



estudios abiertos

SEAS

GRUPO SANVALERO

5
UNIDAD
DIDÁCTICA

Lenguaje JavaScript

5. HTML5 y CSS3

ÍNDICE

OBJETIVOS	359
INTRODUCCIÓN	360
5.1. Introducción a HTML5	361
5.1.1. Qué es HTML5	361
5.1.2. Recursos necesarios para el desarrollo	362
5.2. Estructura de HTML5	367
5.3. Etiquetas de HTML5	371
5.3.1. Vídeo	371
5.3.2. Audio	374
5.3.3. Canvas	377
5.3.4. Pintar imágenes	397
5.4. Introducción a CSS3	423
5.5. Prefijos del navegador	424
5.6. Etiquetas CSS3	425
5.6.1. Bordes	425
5.6.2. Sombras	432
5.6.3. Transformaciones 2D	437
5.6.4. Opacidad (opacity)	448
5.6.5. Opacidad (color)	449
5.6.6. Múltiples columnas	451
5.6.7. Importar fuentes	457
5.6.8. Tratamiento de Imágenes	460
5.6.9. Transiciones	465
5.6.10. Animaciones	471
5.6.11. Sintaxis de una animación	472
5.6.12. Animaciones 3D	484
RESUMEN	493

OBJETIVOS

- En esta unidad tendremos una visión general de HTML5 y CSS3. Veremos como implementar las diferentes etiquetas existentes, para dar a nuestras páginas un aspecto más profesional.
- Comenzaremos introduciéndonos en HTML5 y CSS3. Para eso necesitaremos conocimientos previos de ambas notaciones, por lo que si no estáis familiarizados con estos, deberéis mirar los anexos de HTML y CSS que se proporcionan.
- Descubriremos las etiquetas más utilizadas de HTML5, y la forma de introducirlas. Descubriremos como introducir audio, vídeo y como utilizar el elemento canvas.
- Nos introduciremos en CSS3 con los prefijos necesarios para usar las etiquetas en los distintos navegadores.
- También descubriremos las etiquetas más importantes de CSS3. Con ellas podremos:
 - Modificar los bordes.
 - Aplicar sombras.
 - Realizar transformaciones en 2D.
 - Cambiar la opacidad de los elementos.
 - Insertar múltiples columnas de texto.
 - Importar fuentes externas al navegador.
 - Usar imágenes de una forma sencilla.
 - Realizar transiciones.
 - Realizar animaciones en 2D y 3D.

INTRODUCCIÓN



Vamos a descubrir en esta unidad los nuevos estándares de desarrollo web. Comenzaremos descubriendo la estructura del estándar para pasar a ver las diferentes etiquetas que son soportadas en esta versión.

Después nos introduciremos en el lenguaje de CSS3, descubriendo primeramente los prefijos utilizados por cada navegador para pasar a ver todas las etiquetas disponibles en esta nueva versión de CSS.

Con HTML 5 y CSS 3 vamos a poder dar muchas más funcionalidades a nuestras páginas web, dotándolas de efectos y animaciones de una forma sencilla, siendo soportada por la mayoría de dispositivos móviles, en estos momentos que están dejando de dar soporte a Flash.

5.1. Introducción a HTML5

Antes de empezar con la sintaxis de HTML 5, debemos situarnos y ver que herramientas vamos a utilizar, los conocimientos previos que debéis de tener antes de continuar y los recursos necesarios de los que disponemos para su realización.

Si recordáis las primeras páginas web que había al principio, estaban formadas únicamente por imágenes y por texto. Eran los principales recursos para realizar una página web.

Posteriormente, empezó a aparecer el vídeo en las páginas webs, pero con el problema de que no había un estándar, ya que existían un sinfín de códec para su reproducción.



Códec es la abreviatura de codificador-decodificador. Los códec codifican el flujo o la señal de un archivo y lo recuperan o descifran del mismo modo para su reproducción en un formato más apropiado para operaciones como la reproducción del mismo.

El problema era que el usuario debía tenerlos instalados en su ordenador para que cuando una página necesitase reproducirlo, poder hacer uso de ellos. Cuando aparece YouTube, los desarrolladores vieron la luz al descubrir una plataforma capaz de contener los vídeos, sin necesidad de realizar grandes programaciones.

Cual fue el problema que surgió con esto, que dichos vídeos se realizaban y se necesitaba tener Flash Player para poder reproducirlo. Esto hizo que Adobe tuviese el mando sobre el vídeo en el mundo web, y de todos es conocido los problemas de seguridad que Flash tiene. Por lo tanto, debemos evitar insertar contenido para reproducir en Flash Player en nuestras páginas web.

También es verdad, que si dejamos de insertar contenido flash en nuestro sitio web, vamos a perder muchas funcionalidades como son las animaciones y los vídeos.

Por lo tanto, debemos pensar en una opción que nos supla estas carencias, y esta tecnología será HTML5.

De esta manera usaremos una nueva forma de construir la web, que va a ser universal, con la que no vamos a perder funcionalidades y en la que iremos añadiendo otras nuevas.

5.1.1. Qué es HTML5

Es un ecosistema de tecnologías. No es solo HTML en sí mismo, sino que está compuesto de tres tecnologías:

- **HTML5:** la utilizaremos para la semántica y la construcción del contenido en sí de la página.

- **CSS3:** que lo utilizaremos para el diseño y la estética de la página.
- **JavaScript:** dará funcionalidad a nuestro sitio web.

HTML5 nos va a permitir insertar, mediante sus nuevas etiquetas, de una forma nativa, tanto los gráficos vectoriales, como el audio y el vídeo, y de una forma estandarizada.

CSS3 se va a actualizar para poder recuperar esa estética que podíamos crear con Flash, que con versiones anteriores no podíamos realizar, como puede ser bordes redondeados en los divs, crear transparencias, gradientes de color, animaciones de transición que podremos utilizar.

Y mediante JavaScript podremos aplicar las funcionalidades a estas dos tecnologías en nuestras aplicaciones web.

5.1.2. Recursos necesarios para el desarrollo

En este punto vamos a ver que recursos gratuitos podemos usar para el desarrollo e implementación de HTML5 y CSS3.

