background: -webkit-linear-gradient(top,#f06529,#e34c26);
    background: -moz-linear-gradient(top,#f06529,#e34c26);
    background: -o-linear-gradient(top,#f06529,#e34c26);
    background: linear-gradient(top bottom,#f06529,#e34c26);
    border: 1px solid rgb(180, 80, 121);

    -moz-border-radius: 5px;
    -webkit-border-radius: 5px;
    -o-border-radius: 5px;
    border-radius: 5px;
}
```

```
padding: .4em;
color: white;
text-decoration:none;
}

input[type="button"]:hover {
background: -webkit-linear-gradient(top,#ff6529,#f34c26);
background: -moz-linear-gradient(top,#ff6529,#f34c26);
background: -o-linear-gradient(top,#ff6529,#f34c26);
background: linear-gradient(top bottom,#ff6529,#f34c26);
}

#barra {
width: 705px;
margin:5px;
height: 16px;
padding: 2px;
border: 1px solid #CCCCCC;
background: #EEEEEE;
}

#progreso {
position: absolute;
width: 0px;
height: 16px;
background: rgba(0,0,150,.2);
}
```

Dentro del fichero de recursos tenemos una carpeta media que contiene los vídeos trailer.mp4 y trailer.ogg. Esta carpeta debemos moverla a nuestra web.

Además, dentro de este mismo fichero, tenemos también un fichero JavaScript llamado reproductor.js. Debemos guardarlo dentro de una carpeta scripts en nuestra web. Después, enlazaremos este fichero JavaScript con nuestra página vídeos.html.

El contenido del fichero es el siguiente:

```
function redimensionaBarra() {
    if(!medio.ended)
    {
        var total=parseInt(medio.currentTime*maximo / medio.duration);
        progreso.style.width=total+'px';
    } else {
    } }
    progreso.style.width='0px';
    play.value='\u25BA';
    window.clearInterval(bucle);
}



function desplazarMedio(e) {
    if(!medio.paused && !medio.ended) {
        var ratonX=e.pageX-barra.offsetLeft;var
        nuevoTiempo=ratonX*medio.duration/maximo;
        medio.currentTime=nuevoTiempo;
        progreso.style.width=ratonX+'px';
    }
}
```

```
function accionPlay() {
    if(!medio.paused && !medio.ended) {
        medio.pause();
        play.value='\u25BA';
        window.clearInterval(bucle);
    } else {
        medio.play();
        play.value='||';
        bucle=setInterval(redimensionaBarra, 1000);
    }
}



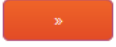
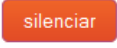

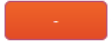

function iniciar() {
    maximo=700;
    medio=document.getElementById('medio');
    barra=document.getElementById('barra');
    progreso=document.getElementById('progreso');
    play=document.getElementById('play');
    /* obtener los objetos del resto de elementos necesarios */
    play.addEventListener('click', accionPlay, false);
    /* crear los manejadores de eventos para el resto de botones */
    barra.addEventListener('click', desplazarMedio, false);
}

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


Las funcionalidades que hay implementadas son:

- 🚦 El botón play/pause .   Si pulsamos el botón play se iniciará la reproducción del vídeo y si volvemos a pulsarlo se pausará.
- 🚦 El control de reproducción mediante la barra que tenemos debajo del vídeo. Si pulsamos en cualquier punto de la barra, la reproducción del vídeo se moverá hasta ese punto.

Las funcionalidades que faltan por implementar y que debéis implementar en este ejercicio son:

- 🚦 Al pulsar el botón reiniciar () si el vídeo está iniciado se reiniciará, es decir, comenzará a reproducirse de nuevo desde el principio.
- 🚦 Al pulsar el botón retrasar () la reproducción del vídeo saltará 5 segundos hacia atrás.
- 🚦 Al pulsar el botón adelantar () la reproducción del vídeo saltará 5 segundos hacia delante.
- 🚦 Al pulsar el botón silenciar () el sonido del vídeo se desactivará y el texto del botón cambiará a "escuchar" (). Al volver a pulsar el botón se activará el sonido y se cambiará de nuevo el texto a "silenciar".
- 🚦 Al pulsar el botón menosVolumen () se bajará el volumen del vídeo 0.1 puntos. Si el volumen llega a 0 puntos no haremos nada.
- 🚦 Al pulsar el botón masVolumen () se subirá el volumen del vídeo

0.1 puntos. Si el volumen llega a 1 punto no haremos nada.

Con esto habremos terminado los ejercicios del tema. Recuerda que para realizar la entrega del ejercicio, debes comprimir la carpeta curso dentro de un fichero llamado:

nombre-alumno-tema3.zip

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