

Implantación de Contenido Multimedia

Diseño de Interfaces Web

2º Desarrollo de Aplicaciones Web

Ana Gloria Palacios Caminero

Tabla de contenido

1.- Vídeo	3
1.1.- Conceptos básicos.....	3
1.1.1.- Códecs.....	3
1.1.2.- Contenedores	5
1.2.- Vídeo en HTML5.....	8
1.2.1.- Inserción de vídeo en HTML5	8
1.2.2.- Tipos MIME	9
1.2.2.- Controlando la reproducción multimedia	10
1.3.- Vídeo de fondo a pantalla completa	15
1.3.1.- Código fuente	16
1.4 - Media Queries en línea	18
2.- Audio	20
3.- Imágenes.....	21
3.1.- FORMATOS de iMAGEN	21
3.1.1.- JPEG.....	21
3.1.2.- GIF	22
3.1.3.- PNG	23
3.1.4.- WEBP.....	24
3.2.- El elemento picture.....	24
3.3.- Fuentes de iconos	27
4.- El elemento <canvas> de HTML5.....	29
4.1.- Conceptos básicos.....	29

4.2.- Formas básicas en el canvas	30
5.2.1.- Rectángulos.....	30
4.2.2.- Rutas (paths)	30
4.2.3.- Aplicando colores y estilos.....	31
4.2.3.- Gradientes.....	33
4.3.- Dibujando texto	33

1.- VÍDEO

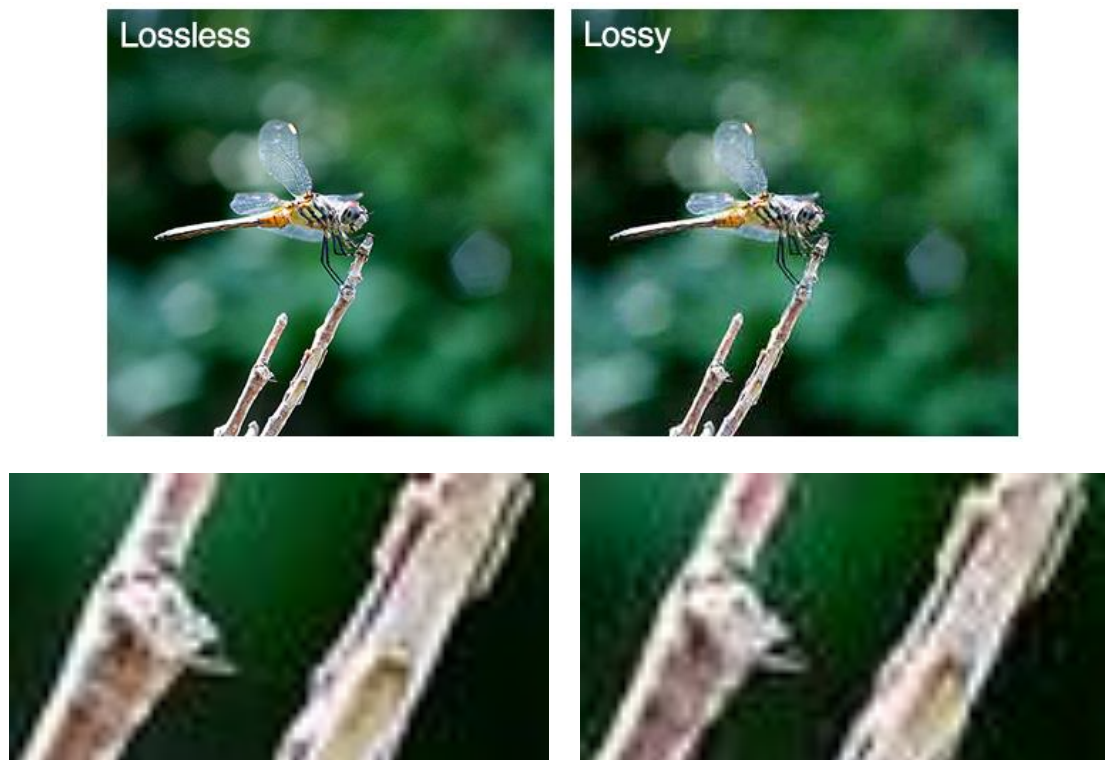
1.1.- CONCEPTOS BÁSICOS

1.1.1.- CÓDECS

Un **códec** es una fórmula matemática que reduce el tamaño de un fichero de vídeo o audio. Los códecs de vídeo más relevantes en HTML5 son **H.264, Theora, VP8 y VP9**.

Los tipos de compresión que un códec aplica a un fichero de vídeo se pueden dividir en dos grandes grupos: **sin pérdida (losslesscompression) y con pérdida (lossyCompression)**.

- **Compresión sin pérdida:** es el proceso de **comprimir la información** en un fichero más pequeño **sin que se produzca ninguna pérdida de información**.
- **Compresión con pérdida:** también se denomina en ocasiones *codificación perceptual*, y es el proceso de **reducir el tamaño de un elemento multimedia eliminando parte de la información**. Los algoritmos que se utilizan para esto son complejos e intentan preservar la experiencia perceptual del usuario a la vez que descartan todos los datos que sean posibles.



BITRATE Y CALIDAD

La **calidad de un vídeo digital** está determinada por la cantidad de información codificada (**bitrate o tasa de bits**) y el tipo de compresión de vídeo (**códec**) utilizado.

El **bitrate** es literalmente el **número de bits por segundo de vídeo (y/o audio) utilizado en la codificación**. Para un determinado códec, un mayor bitrate implica una mayor calidad. Para una misma duración, un mayor bitrate implica un fichero mayor.

Para hacernos una idea las cámaras DV graban vídeo y audio a 25 Mbit/s, los DVDs son codificados entre 6 y 9 Mbit/s, el vídeo por Internet depende en gran medida del ancho de banda disponible, pero un valor habitual es 700 kbit/s

Diferentes tipos de video requieren diferentes bitrates para obtener el mismo nivel de percepción de calidad. Un vídeo con muchos cortes y muchos movimientos de cámara requieren más información para describir el vídeo. Así por ejemplo una película de acción requerirá un mayor bitrate que un documental con muchas imágenes estáticas.

La mayoría de los códecs actuales permiten un **bitrate variable**. Esto significa que el bitrate puede variar a lo largo del tiempo en función del detalle requerido en el vídeo. De esta forma el códec de vídeo utilizará más bits para codificar 10 segundos de película con muchos cambios que en codificar 10 segundos de una escena relativamente estática.

CÓDECS Y CALIDAD

Los códecs reducen el **bitrate necesario** para un fichero multimedia describiéndolo de una forma más eficiente.

- Los **códex de vídeo** describen los cambios entre una imagen (**frame**) y la siguiente, en lugar de describir cada frame independientemente.
- Los **códex de audio** ignoran ciertas frecuencias que el oído humano no puede percibir.

Estas simples técnicas pueden reducir enormemente el bitrate y el tamaño del fichero. Cuanto más sofisticadas sean las técnicas utilizadas mayor será la reducción de tamaño y esto es lo que hace que unos códec sean superiores a otros.

Cuando un códec utiliza técnicas matemáticas complicadas para codificar vídeo es necesario tener suficiente capacidad de procesamiento para decodificar el vídeo a la velocidad suficiente para reproducirlo. Es por esta razón, por lo que en ocasiones, es mejor utilizar un códec más sencillo, por ejemplo, si queremos que se pueda reproducir en ordenadores antiguos.

CÓDECS POPULARES

Algunos de los códec más populares son:

- **H.264 (AVC)**: Es uno de los codecs más utilizados y es compatible con una amplia variedad de dispositivos y plataformas. Tiene una calidad de compresión y es eficiente en tamaño de archivo. Se suele utilizar en archivos MP4 y flujos de transmisión en línea.
- **H.265 (HEVC)**: Es la evolución de H.264 y proporciona una mayor eficiencia de compresión, lo que resulta en tamaños de archivo más pequeños para la misma calidad de video. Sin embargo, la compatibilidad con navegadores y dispositivos puede variar.
- **P9**: Desarrollado por Google, es un codec de código abierto que ofrece una buena eficiencia de compresión. Se usa especialmente en el ecosistema de Google, como YouTube y algunos navegadores como Google Chrome.
- **AV1**: También es un codec de código abierto desarrollado por la Alliance for Open Media. Ofrece una alta eficiencia de compresión y calidad, pero su adopción puede depender de la compatibilidad del navegador y del hardware.
- **MPEG-2**: Aunque es un codec más antiguo, todavía se utiliza en algunos contextos, especialmente en la transmisión de televisión por cable y satélite. No es tan eficiente en términos de compresión como algunos de los codecs más nuevos.
- **MPEG-4**: Este estándar incluye varios codecs, como DivX y Xvid, y se utiliza en una variedad de aplicaciones, desde transmisión en línea hasta dispositivos de almacenamiento multimedia.
- **WebM (VP8 y VP9)**: Es un formato de archivo de video y codec de código abierto desarrollado para la web por la Alliance for Open Media. Es compatible con navegadores como Google Chrome y Firefox.