Podemos utilizar múltiples editores de código HTML, todos ellos gratuitos, pero en esta unidad trabajaremos con Sublime Text2 que os lo podéis descargar de la siguiente dirección: <http://www.sublimetext.com/2>.

Cuando accedáis a la página, tendréis varias versiones para descargar. Debéis seleccionar la que mejor se adapte a vuestro sistema operativo.

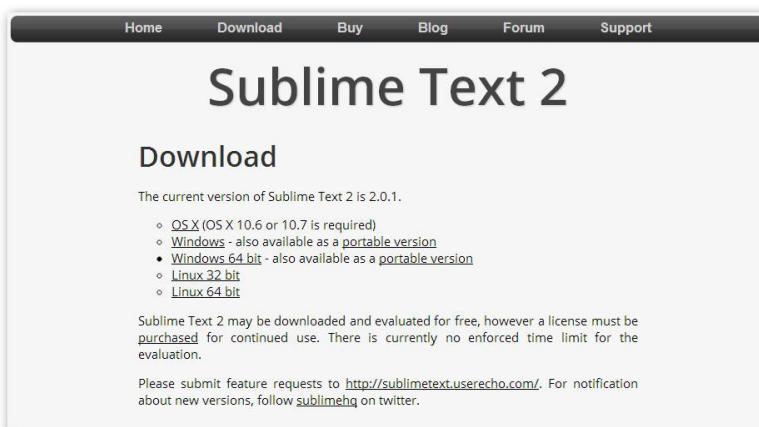


Figura 5.132. Página de descarga del programa Sublime Text 2.

Una vez descargado, procedemos a su instalación. En la primera pantalla le damos a “Next”:



Figura 5.133. Pantalla inicial de instalación.

En la siguiente pantalla, nos permite cambiar el directorio de instalación. Por defecto dejamos el que nos sugiere:

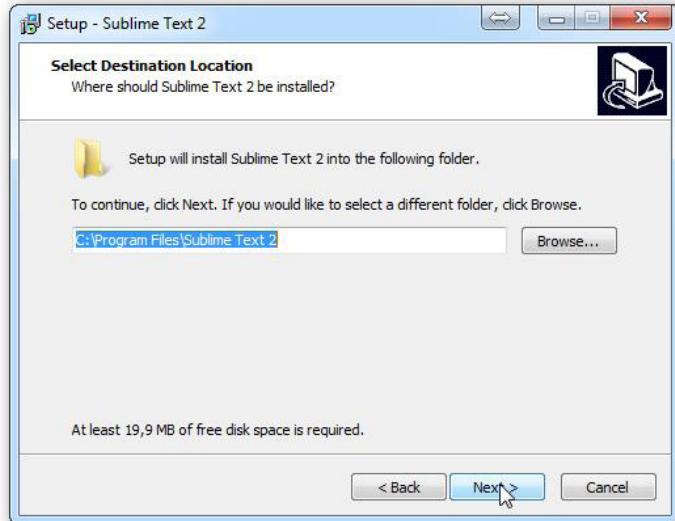


Figura 5.134. Pantalla de selección del directorio de instalación.

En la siguiente pantalla seleccionamos la casilla, para que se nos añada al menú contextual del navegador y pulsamos en “Next”:

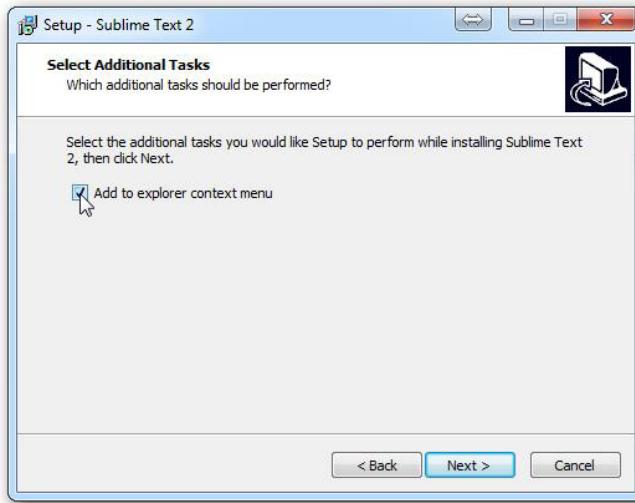


Figura 5.135. Marcamos la opción de añadir al menú contextual.

Y por último pulsamos en “Install”:

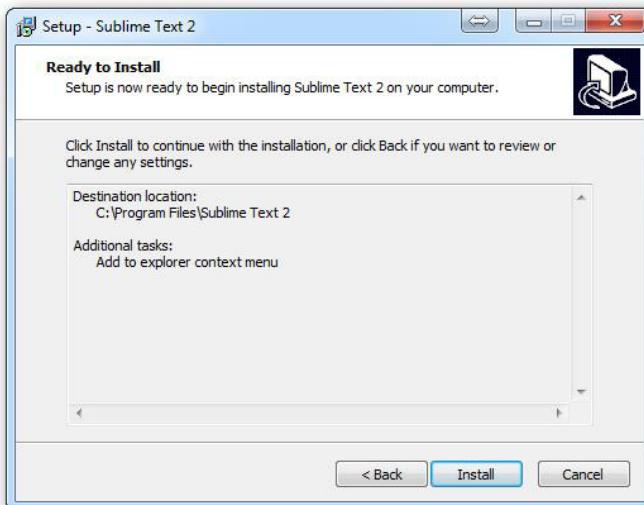


Figura 5.136. Pantalla para proceder a la instalación.

Cuando ya se ha instalado el programa, podemos pulsar en “Finish”, para finalizar la instalación del programa:



Figura 5.137. Pantalla de finalización de la instalación.

Una vez que ha acabado, podemos abrir el programa, accediendo al menú inicio y buscándolo en programas.

Lo interesante de este programa es la opción que nos da de poder predefinir el lenguaje que vamos a usar en nuestros ejemplos, pudiendo seleccionar dicho lenguaje, para que nos sea más fácil realizarlo.