1.1.2.- CONTENEDORES

Un **contenedor** o **wrapper** es un formato de fichero que especifica cómo diferentes flujos de datos pueden ser almacenados o enviados por la red juntos. Permite que datos de audio y vídeo sean almacenados en un único fichero y reproducidos de forma sincronizada.

Además del audio y el vídeo, los contenedores también guardan **meta datos** sobre su contenido, por ejemplo:

- **Tamaño de los frames**
- **Imágenes por segundo** (framerate)
- **Si es mono o estéreo**
- **Frecuencia de muestreo** (samplerate)
- **Información sobre los códecs utilizados para codificar los datos.**

Hay que tener clara la diferencia entre contenedor y códec. Cuando vemos un fichero de vídeo las tres letras de la extensión hacen referencia al contenedor, no al códec. Cuando un fichero termina en .mp4 o .avi, son contenedores que pueden tener diferentes combinaciones de audio y vídeo. Determinados contenedores pueden no funcionar con determinados códecs, y ciertos códecs funcionan mejor con determinados contenedores.

CONTENEDORES POPULARES

Algunos de los contenedores más populares son:

- **Flash Video (.flv):** Macromedia creó originalmente Flash antes de ser adquirida por Adobe en 2005. Flash es un contenedor bastante antiguo que está cayendo en desuso debido a las limitaciones en su tecnología. Como curiosidad Steve Jobs definió este formato como *buggy* (defectuoso), lo cual llevó a una llamativa omisión de su soporte en dispositivos iOS siendo considerado como el comienzo del fin de este formato.

El soporte de vídeo en HTML5 ha acelerado la desaparición de este formato, cuyo uso cada vez es más residual. De hecho en diciembre de 2016 Google anunció que su navegador Chrome bloquearía por defecto los contenidos de Adobe Flash Player¹.

- **Matroska (.mkv):** MKV es un formato en rápido crecimiento que está diseñado para durar muchos años. Este contenedor soporta la mayoría de los formatos de audio y vídeo lo que le hace adaptable y eficiente. Además soporta múltiples ficheros de vídeo, audio y subtítulos, incluso aunque estén codificados en diferentes formatos.

Debido a todas las opciones de que dispone este contenedor, así como su capacidad de recuperación ante errores (que permite reproducir ficheros corruptos), sea erigido como el mejor contenedor disponible actualmente.

- **MP4:** El contenedor MP4 utiliza codificación MPEG-4 o H.264, así como AAC o AC3 para audio. Es ampliamente soportado en la mayoría de los dispositivos y el contenedor más comúnmente utilizado para vídeo online.
- **WebM:** es un formato de contenedor de código abierto desarrollado por Google, basado en Matroska y diseñado específicamente para la web, en concreto para ser utilizado con HTML5. Utiliza los códecs **V P8 o VP9** para vídeo y **Vorbis u Opus** para audio.

¹<https://blog.chromium.org/2016/12/roll-out-plan-for-html5-by-default.html>

Tipos de mime:

- video/webm. Para archivos de video e incluso de audio
- audio/webm. Para archivo de audio.

- **Ogg Theora Vorbis**

Es un contenedor de media Ogg.

- Codec de video: Theora
- Codec de audio: Vorbis

WebM tiene mayor compresión que Ogg.

Tipos de mime:

- audio/ogg --→ Para archivos de audio
- video/ogg--→ Para archivo de video.
- application/ogg--→ Si no se conoce el contenido del contenedor.

- **Ogg Opus**

Es un contenedor Ogg pero utilizan el codec de audio Opus.

Tipos de mime:

- audio/ogg; codecs=opus**

- **Ogg FLAC**

Contenedor Ogg que utiliza un codec de audio FLAC.

- **MP4 H.264 (AAC O MP3)**

Contenedor MP4.

Codec de video : H.264

Codec de audio: AAC.

Navegadores que lo soportan: Internet Explorer, Safari y Chrome.

Se puede encontrar contenedores MP4 con un codec de audio MP3. Solo lo soportan Internet Explorer y Chrome

- **MP4 FLAC**

Contenedor de MP4 pero tiene un codec de audio FLAC.

TABLA 1- Compatibilidad de los diferentes formatos de video con los navegadores:

	MP4	OGG	WebM
Firefox	-	+3.5	+4.0
Chrome	+3.0	+3.0	+3.0
Opera	-	+10.5	+10.6
Safari	+3.1	-	-
IE Explorer	+IE9	-	-

1.2.- VÍDEO EN HTML5

1.2.1.- INSERCIÓN DE VÍDEO EN HTML5

La etiqueta para insertar vídeo en un documento HTML5 es `<video>` y su sintaxis es bastante sencilla. El siguiente código muestra un vídeo de ejemplo disponible en el sitio Web de Theora.

```
<video src="http://v2v.cc/~j/theora_testsuite/320x240.ogg" controls>
  Tu navegador no implementa el elemento <code>video</code>.
</video>
```

El atributo **src** apunta en este caso a una URL pero puede hacer referencia también a un fichero en el sistema de ficheros local.

Observa también como el texto que hay entre la etiqueta `<video>` y su cierre se muestra únicamente en los navegadores que no soporten la reproducción de vídeo, aunque como se ve en la siguiente imagen es una etiqueta bien soportada por la mayoría de los navegadores.

Element					
<video>	4.0	9.0	3.5	4.0	10.5

Esta etiqueta admite los siguientes atributos:

- **controls**: muestra los controles estándar de HTML5 para audio en una página web.
- **autoplay**: hace que el vídeo se reproduzca automáticamente. Hay que tener en cuenta que muchos dispositivos móviles ignoran este atributo para evitar un consumo excesivo de datos.
- **muted**: el vídeo tiene el sonido deshabilitado por defecto. Es altamente recomendable sobre todo cuando el vídeo se reproduce automáticamente.
- **loop**: el vídeo se reproduce en un bucle, volviendo a comenzar cuando llegue a su fin.
- **preload**: este atributo indica al navegador qué debe hacer respecto a la precarga del archivo de vídeo. Esta es una característica de muchos navegadores que descargan por anticipado los vídeos para poder disponer inmediatamente de ellos en el momento en que el usuario solicite su reproducción. Este atributo puede tener tres valores:
 - **"none"**: el archivo no es descargado hasta que se solicita su reproducción.

- “auto”: descarga automáticamente el archivo multimedia.
- “metadata”: descarga automáticamente únicamente los metadatos del fichero.
- **poster**: en este atributo se indica una imagen que se mostrará en la pantalla del reproductor hasta que el vídeo comienza su reproducción. Es aconsejable que esta misma imagen también esté en los primeros frames del vídeo para que haya una transición suave.

Esta etiqueta también permite especificar diversos formatos de fichero de vídeo para que el navegador reproduzca aquel que soporte. Para ello es necesario utilizar la etiqueta <source> de la siguiente forma:

```
<video controls>
  <source src="foo.ogg" type="video/ogg">
  <source src="foo.mp4" type="video/mp4">
  Tu navegador no implementa el elemento <code>video</code>.
</video>
```

En este ejemplo el navegador utilizará en primer lugar el fichero en formato .ogg. Si no soportara dicho formato utilizaría el fichero .mp4.