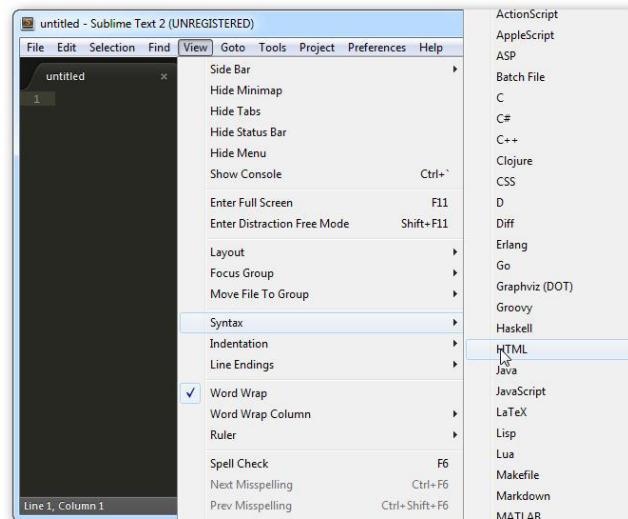


Figura 5.138. Vista del menú de selección del lenguaje a utilizar.

Una vez seleccionado el lenguaje, el programa nos ayudará a interpretar la inserción del código.

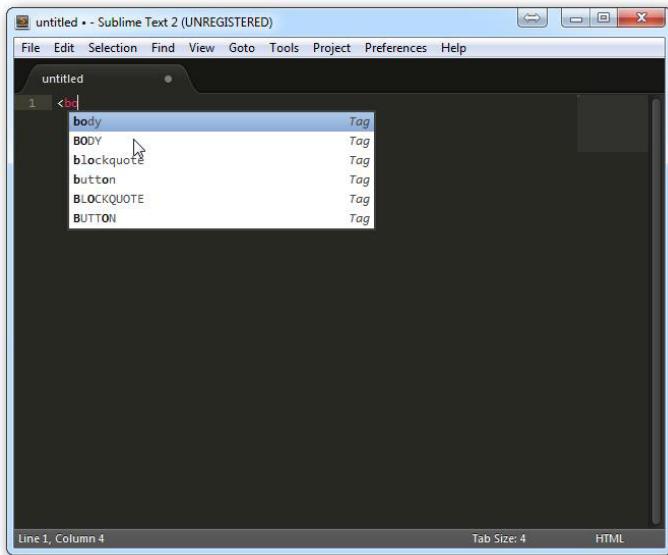


Figura 5.139. Vista de las ayudas mostradas por el programa.

Coloreará las etiquetas y nos ayudará en la inserción de las mismas.

5.2. Estructura de HTML5

Tenemos que tener claro que HTML5 se construye de la misma manera que su antecesor, a través de etiquetas. Recordad que toda etiqueta siempre va a tener la etiqueta de apertura y de cierre.

Las páginas HTML van a tener la misma estructura, comenzando siempre por la definición del DOCTYPE. Esta sección se ubica en la primera línea del archivo HTML, es decir, antes de la marca html

Según el rigor de HTML 4.01 utilizado la podíamos declarar como:

- Declaración transitoria:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- Declaración estricta:

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

A partir de este momento, con el desarrollo de HTML 5 se simplifica esta sección con la siguiente sintaxis:

```
<!DOCTYPE html>
```

De esta forma el navegador puede saber que estamos utilizando la especificación del HTML 5.

Otra cosa que debemos definir en nuestras páginas es el lenguaje, y lo haremos de la siguiente forma:

```
<html lang="es">
```

El resto del documento será siempre igual en todos. Lo dividiremos en:

```
<head>
    Contendrá los scripts, los metadatos y otros datos no semánticos
</head>
<body>
    Contenido de la página, texto, imágenes...
</body>
```

Las qué ventajas obtenemos con las estructuras de HTML5 en nuestras páginas web, sobre todo es el poder conseguir una mayor optimización SEO o posicionamiento de las mismas, una sintaxis clara y ordenada y un código mejor y más estandarizado.

A continuación vamos a ver como organizar el código dentro del body. Debemos seguir la siguiente estructura:

```
<body>
    <header>
    </header>
    <nav>
        <a href="inicio.html">Inicio</a>
        <a href="nosotros.html">Nosotros</a>
        <a href="opcion1.html">Opcion1</a>
        <a href="opcion2.html">Opcion2</a>
    </nav>
    <aside>
        <p>el contenido de aside es algo que no tiene gran relevancia en la página, como un submenú, buscador, formulario de contacto etc...</p>
    </aside>
    <section>
        <p>un section es una manera de dividir la página por "secciones" de tal modo que podemos dividir el contenido de esta por temas</p>
        <article>
            <h1>El article es la parte más importante del sitio</h1>
            <p>Básicamente en el article pondremos lo más importante de la página, la información jerárquicamente más importante, un article puede tener header y footer, sin estropear el SEO.</p>
        </article>
    </section>
    <footer>
        <p>Derechos reservados</p>
        <p>Contáctenos en seas@seas.es</p>
    </footer>
</body>
```

Esta debe ser la semántica de nuestra página.



La semántica no es más que el significado que les damos a las palabras.

En la semántica de HTML, el lenguaje está hecho para que cada “etiqueta HTML” signifique algo. De esta manera el explorador sabe qué pasa cuando encuentra un ``, sabe que es una imagen, o si se encuentra un `<p>` ya sabe que es un párrafo etc. Con los buscadores sucede lo mismo, por eso para un buscador es más importante un `<h1>` que un `<p>`.

Definamos cada etiqueta de nuestra estructura anterior:

- <header> es el que delimita el encabezado visible del sitio. Dentro de esta etiqueta generalmente encontraremos el menú, logotipo y encabezados del sitio.
- <nav> dentro de esta etiqueta pondremos siempre los enlaces mas importantes del sitio.
- <footer> dentro de footer generalmente se ponen cosas como otros enlaces, los derechos de autor, etc., pero no es exclusivo, podemos usarlos para desplegar lo que necesitemos, siempre y cuando esté en el “pie” de la página.
- <aside> dentro del aside va el contenido que no tiene gran relevancia en la página, como un submenú, un buscador, un formulario de contacto, etc.
- <section> un section es la manera de dividir la página por “secciones” de tal forma, que podemos dividir el contenido de esta por temas. Dentro de esta etiqueta van los <articles>
- <article> el article es la parte más importante del sitio. Pondremos lo más importante de la página, es decir, la información más relevante jerárquicamente hablando. Debéis saber que un article puede tener header y footer, sin estropear el SEO.