En la siguiente imagen puedes ver un listado de los códecs soportados por las diferentes versiones de los navegadores extraído de la página de MDN²:

Desktop	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	3.0	3.5 (1.9.1)	9.0	10.50	3.1
caution: PCM in WAV	(yes)	3.5 (1.9.1)	No support	10.50	3.1
caution: Vorbis in WebM	(yes)	4.0 (2.0)	No support	10.60	3.11 ¹
caution: Streaming Vorbis/Opus in WebM via MSE	?	36.0 (36.0 ²)	?	?	?
caution: Vorbis in Ogg	(yes)	3.5 (1.9.1)	No support	10.50	No support
caution: MP3	(yes) ⁴	(yes) ⁵	9.0	(yes)	3.1
caution: MP3 in MP4	?	?	?	?	(yes)
caution: AAC in MP4	(yes) ⁶	(yes) ⁷	9.0	(yes)	3.1
caution: Opus in Ogg	27.0	15.0 (15.0)	?	?	?
caution: FLAC	No support	51 (51)	No support	No support	No support
caution: FLAC in Ogg	No support	51 (51)	No support	No support	No support
video: VP8 and Vorbis in WebM	6.0	4.0 (2.0)	9.0 ⁸	10.60	3.1 ⁹
video: VP9 and Opus in WebM	29.0	28.0 (28.0 ¹⁰)	?	(yes)	?
video: Streaming WebM via MSE	?	42.0 (42.0 ¹¹)	?	?	?
video: Theora and Vorbis in Ogg	(yes)	3.5 (1.9.1)	No support	10.50	No support
video: H.264 and MP3 in MP4	(yes) ⁴	(yes) ¹²	9.0	(yes)	(yes)
video: H.264 and AAC in MP4	(yes) ⁴	(yes) ¹³	9.0	(yes)	3.1
video: FLAC in MP4	?	51 (51)	?	?	?
any other format	No support	No support	No support	No support	3.11 ¹⁰

También es posible especificar qué codecs requiere el archivo multimedia, de forma que el navegador tomará decisiones más inteligentes.

```
<video width="640" height="360" controls autoplay preload>
  <source src="mivideo.mp4" type='video/mp4; codecs="avc1,mp4a"' />
  <source src="mivideo.ogv" type='video/ogg; codecs="theora,vorbis"' />
  <source src="mivideo.webm" type='video/webm; codecs="vp8,vorbis"' />
</video>
```

Aquí estamos indicando para cada contenedor los códecs que utiliza. De esta forma si un navegador soporta el contenedor pero no esos códecs exactamente pasará al siguiente sin necesidad de descargarlo.

1.2.2.- TIPOS MIME

Una cosa que hay que tener en cuenta cuando incluimos vídeo en nuestra página Web es que nuestro servidor debe reconocer los **tipos MIME**. Estos sirven para indicarle al navegador qué tipo de contenidos está descargando. Cada vez que el navegador solicita una página, el servidor

²https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats

envía las cabeceras (*headers*) antes de enviar cualquier fichero y entre otros campos indica el tipo de fichero de que se trata.

```
▼ Response Headers    view source
Cache-Control: private, no-cache, no-cache=Set-Cookie, no-store, proxy-revalidate
Connection: keep-alive
Content-Length: 43
Content-Type: image/gif
Date: Mon, 16 Jan 2017 10:21:42 GMT
Expires: Mon, 01 Jan 1990 00:00:00 GMT
Pragma: no-cache
```

En el caso de la reproducción de vídeo en HTML hay 3 tipos MIME que nos afectan. Para asegurarnos de que el servidor no tiene problemas para reproducirlos debemos incluir las siguientes líneas en el fichero `.htaccess`

```
AddType video/ogg .ogv
```

```
AddType video/mp4 .mp4
```

```
AddType video/webm .webm
```

1.2.2.- CONTROLANDO LA REPRODUCCIÓN MULTIMEDIA

Otra de las ventajas de utilizar vídeo en HTML5 frente a las soluciones propuestas por terceros como Flash es que el vídeo está realmente integrado en la página y no es únicamente un plugin incrustado y cerrado. Esto se traduce en que, al igual que con otros elementos de la página, es posible aplicarle estilos e incluso realizar determinadas operaciones con él utilizando JavaScript.

Como vimos HTML5 permite crear unos controles por defecto para controlar la reproducción del vídeo, pero su aspecto visual está determinado por el navegador. Por ejemplo en las siguientes imágenes se pueden ver los controles por defecto en los navegadores Mozilla Firefox y Chrome.

MozillaFirefox



Chrome



Esto puede ser un problema cuando queremos controlar al máximo la apariencia de la página. Afortunadamente es muy fácil crear unos controles personalizados con un poco de JavaScript y CSS.

Vamos paso a paso como hacerlo.

1.2.2.1.- DISEÑANDO EL CONTROL

En este ejemplo vamos a crear un control muy sencillo que únicamente tendrá las siguientes funcionalidades:

- Botón de inicio y parada

- Contador que indique el tiempo que lleva reproducido el vídeo
- Botón de *mute* para apagar el sonido.

El código HTML que utilizaremos para los controles es el siguiente:

```
<div id="controls" class="hidden">
  <a id="playPause">Play</a>
  <span id="timer">00:00</span>
  <a id="muteUnmute">Mute</a>
</div>
```

Algunas cosas que tenemos que tener en cuenta antes de comenzar:

- El texto de los botones de reproducción y de *mute* lo eliminaremos posteriormente.
- Este texto lo reemplazaremos por una imagen que represente los estados de los botones.
- Estos estados los indicaremos mediante clases que añadiremos o eliminaremos mediante JavaScript
- Aquí no lo hacemos pero lo ideal sería colocar los controles mediante posicionamiento absoluto para que se solapen sobre el recuadro del vídeo.
- Los controles tendrán una opacidad del 50% que se incrementará al 100% cuando el usuario pase el cursor por encima.
- Por defecto los controles estarán ocultos mediante una clase "hidden" que aplicará la propiedad display: none.

1.2.2.2.- ELEMENTOS DE LA API

El primer paso será crear una serie de **variables en JavaScript** para poder hacer referencia a los controles. El código se muestra a continuación:

```
var videoEl = document.getElementsByTagName('video')[0],
    playPauseBtn = document.getElementById('playPause'),
    vidControls = document.getElementById('controls'),
    muteBtn = document.getElementById('muteUnmute'),
    timeHolder = document.getElementById('timer');
```

Observa que con el código anterior estamos referenciando el vídeo mediante el nombre de etiqueta. La etiqueta `getElementsByTagName()` devuelve un array con todos los elementos con la etiqueta `<video>` y extraemos la primera ocurrencia.

El resto de elementos los referenciamos mediante su identificador.

El siguiente paso será **eliminar los controles por defecto**. Obviamente la forma más fácil de conseguir esto es no poniendo el atributo `controls` en el HTML pero tiene un inconveniente: si un usuario tiene deshabilitado JavaScript en su navegador se encontrará con que no tiene ninguna forma de controlar el vídeo. Eliminando los controles desde JavaScript garantizamos que de una forma u otra el usuario disponga de los controles.

El código para eliminar los controles en JavaScript es el siguiente:

```
videoEl.removeAttribute('controls');
```

El siguiente paso es mostrar los controles que como recordarás los teníamos ocultos por defecto. El código para ello es el siguiente:

```
videoEl.addEventListener('canplaythrough', function() {  
    vidControls.classList.remove('hidden');  
}, false);
```

En este caso estamos utilizando la función `addEventListener()` sobre el elemento vídeo. Esta función se queda a la espera de que se produzca el evento indicado por el primer parámetro sobre el vídeo, y cuando se produzca dicho evento ejecuta la función que se le pasa como segundo parámetro.

HTML5 dispone de muchos eventos pero el que utilizamos aquí es `canplaythrough`, el cual se lanza cuando el navegador considera que, teniendo en cuenta la velocidad de descarga del vídeo, se puede comenzar su reproducción de forma que pueda verse de forma continuada hasta el final sin necesidad de para para hacer *buffering*.

Esto quiere decir que nuestros controles se mostrarán cuando haya garantías de que el vídeo se va a poder reproducir de continuo desde el momento en que hay controles disponibles.

2.2.2.3.- REPRODUCIR Y PARAR EL VÍDEO

Ahora nos falta añadir la funcionalidad a nuestros botones. Comencemos con el botón Play/Pause.

```
playPauseBtn.addEventListener('click', function() {  
    if (videoEl.paused) {  
        videoEl.play();  
    } else {  
        videoEl.pause();  
    }  
}, false);
```

Aquí estamos utilizando la propiedad `paused` del vídeo que devuelve True si el vídeo está parado y False en caso de que se esté reproduciendo. Si el vídeo no se está reproduciendo utilizamos la función `play()` para comenzar la reproducción. Si se está reproduciendo entonces utilizamos la función `pause()` para pararlo.

Con este código controlamos la reproducción del vídeo pero no modificamos el aspecto del botón. Vamos a ver como sería el código para ello:

```
videoEl.addEventListener('play', function() {  
    playPauseBtn.classList.add('playing');  
}, false);  
  
videoEl.addEventListener('pause', function() {  
    playPauseBtn.classList.remove('playing');  
}, false);
```

Ahora utilizamos los eventos **play** y **pause**, que son lanzados cuando el vídeo comienza a reproducirse y cuando es parado. En función de estos eventos añadimos o quitamos la clase **playing** que es la que, mediante CSS, controla el icono mostrado en el botón.

1.2.2.4.- ACTIVAR Y DESACTIVAR EL SONIDO DEL VÍDEO

Para implementar esta funcionalidad tenemos que definir funciones muy parecidas a las del punto anterior. En primer lugar añadimos el *listener* que vigila los *clicks* del ratón sobre el botón.

```
muteBtn.addEventListener('click', function() {  
    if (videoEl.muted) {  
        videoEl.muted=false;  
    } else {  
        videoEl.muted=true;  
    }  
}, false);
```

Como puedes ver utilizamos la propiedad **muted** para comprobar si el sonido está activado o desactivado y le aplicamos el valor en consecuencia.

Para cambiar la apariencia del botón no hay ningún evento que nos indique que el sonido se ha silenciado pero sí tenemos un evento que se activa cuando se cambia el volumen llamado **volumechange**.

```
videoEl.addEventListener('volumeChange', function() {  
    if (videoEl.muted) {  
        muteBtn.classList.add('muted');  
    } else {  
        muteBtn.classList.remove('muted');  
    }  
}, false);
```

1.2.2.5.- DETECTAR CUANDO EL VÍDEO HA FINALIZADO

Tal y como lo tenemos hasta ahora, cuando el vídeo se ha reproducir por completo se quedará mostrando el último *frame*. Un detalle interesante sería dejar el vídeo en el primer *frame* para el caso de que el usuario quiera volver a reproducirlo. Esto lo podemos conseguir con el siguiente código:

```
videoEl.addEventListener('ended', function() {  
    videoEl.currentTime = 0;  
}, false);
```

Seguro que a estas alturas ya entiendes perfectamente el código. El evento `ended` es lanzado cuando el vídeo ha finalizado su reproducción y la propiedad `currentTime` indica el frame actual del vídeo.

2.2.2.6.- ACTUALIZAR EL TIEMPO A MEDIDA QUE EL VÍDEO SE REPRODUCE

Solo nos queda un elemento de nuestros controles personalizados, el indicador de tiempo que ha transcurrido del vídeo, que como recordarás insertamos en un elemento `` con el identificador `timer`.

En este caso utilizaremos el evento `timeupdate`, que es disparado cada vez que el tiempo del vídeo cambia. Cada vez que se dispare este evento utilizaremos la propiedad `currentTime` que vimos en el punto anterior para mostrar el tiempo transcurrido del vídeo.

```
videoEl.addEventListener('timeupdate', function() {  
    timeHolder.innerHTML = videoEl.currentTime;  
})
```

Si has probado el código anterior verás que el resultado no es el esperado ya que el tiempo se muestra en segundos con decimales. Para solventar este problema deberás convertir ese valor a un formato más legible, pero no es algo que solucionaremos ahora sino que ya lo harás como parte de la práctica que realizaremos.

1.2.2.7.- EVENTOS DE LA API

Hemos visto hasta ahora algunos de los eventos de la API pero realmente hay muchos más eventos que podemos vigilar.

- **canplay**: similar a `canplaythrough`, pero se dispara en el momento en que el vídeo se puede reproducir, incluso aunque únicamente se hayan descargado unos pocos frames.
- **error**: es disparado cuando se ha encontrado algún error. También hay una propiedad `error`.
- **loadeddata**: el primer frame del vídeo se ha descargado.
- **loadedmetadata**: este evento es disparado una vez que se hayan descargado los metadatos del vídeo. Los metadatos pueden incluir dimensiones, duración y pistas de texto.
- **playing**: indica que el vídeo ha comenzado a reproducirse. La diferencia entre `playing` y `play` es que `play` no se dispara cuando el atributo `loop` está establecido y el vídeo comienza a reproducirse y `playing` si lo hace.
- **seeking**: es enviado cuando una operación de búsqueda comienza, por ejemplo cuando un usuario ha movido la barra de búsqueda a otra posición del vídeo.

- **seeked**: cuando la operación de búsqueda ha finalizado.

1.2.2.8.- PROPIEDADES DE LA API

En cuanto a otras propiedades tenemos las siguientes:

- **playbackRate**: el valor por defecto es 1 y puede ser cambiado para acelerar o frenar la reproducción.
- **src**: devuelve la URL del vídeo que está reproduciendo. Únicamente funciona cuando el atributo `src` se ha definido.
- **currentSrc**: contiene la URL del vídeo independientemente de que este se haya indicado mediante el atributo `src` o el elemento `source`.
- **readyState**: contiene un valor numérico entre 0 y 4 que representa hasta qué punto está preparado el vídeo para ser reproducido. Por ejemplo un valor de 1 indica que los metadatos ya se han descargado mientras que un valor de 4 equivale a la condición que dispara el evento `canplaythrough`.
- **duration**: devuelve la duración del vídeo en segundos.
- **buffered**: representa el rango de tiempo de vídeo que se ha descargado y está disponible para ser reproducido.
- **videoWidth, videoHeight**: devuelve las dimensiones intrínsecas del vídeo, es decir, el tamaño con el que fue codificado y no al que se está mostrando.

https://developer.mozilla.org/es/docs/Web/HTML/Usando_audio_y_video_con_HTML5

Vídeos de ejemplo: <http://techslides.com/sample-webm-ogg-and-mp4-video-files-for-html5>

1.3.- VÍDEO DE FONDO A PANTALLA COMPLETA

Un efecto muy interesante con el que podemos dotar a nuestra página web es la reproducción de vídeo a pantalla completa de fondo de la página. Este recurso puede servir para realzar nuestra página en determinadas ocasiones, siempre y cuando sigamos una serie de pautas:

- No hay que utilizar esta técnica simplemente porque es posible, **solamente debe utilizarse si sirve para amplificar el mensaje del sitio**.
- El vídeo debe estar marcado como `autoplay`, pero **debe estar silenciado por defecto** y disponer de un botón para activar el sonido si fuera necesario, aunque lo ideal sería que no tuviera sonido.
- El vídeo debe incluir una **imagen (placeholder)** para que se muestre en los navegadores que no soporten **HTML5**. Esta imagen también se podría utilizar de fondo en dispositivos móviles que no permiten el *autoplay*.
- **La longitud es muy importante**: un vídeo muy corto resultará repetitivo (piensa que lo normal es que el vídeo se muestre en *loop*), mientras que un vídeo muy largo sería ya

una narrativa y tal vez su sitio no sería el fondo de la página. Una longitud recomendable puede ser entre 12 y 20 segundos.

- No hay que olvidar la accesibilidad: **cualquier texto que se muestre sobre el vídeo tiene que contrastar** lo suficiente para que se vea claramente. Además los usuarios tienen que disponer de un botón que les permita parar la reproducción del vídeo.
- Otro factor a tener en cuenta es el ancho de banda, **el vídeo tiene que ocupar poco espacio** y estar comprimido todo lo que sea posible. Lo ideal sería incluir diferentes versiones del vídeo con diferentes resoluciones para diferentes tamaños de pantalla. Esto se puede conseguir por ejemplo con *media queries en línea* como veremos posteriormente.