El Header y footer no siempre tiene que ir en el encabezado y pie de página de una página. Estos pueden repetirse y ser contenidos en otras etiquetas, pero está claro, que la etiqueta en una posición con mayor valor jerárquico en el documento tendrá más peso que las otras.

Visto en una forma gráfica, la estructura quedaría de la siguiente forma:

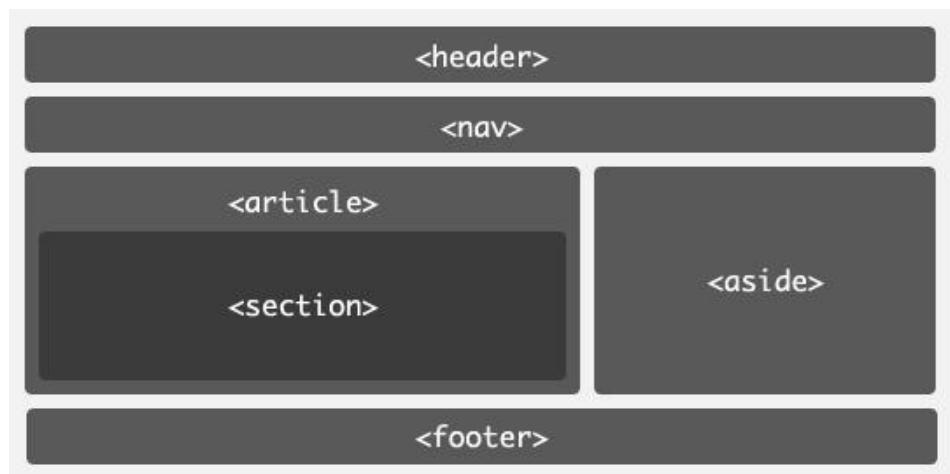


Figura 5.140. Estructura HTML5 completa.

Existe una etiqueta más, <hgroup> que se usa para agrupar varias etiquetas de cabeceras (<h1> <h2> <h3> <h4> <h5> <h6>) cuando se van a utilizar títulos, subtítulos, de esta forma, evitaremos cortar el flujo del esquema del documento:

```
<hgroup>
    <h1>Título de la página</h1>
    <h2>Subtítulo de la página</h2>
</hgroup>
```



Si os habéis fijado, no hemos dicho nada de la etiqueta <div>. No es que haya desaparecido por completo, pero su utilización se restringirá a una sección con propósitos netamente de posicionamiento, o de aplicación de estilos, etc.

Son los únicos casos en los que es recomendable seguir usando la etiqueta <div> en vez de usar las etiquetas mencionadas anteriormente.

Las etiquetas siguientes, podemos seguir usándolas con normalidad, pero con ciertas restricciones:

```
<h1>Título de la página</h1>
<h2>Subtítulo de la página</h2>
```

Siempre que usemos la etiqueta <h1>, que sea estrictamente justificado su uso, ya que penalizará el uso de varias etiquetas de este tipo en una misma página web.

La etiqueta <p> párrafo, podemos usarla de forma normal para insertar un conjunto de texto.

En los siguientes puntos veremos todos los nuevos elementos posibles que se pueden realizar con HTML5.

5.3. Etiquetas de HTML5

En este punto mostraremos las diferentes etiquetas que se han introducido en HTML5. Para ello, vamos a crear un sencillo ejemplo, con el que poder probar sus funcionalidades.

A partir de este momento, usaremos siempre como punto de partida el siguiente código de página web con las etiquetas necesarias de la estructura de HTML5:



```

EJEMPLO

<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>
    <header>
        </header>
    <nav>
        </nav>
    <aside>
        </aside>
    <section>
        <article>
            </article>
        </section>
        <footer>
            </footer>
    </body>
</html>

```

5.3.1. Vídeo

Esta etiqueta nos va a permitir mostrar un vídeo sin la necesidad de plugin (Flash). En estos momentos, los navegadores permiten mostrar una cantidad muy limitada de formatos de vídeo.

Para asegurarnos el correcto funcionamiento en todos los navegadores, debemos usar dos o tres códec de vídeo y ese dicho vídeo en dicho formato, ya que el futuro del vídeo en la red pasa por solo tres: H.264, Theora, y VP8. Dichos códec son:

- **MPEG-4 (.mp4):** este formato se basa en el QuickTime (.mov) de Apple.
- **Ogg (.ogv):** es un estándar abierto, por lo tanto no sujeto a ninguna patente conocida.
- **WebM es un formato nuevo.** Técnicamente es similar a Matroska. Está diseñado para ser utilizado exclusivamente con el códec de vídeo VP8.

Veamos ahora cada uno de ellos con sus navegadores soportados y ventajas de cada uno de ellos:

■ Códice H.264

- **Navegadores soportados:** Safari, Google Chrome y el futuro Internet Explorer 9.
- **Compatibilidad:** es algo generalizado, se utiliza tanto en el vídeo para la web como para fuera de la red.
- **Ventajas:** está muy consolidado. Tiene una calidad aceptable con ficheros pequeños, además tiene múltiples usos.
- **Inconvenientes:** es propietario.

■ Códice OGG Theora

- **Navegadores soportados:** Firefox, Opera y Google Chrome soportan de fábrica este códec.
- **Compatibilidad:** los dispositivos digitales que permiten crear contenidos en Theora son prácticamente nulos.
- **Ventajas:** es libre.
- **Inconvenientes:** necesita un 25% más recursos que H.264, además ofrece una calidad menor y los ficheros generados son mucho más grandes.

■ Códice WebM

- **Navegadores soportados:** Google Chrome, Firefox y Opera soportarán de fábrica. Internet Explorer 9 añadirá soporte, pero solo si el usuario tiene instalado ese códec. Y como Safari utiliza Quicktime como base, podrá soportar también este códec.
- **Ventajas:** es libre y de una calidad similar a H.264, además sus ficheros que pesan la mitad.
- **Inconvenientes:** las cámaras actuales no soportan este formato, por lo que tendremos que convertir los vídeos que grabemos.