1.3.1.- CÓDIGO FUENTE

La inserción del vídeo en HTML hemos de hacerla de forma similar a como hemos visto en ejemplos anteriores:

```
<video autoplay muted loop poster="cover.jpg" id="bgvid">
  <source src="small.webm" type="video/webm">
  <source src="small.ogv" type="video/ogg">
</video>
```

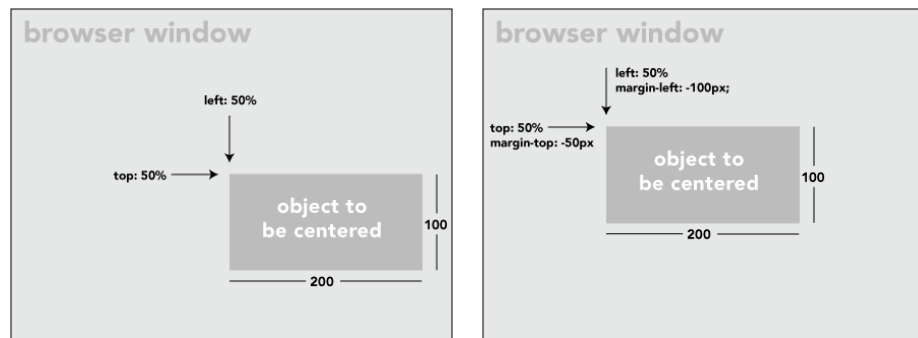
Ahora en el CSS hemos de indicar que el vídeo debe ocupar todo el tamaño de la ventana del navegador:

```
video#bgvid {
  position: fixed;
  min-width: 100%;
  min-height: 100%;
  width: auto;
  height: auto;
  z-index: -100;
  top: 50%;
  left: 50%;
  transform: translateX(-50%) translateY(-50%);
  background: url(cover.jpg) no-repeat;
  background-size: cover;
```

Analicemos lo más destacable del código anterior:

- Utilizamos **posicionamiento fijo** para asegurarnos de que el vídeo siempre está en la misma posición. Esto por ejemplo nos permitiría hacer scroll sobre el resto de elementos de la página si ocuparan más del alto de la ventana del navegador mientras el vídeo se mantiene en la misma posición.
- Indicando un ancho y un alto como *auto* indicamos que el vídeo conserve su tamaño nativo, sin embargo, al combinarlo con un valor del 100% en el ancho y alto mínimo forzamos a que el vídeo crezca hasta ocupar el 100% de la ventana del navegador si fuera más pequeño. Si fuera mayor simplemente se recortará y solo se mostrará la parte correspondiente a la ventana del navegador.

- Ponemos un valor muy bajo en la propiedad *z-index* para asegurarnos de que el vídeo se sitúa detrás de todos los elementos.
- Seguro que te ha llamado la atención que modifiquemos la posición del vídeo un 50% del ancho y del alto mediante *top* y *left* para volver a desplazarlo otra vez a su posición inicial ese 50%. El objetivo de esto es conseguir que el vídeo quede centrado de forma que cuando el vídeo sea mayor que la ventana del navegador veamos la parte central del mismo y no la parte superior izquierda.
- Para comprenderlo te tienes que tener en cuenta cuál es el elemento de referencia para el posicionamiento absoluto y cuál lo es en las transformaciones.
 - En el caso del posicionamiento la referencia es el elemento padre; por lo tanto estamos moviendo el vídeo de forma que su esquina superior izquierda se encuentra exactamente en el centro de la ventana.
 - En el caso de las transformaciones la referencia es el propio objeto, por lo que estamos moviendo el vídeo una distancia igual a la mitad de su propio tamaño haciendo que el centro del vídeo quede exactamente sobre el centro de la ventana del navegador.



- Finalmente fíjate que indicamos una imagen de fondo que se mostrará en caso de que el navegador no soporte HTML5.
- También hay que tener en cuenta que por claridad no se han incluido las versiones prefijadas de la propiedad *transform*, pero su inclusión sería deseable si buscamos compatibilidad con todos los navegadores.
- El punto anterior no es totalmente cierto ya que el código no funcionaría en versiones de Internet Explorer anteriores o iguales a IE8, ya que estas versiones no solo no reconocen HTML5 sino que cuando encuentran una etiqueta desconocida la ignoran. Este efecto indeseable lo podemos solucionar con un poco de JavaScript:

```
<!-- [if lt IE 9]>
<script>
  document.createElement('video');
</script>
<![endif]-->
```

Con el código anterior simplemente creamos una etiqueta nueva llamada *video* evitando de esta manera que Internet Explorer la ignore. Una vez creada solamente faltaría indicarle que se debe visualizar como elemento en bloque.

```
video {  
    display: block;  
}
```

<http://thenewcode.com/777/Create-Fullscreen-HTML5-Page-Background-Video>

1.4 - MEDIA QUERIES EN LÍNEA

Cuando insertamos vídeo en una página hemos de seleccionar muy cuidadosamente el tamaño del vídeo para encontrar un compromiso entre calidad y tamaño del mismo: un vídeo muy comprimido consumirá muy poco ancho de banda pero tendrá una muy baja calidad. En cambio un vídeo menos comprimido tendrá mayor calidad pero tendrá una penalización por el tamaño de descarga.

Esto se complica mucho más cuando tenemos en cuenta que los usuarios de nuestra página tendrán dispositivos con resoluciones de pantalla muy variadas.

Para solucionar este problema disponemos del **atributo media** para la etiqueta *source*, que nos va a permitir especificar diferentes resoluciones de vídeo en función del tamaño de la ventana del usuario.

El código podría ser parecido al que se muestra a continuación:

```
<video controls>  
  <source src="the-sky-is-calling-large.mp4"  
    media="screen and (min-width:800px)">  
  <source src="the-sky-is-calling-large.webm"  
    media="screen and (min-width:800px)">  
  <source src="the-sky-is-calling-small.mp4"  
    media="screen and (max-width:799px)">  
  <source src="the-sky-is-calling-small.webm"  
    media="screen and (max-width:799px)">  
</video>
```

Como puedes ver la sintaxis del atributo *media* es igual que la de las *queries* CSS que ya vimos por encima cuando estudiamos el posicionamiento Flex y que veremos en detalle en temas posteriores.

Básicamente lo que estamos haciendo en cada etiqueta *<source>* es indicar las características de tamaño que debe cumplir la pantalla para que se cargue un vídeo u otro. En concreto se cargará el vídeo pequeño cuando el tamaño de la pantalla sea inferior a 800 píxeles y el vídeo grande en caso contrario.

Por supuesto todo esto no sirve de nada si el propio elemento de vídeo es *responsive*. Si por ejemplo queremos que el vídeo ocupe el 80% del ancho de la ventana del navegador podemos hacerlo plenamente *responsive* con el siguiente código:

```
video {  
  width: 80%;  
}  
video source {  
  width: 100%;  
  height: auto;  
}
```

Un inconveniente del código anterior es que la consulta se realiza cuando se descarga la página, por lo que el vídeo usado es el más acorde con el tamaño de la ventana del navegador, aunque luego cambie.

Para evitarlo se puede utilizar el tamaño del dispositivo en vez del de la ventana, por lo que la selección del tamaño del fichero será independiente del tamaño de la ventana del navegador en ese momento.

```
<video controls>  
  <source src="the-sky-is-calling-large.mp4"  
    media="screen and (min-device-width:801px)">  
  <source src="the-sky-is-calling-large.webm"  
    media="screen and (min-device-width:801px)">  
  <source src="the-sky-is-calling-small.mp4"  
    media="screen and (max-device-width:800px)">  
  <source src="the-sky-is-calling-small.webm"  
    media="screen and (max-device-width:800px)">  
</video>
```

<http://thenewcode.com/820/Make-HTML5-Video-Adaptive-With-Inline-Media-Queries>

2.- AUDIO

La inserción de audio en HTML5 es muy similar a la inserción de vídeo, aunque en este caso utilizaremos la etiqueta **<audio>**.

```
<audio src="musica.mp3"></audio>
```

En cuanto a los atributos soportados son los mismos que en el caso del vídeo:

- **autoplay**: comienza la reproducción automáticamente, aunque recuerda que esto no está recomendado en ningún caso.
- **loop**: el audio se reproduce de forma iterativa en un bucle.
- **preload**: indica qué se va a precargar, siendo los posibles valores *auto*, *metadata* y *none*.
- **controls**: para incluir controles predeterminados.