Veamos la siguiente tabla con las compatibilidades de los formatos de vídeo y los navegadores que lo soportan:

Navegador/Códec	H.264+ AAC+ MP4	WebM	Theora +Vorbis +Ogg
Mozilla Firefox 3.6		✓	✓
Opera 10.63		✓	✓
Google Chrome 8.0	✓	✓	✓
Apple Safari 5.0.3 (with QuickTime)	✓		
Microsoft IE 9 Beta	✓	✓	

Figura 5.141. Tabla de compatibilidades entre navegadores y códec.

Una vez que sabemos los códec que se pueden implementar, con los dos primeros nos aseguramos su funcionamiento en los navegadores más importantes que actualmente existen.

Para insertarlo en la página nuestra debemos hacerlo de la siguiente manera:

```
<video width="640" height="360" controls poster="logotipo.png" >
<source src="vídeo.mp4" type="video/mp4; codecs='avc1, mp4a'" />
<source src="vídeo.ogg" type="video/ogg; codecs='theora, vorbis'" />
<source src="vídeo.webm" type="video/webm; codecs='vp8, vorbis'" />
Este navegador no permite tag video
</video>
```

En este caso, en la propiedad src de la etiqueta source insertamos la dirección donde se encuentra nuestro vídeo, y en la propiedad type definimos el tipo de formato del vídeo y el códec necesario para su reproducción.

Si el navegador no soporta ninguno de los formatos de vídeo, se mostrará la etiqueta siguiente:

Este navegador no permite tag video

En la etiqueta video podemos definir los siguientes atributos:

- **src:** dirección donde se almacena el vídeo.
- **controls:** se visualiza el panel de control del vídeo: botón de inicio, barra de avance del vídeo etc. Si está presente se muestran los controles.
- **autoplay:** el vídeo se inicia inmediatamente después de que la página se carga en el navegador. Si está presente se empieza a reproducir automáticamente.
- **width:** ancho en píxeles del vídeo.
- **height:** alto en píxeles del vídeo.
- **poster:** contiene la dirección de la imagen previa que se va a cargar cuando se carga la página. Si no se incluye, se muestra el primer fotograma del vídeo.

Si queremos probar los diferentes navegadores con los diferentes tipos de formatos

de vídeo, podéis descargaros conversores gratuitos para convertir de un formato a otro. De esta forma os aseguráis que con cualquier navegador que utilice el usuario, el vídeo se reproduzca perfectamente.

El código completo de la página para que reproduzca nuestro vídeo sería el siguiente:

EJEMPLO

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Vídeo</title>
</head>
<body>
    <header>
        <h1>Vídeos</h1>
    </header>
    <nav>
        <ul>
            <li>Vídeo</li>
            <li>Otro</li>
            <li>Otro</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <video width="640" height="360" controls
                poster="imagen.png" >
                <source src="vídeo.mp4" type="video/mp4;
                    codecs' avc1,mp4a'" />
                <source src="vídeo.ogg" type="video/ogg;
                    codecs='theora, vorbis'" />
                <source src="vídeo.webm" type="video/
                    webm; codecs='vp8, vorbis'" />
                Este navegador no permite tag video
            </video>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>
```

5.3.2. Audio

Con esta etiqueta podremos cargar y ejecutar archivos de audio sin requerir un plugin de Flash, Silverlight o Java. Tiene una forma similar de funcionamiento a la etiqueta de Vídeo.

El comité de estandarización W3C ha dejado abierto a cada navegador los formatos que quieran soportar (algunos soportan mp3, wav, ogg, au).

Como no hay un formato de audio universalmente adoptado por todos los navegadores, el elemento audio nos permite agregarle distintas fuentes. Veamos un ejemplo donde lo vamos a realizar de diferentes formas:

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

Ya veis, que la forma es muy similar a la etiqueta vídeo. Hay que definir la localización del archivo mediante el atributo src, y el tipo de sonido mediante el atributo type. Tal y como lo hemos hecho, el propio navegador será el que reproduzca el formato que soporte.

Para comprobar cuál es el tipo soportado en cada navegador, podríamos hacerlo de la siguiente forma:

```
<article>
  <audio src="sounds/gallo.ogg" controls type="audio/ogg" title="OGG">
  </audio>
  <br />
  <audio src="sounds/gallo.mp3" controls type="audio/mpeg" title="MP3">
  </audio>
  <br />
  <audio src="sounds/gallo.wav" controls type="audio/wav" title="WAV">
  </audio>
  <br />
</article>
```

De esta manera, nos saldrán 3 reproductores, y comprobaremos cuál de los formatos soporta cada navegador.

Veamos la siguiente tabla con las compatibilidades de los formatos de audio y los navegadores que lo soportan:

Navegador	.mp3	.wav	.ogg
Mozilla Firefox 3.6		✓	✓
Opera 10.63		✓	✓
Google Chrome 8.0	✓	✓	✓
Apple Safari 5.0.3 (with QuickTime)	✓	✓	
Microsoft IE 9	✓	✓	

Las propiedades que podemos utilizar con la marca audio son:

- **src:** la URL donde se almacena el archivo de audio. Si no definimos la URL, la busca en el mismo directorio donde se almacena la página.

- **autoplay:** en caso de estar presente el archivo se ejecuta automáticamente después de cargarse la página sin requerir la intervención del visitante.
- **loop:** el archivo de audio se ejecuta una y otra vez.
- **controls:** indica que se deben mostrar la interface visual del control en la página (este control permite al visitante arrancar el audio, detenerlo, desplazarse, etc.).
- **autobuffer:** en caso de estar presente indica que primero debe descargarse el archivo en el cliente antes de comenzar a ejecutarse.
- **title:** definiremos el texto que queremos que se muestre mediante un tooltip que aparecerá cuando nos situemos sobre el control.

El código completo de la página para que reproduzca nuestro audio sería el siguiente:

```
<!DOCTYPE html />

<html lang="es">
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>
    <header>
        <h1>Audio</h1>
    </header>
    <nav>
        <ul>
            <li>Otro</li>
            <li>Audio</li>
            <li>Otro</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <audio src="sounds/gallo.ogg" controls type="audio/
ogg" title="OGG">
            <br />
            <audio src="sounds/gallo.mp3" controls type="audio/
mpeg" title="MP3">
            <br />
            <audio src="sounds/gallo.wav" controls type="audio/
wav" title="WAV">
            <br />
        </article>
    </section>
    <section>
        <article>
            <audio controls>
                <source src="sounds/gallo.ogg" type="audio/ogg">
                <source src="sounds/gallo.mp3" type="audio/mpeg">
                <source src="sounds/gallo.wav" type="audio/wav">
            </audio>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>
```

5.3.3. Canvas

El elemento canvas nos va a permitir especificar un área de la página donde vamos a poder, a través de scripts, dibujar y renderizar imágenes, con lo que ampliaremos las posibilidades de las páginas dinámicas y podremos realizar cosas que hasta ahora solo podían hacer los desarrolladores en Flash, pero con una ventaja sobre estos, que para usar canvas no será necesario añadir ningún plugin en el navegador, por lo que mejorará la disponibilidad de esta nueva aplicación.

Los inicios del canvas fueron desarrollados inicialmente por Apple para su navegador Safari y posteriormente fue utilizado y estandarizado por la organización WHATWG para incorporarlo a HTML 5. Firefox y Opera también han adoptado este elemento.

Chrome, como usa el mismo motor de renderizado que Safari, también soporta el elemento Canvas.

Hasta Internet Explorer 9, todas las versiones anteriores no soportaban canvas con funciones nativas, pero existen proyectos y plugins que amplían las funcionalidades del navegador dando soporte a este nuevo elemento de HTML 5.

La forma de insertar este elemento en la página será de la siguiente manera:

```
<canvas id="lienzo" width="50" height="50">
    Su navegador no permite utilizar canvas.
</canvas>
```

Definimos la etiqueta Canvas asignándole los atributos width y height, con los que especificamos su alto y ancho, y mediante el id, que será sobre el que ejecutaremos las funciones de dibujar.

Para usar este nuevo elemento de HTML tenemos que manejar JavaScript. Como muchas de las sentencias JavaScript no son compatibles con todos los navegadores, vamos a hacer una función que realizará unas comprobaciones necesarias para no hacer nada en el caso que el navegador no sea compatible con canvas. La estructura de un programa que accede al canvas será la siguiente:

```
<script type="text/javascript">
    function devolverLienzo(x) {
        //Recibimos el elemento canvas
        var canvas = document.getElementById(x);
        //Comprobación sobre si encontramos un elemento
        //y podemos extraer su contexto con getContext(),
        //que indica compatibilidad con canvas
        if (canvas && canvas.getContext) {
            //Accedo al contexto de '2d' del canvas, necesario para dibujar
            var lienzo = canvas.getContext("2d");
            //devolvemos el lienzo para después realizar el dibujo
            return lienzo;
        }
        else
            return false;
    }
</script>
```

La función devolverLienzo() va a recibir el “id” que hemos especificado en el elemento canvas dispuesto en la página:

```
<canvas id="lienzo1" width="300" height="200">
Su navegador no permite utilizar canvas.
</canvas>
```

Si el navegador que está usando el usuario, no implementa canvas, el if devolverá false, ya que este if comprueba si existe el elemento y se puede acceder a su contexto getContext. Si devuelve true, significa que el navegador soporta canvas:

```
if (canvas && canvas.getContext) {
```

En caso de implementar el canvas, obtendremos una referencia del mismo llamando a la función getContext y pasándole como parámetro un String con el valor “2d”:

```
var lienzo = canvas.getContext("2d");
```

Veamos unos cuantos ejemplos en los que vamos a dibujar diferentes objetos canvas.

5.3.3.1. Dibujar líneas

Comenzaremos por lo más sencillo. Para dibujar líneas disponemos de una serie de métodos que debemos llamar en un orden predeterminado.

Veamos el algoritmo para dibujar dos líneas que formen una letra “V”:

```
<script type="text/javascript">
function devolverLienzo(x) {
    //Recibimos el elemento canvas
    var canvas = document.getElementById(x);
    //Comprobación sobre si encontramos un elemento
    //y podemos extraer su contexto con getContext(),
    //que indica compatibilidad con canvas
    if (canvas && canvas.getContext) {
        //Accedo al contexto de '2d' de este canvas, necesario
        //para dibujar
        var lienzo = canvas.getContext("2d");
        //devolvemos el lienzo para después realizar el dibujo
        return lienzo;
    }
    else
        return false;
}
window.onload = function dibujar() {
    var lienzo = devolverLienzo("lienzo1");
    if (lienzo) {
        lienzo.strokeStyle = "rgb(200,0,0)";
        //Inicio de camino
        lienzo.beginPath();
        lienzo.moveTo(0, 0);
        lienzo.lineTo(150, 300);
        lienzo.lineTo(300, 0);
        //Trazar linea
        lienzo.stroke();
    }
}
</script>
```

A partir de este punto no volveremos a explicar la función devolverLienzo(), ya que siempre realiza las mismas operaciones. La función dibujar() se ejecutará en cuanto la página termine de cargarse.

Lo primero que haremos será inicializar la propiedad que fija el color de la línea:

```
lienzo.strokeStyle = "rgb(200,0,0)";
```

Después llamaremos al método beginPath para indicarle que debe comenzar a dibujar la/s línea/s:

```
lienzo.beginPath();
```

Vamos a mover el puntero gráfico a la coordenada donde queramos que empiece a dibujar, llamando al método moveTo:

```
lienzo.moveTo(0, 0);
```

Trazaremos una línea desde donde se encuentra el puntero gráfico hasta la coordenada indicada mediante dos parámetros:

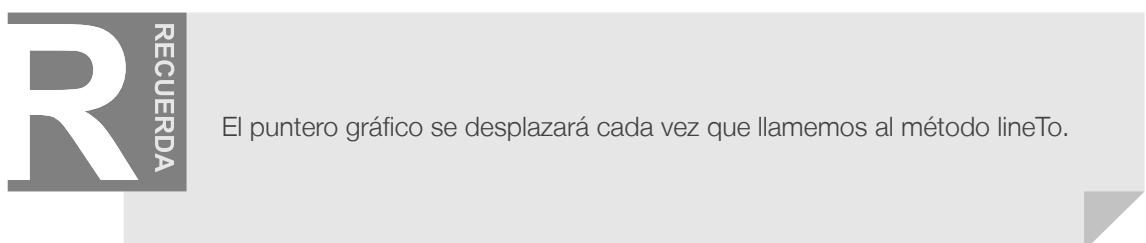
```
lienzo.lineTo(150, 300);
```

Trazamos una segunda línea desde el punto en el que finalizó la anterior hasta la nueva coordenada:

```
lienzo.lineTo(300, 0);
```

Para que todas las líneas trazadas hasta el momento, se hagan efectivas, llamaremos al método stroke:

```
lienzo.stroke();
```



Únicamente debemos implementar la etiqueta canvas en nuestra página, en nuestro caso lo haremos dentro de section y article:

```
<body>
  <header>
    <h1>CANVAS</h1>
  </header>
  <nav>
    <ul>
      <li>Dibujar v</li>
    </ul>
  </nav>
  <aside>
```

```

</aside>
<section>
    <article>
        <canvas id="lienzo1" width="350" height="350">
            Su navegador no permite utilizar canvas.
        </canvas>
    </article>
</section>
<footer>
</footer>
</body>

```

5.3.3.2. Dibujar rectángulo

Para dibujar rectángulos disponemos del método strokeRect. A este método debemos pasarle cuatro parámetros:

- Los dos primeros indican la columna y fila inicial del rectángulo.
- Los otros dos representan el ancho y alto en píxeles.

Para definir el color del borde del rectángulo, utilizamos la propiedad strokeStyle del objeto del contexto del canvas, a la que asignaremos el valor RGB que deseemos para el borde de los cuadrados o aquello que vayamos a dibujar en el canvas.

Veámoslo con un ejemplo, en el que vamos a dibujar un rectángulo ubicado en la columna 50 y fila 10 con un ancho de 200 píxeles y 100 píxeles de altura:

```

<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title></title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            //Recibimos el elemento canvas
            var canvas = document.getElementById(x);
            //Comprobación sobre si encontramos un elemento
            //y podemos extraer su contexto con getContext(),
            //que indica compatibilidad con canvas
            if (canvas && canvas.getContext) {
                //Accedo al contexto de '2d' de este canvas, necesario
                //para dibujar
                var lienzo = canvas.getContext("2d");
                //devolvemos el lienzo para después realizar el dibujo
                return lienzo;
            }
            else
                return false;
        }
        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.strokeStyle = "rgb(0,0,255)";
                lienzo.strokeRect(50, 10, 200, 100);
            }
        }
    </script>
</head>
<body>
    <div id="lienzo1" style="border: 1px solid black; width: 350px; height: 350px;">
```

```

        }
    }
</script>
</head>
<body>
<header>
    <h1>CANVAS</h1>
</header>
<nav>
    <ul>
        <li>Dibujar Rectangulo</li>
    </ul>
</nav>
<aside>
</aside>
<section>
    <article>
        <canvas id="lienzo1" width="350" height="350">
            Su navegador no permite utilizar canvas.
        </canvas>
    </article>
</section>
<footer>
</footer>
</body>
</html>

```

Lo que hemos hecho es activar el color azul y dibujar el rectángulo:

```

lienzo.strokeStyle = "rgb(0, 0, 255)";

lienzo.strokeRect(50, 10, 200, 100);

```

5.3.3.3. Estilos de línea

Disponemos de varias propiedades para configurar el etilo de la línea. Mediante un ejemplo veremos como fijar valores a las propiedades: lineWidth, lineCap y lineJoin:

```

<!DOCTYPE HTML>
<html>
<head>
    <title>Canvas - Estilos de linea</title>
    <script type="text/javascript">
        function retornarLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            } else
                return false;
        }

        function dibujar() {
            var lienzo = retornarLienzo("lienzo1");
            if (lienzo) {
                lienzo.beginPath();
                lienzo.strokeStyle = "rgb(255,0,0)";
                // definimos el ancho de linea
                lienzo.lineWidth = 7;
                lienzo.moveTo(10, 5);
                lienzo.lineTo(10, 295);
                lienzo.stroke();
            }
        }
    </script>
</head>
<body>
    <h1>Canvas - Estilos de linea</h1>
    <button onclick="dibujar()">Dibujar</button>
    <div id="lienzo1" style="border: 1px solid black; width: 350px; height: 350px; margin-top: 10px;">
```

```

        lienzo.beginPath();
        lienzo.strokeStyle = "rgb(0,255,0)";
        // pintamos puntos en los extremos de la linea
        lienzo.lineCap = "butt";
        lienzo.moveTo(30, 5);
        lienzo.lineTo(30, 295);
        lienzo.stroke();
        // redondeamos el final de la linea
        lienzo.lineCap = "round";
        lienzo.beginPath();
        lienzo.moveTo(50, 5);
        lienzo.lineTo(50, 295);
        lienzo.stroke();
        // pintamos un cuadrado en el final de la linea
        lienzo.lineCap = "square";
        lienzo.beginPath();
        lienzo.moveTo(70, 5);
        lienzo.lineTo(70, 295);
        lienzo.stroke();
        lienzo.beginPath();
        lienzo.strokeStyle = "rgb(0,0,255)";
        // en las uniones pinta un cuadrado
        lienzo.lineJoin = "bevel";
        lienzo.moveTo(100, 90);
        lienzo.lineTo(130, 10);
        lienzo.lineTo(160, 90);
        lienzo.stroke();
        lienzo.beginPath();
        // redondea las uniones
        lienzo.lineJoin = "round";
        lienzo.moveTo(100, 180);
        lienzo.lineTo(130, 100);
        lienzo.lineTo(160, 180);
        lienzo.stroke();
        lienzo.beginPath();
        // las uniones las pinta en punta
        lienzo.lineJoin = "miter";
        lienzo.moveTo(100, 270);
        lienzo.lineTo(130, 190);
        lienzo.lineTo(160, 270);
        lienzo.stroke();
    }
}
</script>
</head>
<body onload="dibujar()">
<canvas id="lienzo1" width="300" height="300">
Su navegador no permite utilizar canvas.
</canvas>
</body>
</html>

```

La propiedad **lineWidth** define el ancho de la línea:

```
lienzo.lineWidth = 7;
```

La propiedad **lineCap** indica como se pintan los puntos extremos de la línea.

Los valores permitidos para esta propiedad son:

- “*butt*” lo finaliza inmediatamente.
- “*round*” genera un semicírculo.
- “*square*” dibuja un cuadrado.

La propiedad **lineJoin** indica como debe procederse a pintar en los vértices de las uniones entre los segmentos. Los valores permitidos son:

- “*bevel*” las uniones las pinta cuadradas.
- “*round*” redondea las uniones.
- “*miter*” las uniones las pinta en punta.

5.3.3.4. Dibujar rectángulo relleno

Dos capítulos antes, hemos usado strokeRect y strokeStyle para dibujar un contorno de un rectángulo y fijar su color. Ahora para dibujar un rectángulo relleno usaremos el método **fillRect** y previamente fijamos el color interior inicializando la propiedad **fillStyle**.