También podemos indicar diversos formatos para el caso de que el navegador no soporte alguno de ellos:

```
<audio controls>  
  <source src="audio.ogg" type="audio/ogg">  
  <source src="audio.mp3" type="audio/mpeg">  
  <source src="audio.wav" type="audio/wav">  
</audio>
```

3.- IMÁGENES

3.1.- FORMATOS DE IMAGEN

3.1.1.- JPEG

JPEG (JointPhotographicExpertsGroup) es uno de los formatos de imágenes más populares. Es un formato **con pérdida (lossy)**, lo que quiere decir que las imágenes sufren una pérdida de calidad durante el proceso de compresión. El nivel de compresión se puede especificar al generar el fichero JPEG de forma que cuanto más pequeño queramos que sea el fichero resultante mayor será la pérdida de calidad.

Imagen original	JPEG con compresión media
	

Como se puede ver en las imágenes anteriores la pérdida de calidad no es tan evidente a primera vista. Pero una observación más detallada nos mostrará algunos defectos en la imagen: los colores son más apagados, las líneas no están tan definidas y la imagen en general tiene más ruido. Además, si hacemos zoom veremos que aparecen los denominados **artefactos**.



Ventajas de JPEG:

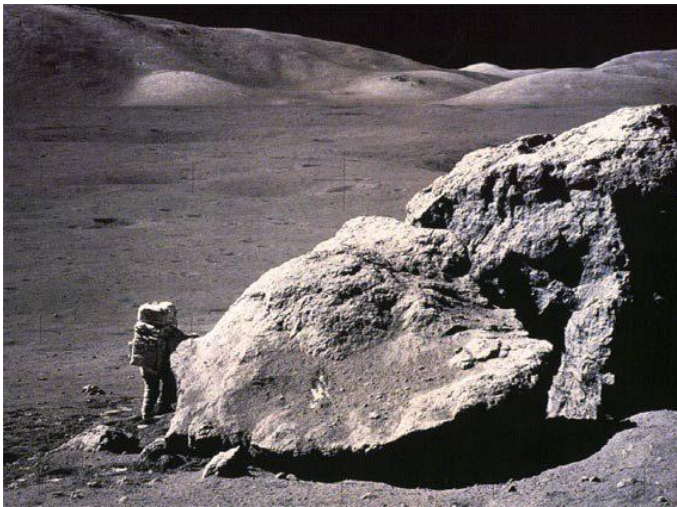
- Tiene color de 24 bits, lo que quiere decir 16 millones de colores
- Riqueza de colores, es un formato excelente cuando queremos resaltar los colores de una imagen.
- Ampliamente utilizado y soportado.

Inconvenientes de JPEG:

- Se pierde información de la imagen
- Aparición de artefactos
- No permite animaciones
- No soporta transparencias

3.1.2.- GIF

GIF (GraphicsInterchangeFormat) es un formato gráfico de paleta. Esto quiere decir que no representa cada color mediante un valor numérico en base a sus componentes (por ejemplo JPEG guarda las componenes RGB de cada color), sino que tiene una paleta de 256 colores. El valor de cada píxel es uno de los colores de la paleta.



GIF comprime las imágenes de dos maneras:

- Al reducir el número de colores necesitamos menos bits por cada píxel. Así, mientras que JPEG requiere 24 bits por cada píxel (8 para cada componente RGB), GIF únicamente necesita 8 bits para referenciar el color de la paleta que utiliza.
- Por otro lado reemplaza múltiples repeticiones de un patrón en uno solo. Por ejemplo en lugar de almacenar cinco tipos de azul, almacena únicamente uno.

GIF es indicado para gráficos, diagramas, dibujos y logotipos con relativamente pocos colores. Además es muy utilizado cuando queremos imágenes animadas.

Comparado con JPEG no tiene pérdida y tiene mayor nivel de compresión cuando trabajamos con imágenes con muy pocos colores, sin embargo en imágenes con muchos colores se pierden la mayoría de los colores.

Otra ventaja de las imágenes GIF es el **entrelazado**, dando la ilusión de que los gráficos cargan más rápido.

Ventajas de los GIF:

- Soporta transparencias
- Permite animaciones
- Compresión sin pérdida de la calidad, aunque si en el número de colores.
- Bueno para imágenes con colores limitados, o con regiones con colores lisos.

Inconvenientes de los GIF:

- Solo soporta 256 colores
- Es el formato más antiguo en la red y no ha sido actualizado desde entonces. En ocasiones el tamaño es mayor que PNG.

<http://1stwebdesigner.com/image-file-types/>

<https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/image-optimization?hl=es-419>

3.1.3.- PNG

PNG (Portable Network Graphics) es un formato de imagen sin pérdida diseñado específicamente para la web. Al igual que JPG también tiene 16 millones colores. Además, soporta transparencia de 8 bits.

Ventajas de PNG:

- No tiene pérdida, así que no pierde nada de calidad tras la compresión.
- Es mucho mejor que GIF, generando archivos más pequeños.
- Permite transparencias en las imágenes.
- Permite la inclusión de metadatos, como información de texto, derechos de autor...

Inconvenientes de PNG:

- No es adecuado para fotografías, generando ficheros muchos mayores que JPEG
- No permite animaciones



3.1.4.- WEBP

WebP es un formato de imagen desarrollado por Google liberado en 2010. Soporta tanto compresión con pérdida como sin pérdida, lo que lo hace adecuado para cualquier tipo de imagen y una gran alternativa frente a formatos como PNG o JPEG.



En el ejemplo superior se puede apreciar que las diferencias de ambas imágenes en cuanto a calidad es imperceptible.

Ventajas:

- Compresión eficiente. Permite tamaños de ficheros más pequeños en comparación con PNG Y JPEG.
- Buena calidad de imagen
- Soporta transparencias.
- Perfil ICC (International Color Consortium), reproducción precisa de colores.
- Soporte en muchos navegadores.

3.2.- EL ELEMENTO PICTURE




En el apartado anterior has visto que es importante la elección del formato cuando vamos a insertar imágenes en un sitio web y que la elección del mismo depende del tipo de imagen de que se trate.

Pero hay otra decisión importante que hemos de tomar con las imágenes y es la elección de la resolución que deben tener dichas imágenes. Como sabrás **la resolución es el número de píxeles que tiene la imagen tanto de ancho como de alto**. Por lo tanto, cuanto mayor sea la resolución mayor será la calidad de la imagen pero también más ocupará, y por tanto, consumirá un mayor ancho de banda. Este problema se agrava, si tenemos en cuenta la gran variedad de dispositivos con que se puede ver una página web: una imagen con mucha resolución, puede ser lo más indicado cuando se ve en un navegador en un monitor HD, pero si se ve en un móvil, está claro que se vería igual, con menos resolución y, por tanto, estamos desaprovechando ancho de banda.

Hemos visto que este problema está resuelto en las imágenes de fondo mediante CSS gracias a las *media queries*, pero hasta hace poco no había ninguna solución para las imágenes insertadas en el código HTML.

Para solucionar este problema en HTML se ha introducido la etiqueta `<picture>`, que permite describir con detalle qué ficheros de imagen queremos descargar según las características de la pantalla del usuario. En concreto esta elección se podrá basar en la resolución de la pantalla, la orientación, la densidad de píxeles o incluso el formato soportado por el navegador.

Esta etiqueta es bastante reciente por lo que su soporte se limita a las versiones más actuales de los navegadores.

Element					
<code><picture></code>	38.0	13.0	38.0	9.1	25.0

4.2.1.- SINTAXIS DEL ELEMENTO PICTURE

La sintaxis del elemento `<picture>` es muy similar a la que vimos con el elemento `<video>`, apoyándose en los elementos `<source>` y utilizando los atributos `media` y `srcset`. A continuación tienes un fragmento de código donde se puede ver su utilización.

```
<picture>
  <source
    media="(min-width: 650px)"
    srcset="imagen_grande.png">
  <source
    media="(min-width: 465px)"
    srcset="imagen_pequena.png">
  
</picture>
```

Como ves se indican una serie de elementos `<source>` que contienen los diferentes ficheros en función de la resolución de la pantalla. En este ejemplo concreto estamos indicando que si el ancho de la pantalla es mayor o igual de 650 píxeles se debe utilizar la imagen grande; y la pequeña cuando es mayor o igual que 465 píxeles.

La sintaxis del atributo `media` es la misma que en las *media queries* de CSS. Los elementos más comunes para utilizar aquí serían `min-width`, `max-width`, `min-height`, `max-height` y `orientation`, este último con los valores `landscape` y `portrait`. Por ejemplo:

```
<picture>
  <source srcset="smaller_landscape.jpg"
    media="(max-width: 40em) and (orientation: landscape)">
  <source srcset="smaller_portrait.jpg"
    media="(max-width: 40em) and (orientation: portrait)">
  <source srcset="default_landscape.jpg"
    media="(min-width: 40em) and (orientation: landscape)">
  <source srcset="default_portrait.jpg"
    media="(min-width: 40em) and (orientation: portrait)">
  <img srcset="default_landscape.jpg" alt="My default image">
</picture>
```

Lo siguiente que tienes que tener en cuenta en el código anterior es el elemento `` que se incluye al final del elemento `<picture>`. Esta es la imagen de *fallback* y es la que se utilizará si el navegador no reconoce la etiqueta `<picture>` o bien si ninguna de las *queries* coincide. Observa también que es aquí donde se debe indicar el atributo `alt` de la imagen.

3.2.2.- DESCRIPTORES DE DENSIDAD DE PÍXEL

Hay ocasiones, en que la resolución del dispositivo es una referencia incompleta para escoger una resolución de imagen adecuada. Esto nos pasa en dispositivos con una densidad muy alta de píxeles, como las pantallas *retina* de Apple. En estos dispositivos, se habla de píxeles CSS que corresponden cada uno con varios píxeles del dispositivo. Esto es válido, para elementos vectoriales pero en imágenes nos interesará descargar una versión en alta resolución de la imagen. En el siguiente código puedes ver un ejemplo de cómo se consigue esto:



```
<source
  media="(min-width: 650px)"
  srcset="resolucion_baja.png,
         resolucion_media.png 1.5x,
         resolucion_alta.png 2x" />
```

En este caso proporcionamos tres versiones de la imagen, la segunda se aplicará en pantalla que tengan una relación de 1,5 entre los píxeles CSS y los píxeles del dispositivo y la tercera para dispositivos una relación de 2.

3.2.3.- EL ATRIBUTO SIZES

El otro atributo, que puede tener la etiqueta `<source>` es `sizes`. Este es útil cuando el tamaño final de la imagen es desconocido, especialmente, para imágenes *responsive* cuyo tamaño varía en función del ancho de la ventana del navegador.

Veamos primero un ejemplo y luego analicemos el código:

```
<picture>
  <source media="(min-width: 800px)"
    sizes="80vw"
    srcset="lighthouse-landscape-640.jpg 640w,
           lighthouse-landscape-1280.jpg 1280w,
           lighthouse-landscape-2560.jpg 2560w">
  
</picture>
```

En el código anterior, cuando le estamos dando el valor 80vw a sizes estamos indicando que la imagen va a ocupar el 80% del ancho de la ventana del navegador. Recuerda que la unidad **vw** hace referencia a una centésima parte del ancho de la ventana de la ventana de visualización (*viewport width*).

A continuación, indicamos dentro de srcset las imágenes con diferentes resoluciones añadiendo el ancho de las mismas. Así en el ejemplo, indicamos que la primera imagen tiene 640 píxeles de ancho, la segunda 1280 y la tercera 2560. El navegador calculará el tamaño que tendrá la imagen y descargará la versión que mejor se adapte a dicho tamaño.




<https://www.html5rocks.com/en/tutorials/responsive/picture-element/>

3.3.- FUENTES DE ICONOS

Una alternativa muy interesante, cuando queremos minimizar el uso de imágenes en nuestra página Web es la utilización de **fuentes de iconos**. Con este término nos referimos a fuentes que solo incluyen caracteres con símbolos.

Las ventajas que proporciona la utilización de estas fuentes respecto a imágenes son muchas:

- Como sabrás **las fuentes son elementos vectoriales**, es decir, se almacenan las curvas que las forman como ecuaciones en lugar de una matriz de puntos como pasa con las imágenes. La ventaja inmediata que tiene esto, es que cuando, se aumenta su tamaño no tienen pérdida de calidad ni se produce un pixelado de los bordes.
- Las fuentes de iconos **ocupan menos espacio que las imágenes**, con el consiguiente ahorro de ancho de banda que implica. Fíjate en la siguiente imagen cómo el fichero de fuentes ocupa una décima parte del tamaño de la imagen en formato PNG.

 victorjgonzalez.com	200	document	Other	965 B	176 ms	<div><div></div></div>
 Twitter_Logo.png	200	png	(index):64	10.7 KB	175 ms	<div><div></div></div>
 icomoon.ttf?3pmvit	200	font	(index):-Infinity	1.9 KB	175 ms	<div><div></div></div>

- En una única fuente se pueden incluir tantos iconos como queramos, teniendo la ventaja inmediata de que se **reducen el número de solicitudes al servidor**.
- Al ser una fuente podemos aplicarle **diferentes colores o incluso sombras**, obteniendo diversas versiones a partir de un único fichero.

4.3.1.- OBTENCIÓN DE UNA FUENTE DE ICONOS

Hay disponibles un gran número de fuentes de iconos, tanto de pago como gratuitas. En *We Love Icon Fonts*³ tienes una amplia selección con todo tipo de fuentes de iconos gratuitas para entornos de desarrollo (si quieres utilizarlas comercialmente debes verificar la licencia de estas).

Aunque las fuentes disponibles en la página, antes indicada son una buena opción, tienen el problema de que debemos utilizar la fuente entera aunque únicamente vayamos a utilizar un icono, perdiendo de esta manera la ventaja antes indicada de un menor tamaño en los ficheros.

³ <http://weloveiconfonts.com/>

Como seguro que ya estarás pensando, una opción es obtener estas fuentes y utilizar servicios como *Font Squirrel* en el cual seguro que recuerdas que podíamos generar una fuente que únicamente contenga un subconjunto de los caracteres de otra fuente.

Por fortuna, disponemos de otro servicio que nos simplifica este trabajo, automatizando el proceso de selección de los iconos y la generación de la fuente final. Este servicio se llama **IcoMoon**⁴.

El proceso de selección de iconos y creación de una fuente es muy sencillo, por lo que lo omitiremos, pasando a ver los pasos a realizar para utilizar la fuente una vez descargada.





⁴ <https://icomoon.io/>

4.- EL ELEMENTO <CANVAS> DE HTML5

4.1.- CONCEPTOS BÁSICOS

El *canvas* es un elemento de HTML5 que permite dibujar gráficos mediante JavaScript. Este elemento permite dibujar gráficos, realizar composición de fotos o incluso realizar animaciones.

Este elemento tiene un amplio soporte por parte de los navegadores, siendo soportado por las versiones actuales de todos ellos.

Element					
<canvas>	4.0	9.0	2.0	3.1	9.0

La etiqueta `<canvas>` se puede ver como equivalente a la etiqueta `` pero con la particularidad de que no dispone de los atributos `src` y `alt`. Es decir, sus únicos atributos son `width` y `height`. Si estos atributos no son especificados se asigna por defecto un ancho de 300 píxeles y un alto de 150 píxeles.

Al igual que nos sucede con otras etiquetas HTML5, como `<video>`, es buena idea proporcionar **contenido de respaldo** (*fallback*) para que se muestre algo en caso de que el navegador no soporte esta etiqueta. Para ello simplemente debemos insertar dicho contenido de respaldo entre las etiquetas de apertura y cierre.

```
<canvas id="clock" width="150" height="150">
  
</canvas>
```

El elemento `<canvas>` crea una superficie de dibujo de tamaño fijo que proporciona uno o más **contextos de renderizado**, los cuales se utilizan para crear o manipular el contenido mostrado. En esta unidad, únicamente veremos los contextos en 2D pero hay otros contextos disponibles, por ejemplo WebGL dispone de un contexto 3D basado en OpenGL ES.

En el siguiente código puedes ver la estructura básica para crear un *canvas*.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Canvas tutorial</title>
    <script type="text/javascript">
      function draw(){
        var canvas = document.getElementById('tutorial');
        if (canvas.getContext){
          var ctx = canvas.getContext('2d');
        }
      }
    </script>
    <style type="text/css">
      canvas { border: 1px solid black; }
    </style>
  </head>
  <body>
    <canvas id="tutorial" width="150" height="150">
      <img alt="Canvas tutorial" />
    </canvas>
  </body>
</html>
```

```
</style>
</head>
<body onload="draw();" >
  <canvas id="tutorial" width="150" height="150"></canvas>
</body>
</html>
```

Como puedes ver estamos utilizando una función `draw()` en JavaScript que es llamada una vez que haya cargado el cuerpo de la página.

En esta función creamos una variable que hace referencia al elemento `<canvas>` de HTML. Sobre esta variable comprobamos que dispone del método `getContext()` y en caso afirmativo le indicamos que queremos un contexto 2D.

4.2.- FORMAS BÁSICAS EN EL CANVAS

Ahora que ya hemos creado el *canvas* es momento de comenzar a dibujar en él. Tienes que saber que el *canvas* únicamente tiene una primitiva predefinida, el rectángulo. El resto de figuras deben ser creadas mediante rutas (*paths*), una lista de puntos conectados por líneas.

5.2.1.- RECTÁNGULOS

Las 3 funciones disponibles en el *canvas* para dibujar triángulos son:

<code>fillRect(x, y, width, height)</code>	Dibuja un triángulo relleno
<code>strokeRect(x, y, width, height)</code>	Dibuja el contorno de un rectángulo
<code>clearRect(x, y, width, height)</code>	Limpia el área rectangular especificada, haciéndola transparente.

Estas funciones toman como parámetro las coordenadas de la esquina superior izquierda del rectángulo y el ancho y alto del mismo. Ten en cuenta que el **origen de coordenadas** del *canvas* se encuentra en la esquina superior izquierda del mismo.

4.2.2.- RUTAS (PATHS)

La otra primitiva de que dispone el *canvas* son las rutas o *paths*. Una ruta es una lista de puntos, conectados por segmentos de línea que pueden ser tanto rectos como curvos, e igualmente pueden tener diferentes colores y grosores. Las rutas pueden ser cerradas para formar una figura geométrica determinada.

Para crear una ruta debemos seguir los siguientes pasos:

1. Primero creamos o inicializamos la ruta
2. A continuación dibujamos los diferentes puntos y segmentos de la ruta.
3. Cerramos la ruta
4. Y finalmente, una vez que la ruta ha sido creada, se puede marcar el contorno o el relleno de la misma para renderizarla.

Las funciones para realizar los pasos antes indicados son las siguientes:

beginPath()	Crea una nueva ruta, las funciones de dibujo que se indiquen a continuación harán referencia a esta ruta.
closePath()	Cierra la ruta
stroke()	Dibuja el contorno de la ruta
fill()	Rellena el área contenida dentro de la ruta

Desde que abrimos hasta que cerramos la ruta podemos llamar a la funciones para dibujar la misma. Algunas de estas funciones son:

moveTo(x, y)	Desplaza el puntero de la ruta al punto (x, y) sin dibujar.
lineTo(x, y)	Traza una línea entre el punto actual y el punto (x, y)
bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)	Crea una curva de Bezier entre el punto actual y el punto (x, y) utilizando como puntos de control los puntos (cp1x, cp1y) y (cp2x, cp2y)
arc(x, y, radius, startAngle, EndAngle, anticlockwise)	Traza una arco centrado en (x, y) con el radio indicado y entre los ángulos que se pasan como parámetro.

Por ejemplo en el siguiente código podemos ver como dibujar un triángulo.

```
ctx.beginPath();  
ctx.moveTo(75, 50);  
ctx.lineTo(100, 75);  
ctx.lineTo(100, 25);  
ctx.fill();
```



Observa como comenzamos desplazando el puntero de dibujo a la esquina izquierda del triángulo (coordenadas (75, 50)) y una vez ahí dibujamos una línea a la esquina inferior y otra a la esquina superior.

También fíjate que no es necesario cerrar la figura sino que al utilizar la función **fill()** automáticamente se completa la figura uniendo el último punto con el punto de inicio de la misma.

4.2.3.- APLICANDO COLORES Y ESTILOS

Hasta hemos visto como dibujar figuras pero siempre en color negro. El *canvas* permite modificar el color, tanto de lo contornos como de los rellenos. Para ello utilizamos las propiedades **fillStyle** y **strokeStyle** del contexto.

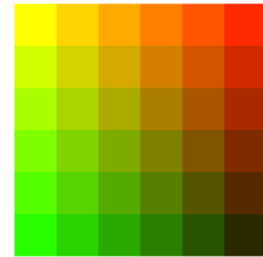
A estas propiedades se le asigna un color que se aplicará a todos los elementos que se dibujen hasta que se les asigne un color diferente.

Las siguientes líneas son todas equivalente para asignar un color naranja al contorno.

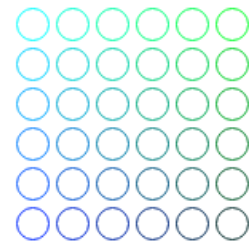
```
ctx.fillStyle = 'orange';
ctx.fillStyle = '#FFA500';
ctx.fillStyle = 'rgb(255, 165, 0)';
ctx.fillStyle = 'rgba(255, 165, 0, 1)';
```

Veamos un par de ejemplos de aplicación de estas propiedades:

```
for (var i = 0; i < 6; i++) {
  for (var j = 0; j < 6; j++) {
    ctx.fillStyle = 'rgb('
      + Math.floor(255 - 42.5 * i)
      + ', '
      + Math.floor(255 - 42.5 * j) + ', 0)';
    ctx.fillRect(j * 25, i * 25, 25, 25);
  }
}
```



```
for (var i = 0; i < 6; i++) {
  for (var j = 0; j < 6; j++) {
    ctx.strokeStyle = 'rgb(0, '
      + Math.floor(255 - 42.5 * i)
      + ', '
      + Math.floor(255 - 42.5 * j) + ')';
    ctx.beginPath();
    ctx.arc(12.5 + j * 25, 12.5 + i * 25,
      10, 0, Math.PI * 2, true);
    ctx.stroke();
  }
}
```



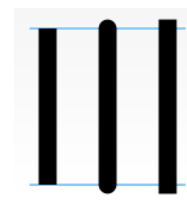
En el caso de los contornos también podemos controlar las características de la línea mediante las siguientes propiedades y funciones:

lineWidth = value

Establece el ancho de las líneas

lineCap = type

Establece la apariencia de los finales de línea. Los valores posibles pueden ser **butt**, **round** o **square** y son los que se ven en la imagen de la derecha respectivamente.



`lineJoin = type`

Establece la apariencia de las esquinas donde se juntan dos líneas. Los valores pueden ser `round`, `bevel` o `miter`.



`setLineDash(segments)`

Establece una línea discontinua en el borde. Se le pasa una array de números que indican la longitud de las líneas y huecos del contorno.



`getLineDash()`

`lineDashOffset = value`

4.2.3.- GRADIENTES

Si no queremos rellenar una figura con un color plano tenemos la posibilidad de utilizar gradientes. Para trabajar en el *canvas* con gradientes tenemos que crear un objeto de tipo `CanvasGradient` que contendrá el gradiente, y será este objeto el que se aplique mediante las propiedades `fillStyle` o `strokeStyle`.

Para trabajar con gradientes disponemos de las siguientes funciones:

`createLinearGradient (x1, y1, x2, y2)`

Crea un gradiente lineal desde las coordenadas (x1, y1) hasta (x2, y2)

`createRadialGradient (x1, y1, r1, x2, y2, r2)`

Crea un gradiente radial

`addColorStop(position, color)`

Crea una parada de color en el gradiente. La posición es un valor entre 0 y 1.

4.3.- DIBUJANDO TEXTO

El *canvas* de HTML5 dispone de dos funciones para dibujar texto, así como una serie de propiedades.

`fillText(texto, x, y [, maxWidth])`

Rellena el texto indicado en la posición (x, y). Opcionalmente se puede indicar un ancho máximo.

`strokeText(texto, x, y [, maxWidth])`

Esta función dibuja únicamente el contorno del texto.

font = value

Indica el estilo con que debe ser dibujado el texto. Utiliza la misma sintaxis que la propiedad **font** de CSS.

textAlign = value

Indica cómo se va a alinear el texto. Los posibles valores son: **start**, **end**, **left**, **right** o **center**.

Si queremos tener un mayor control del texto antes de dibujarlo disponemos de la función **measureText()** que nos devuelve las métricas del texto, siendo la más interesante la que nos da el ancho que va a ocupar una vez sea impreso, como podemos ver en el siguiente código.

```
function draw() {  
  var ctx = document.getElementById('canvas').getContext('2d');  
  var text = ctx.measureText('foo'); // TextMetrics object  
  text.width; // 16;  
}
```