```
<!DOCTYPE html />
<html lang="es">
<head>
<meta charset="utf-8" />
<title>Canvas</title>
<script type="text/javascript">
    function devolverLienzo(x) {
        //Recibimos el elemento canvas
        var canvas = document.getElementById(x);
        //Comprobación sobre si encontramos un elemento
        //y podemos extraer su contexto con getContext(),
        //que indica compatibilidad con canvas
        if (canvas && canvas.getContext) {
            //Accedo al contexto de '2d' de este canvas,
            //necesario para dibujar
            var lienzo = canvas.getContext("2d");
            //devolvemos el lienzo para después realizar el dibujo
            return lienzo;
        }
        else
            return false;
    }

    window.onload = function dibujar() {
        var lienzo = devolverLienzo("lienzo1");

```

```

        if (lienzo) {
            lienzo.fillStyle = "rgb(200,0,0)";
            lienzo.fillRect(10, 10, 100, 300);
        }
    }
</script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Rectangulo Relleno</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <canvas id="lienzo1" width="350" height="350">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>

```

Activamos como color de relleno de figura el rojo (200,0,0) y seguidamente llamamos a la función `fillRect` que dibuja un rectángulo desde la posición (10,10) con ancho de 100 píxeles y un alto de 300 píxeles. La coordenada (0,0) se encuentra en la parte superior izquierda.

```

lienzo.fillStyle = "rgb(200,0,0)";
lienzo.fillRect(10, 10, 100, 300);

```

5.3.3.5. Borrar una región

Disponemos del método `clearRect` para borrar un rectángulo del lienzo. Los parámetros que debemos pasarle son los mismos que al método `fillRect`, es decir x, y, ancho y largo.

Para probar este método, primero dibujaremos un cuadrado rojo de 300px de lado y luego borraremos 3 cuadrados en la diagonal del cuadrado grande.

```

<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            //Recibimos el elemento canvas
            var canvas = document.getElementById(x);
            //Comprobación sobre si encontramos un elemento
            //y podemos extraer su contexto con getContext(),
            //que indica compatibilidad con canvas
            if (canvas && canvas.getContext) {
                //Accedo al contexto de '2d' de este canvas, necesario
                //para dibujar
                var lienzo = canvas.getContext("2d");
                //devolvemos el lienzo para después realizar el dibujo
                return lienzo;
            }
            else
                return false;
        }

        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.fillStyle = "rgb(255,0,0)";
                lienzo.fillRect(0, 0, 300, 300);
                lienzo.clearRect(10, 10, 20, 20);
                lienzo.clearRect(140, 140, 20, 20);
                lienzo.clearRect(270, 270, 20, 20);
            }
        }
    </script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Borrar Región</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <canvas id="lienzo1" width="350" height="350">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>

```

Lo primero que hacemos es dibujar el cuadrado rojo:

```
lienzo.fillStyle = "rgb(255,0,0)";
lienzo.fillRect(0, 0, 300, 300);
```

Para después, llamar 3 veces al método clearRect con las coordenadas respectivas:

```
lienzo.clearRect(10, 10, 20, 20);
lienzo.clearRect(140, 140, 20, 20);
lienzo.clearRect(270, 270, 20, 20);
```

5.3.3.6. Dibujar varias líneas y llenar la figura creada

En puntos anteriores hemos visto como crear varios segmentos y además podemos llenar su interior con un color.

Confeccionaremos un programa que muestre un triángulo con su interior coloreado:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            //Recibimos el elemento canvas
            var canvas = document.getElementById(x);
            //Comprobación sobre si encontramos un elemento
            //y podemos extraer su contexto con getContext(),
            //que indica compatibilidad con canvas
            if (canvas && canvas.getContext) {
                //Accedo al contexto de '2d' de este canvas, necesario
                //para dibujar
                var lienzo = canvas.getContext("2d");
                //devolvemos el lienzo para después realizar el dibujo
                return lienzo;
            }
            else
                return false;
        }
        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.fillStyle = "rgb(255,0,0)";
                lienzo.strokeStyle = "rgb(0,0,255)";
                lienzo.lineWidth = 5;
                lienzo.beginPath();
                lienzo.moveTo(150, 10);
                lienzo.lineTo(10, 290);
                lienzo.lineTo(290, 290);
                lienzo.lineTo(150, 10);
                lienzo.fill();
                lienzo.stroke();
            }
        }
    </script>
```

```

</head>
<body>
  <header>
    <h1>CANVAS</h1>
  </header>
  <nav>
    <ul>
      <li>Dibujar y rellenar</li>
    </ul>
  </nav>
  <aside>
  </aside>
  <section>
    <article>
      <canvas id="lienzo1" width="350" height="350">
        Su navegador no permite utilizar canvas.
      </canvas>
    </article>
  </section>
  <footer>
  </footer>
</body>
</html>

```

Explicamos línea a línea que hemos hecho con este código:

- Inicializamos el color de relleno de la figura:

```
lienzo.fillStyle = "rgb(255,0,0)";
```

- Inicializamos el color del borde de la figura:

```
lienzo.strokeStyle = "rgb(0,0,255)";
```

- Fijamos el ancho de la línea:

```
lienzo.lineWidth = 5;
```

- Indicamos que comenzamos la figura:

```
lienzo.beginPath();
```

- Movemos el puntero gráfico al primer vértice de nuestro triángulo:

```
lienzo.moveTo(150, 10);
```

- Trazamos el primer segmento:

```
lienzo.lineTo(10, 290);
```

- Trazamos el segundo segmento:

```
lienzo.lineTo(290, 290);
```

- Trazamos el último segmento (que coincide con el punto inicial):

```
lienzo.lineTo(290, 290);
```

- Confirmamos que finalizamos la figura y se procede a llenar la figura:

```
lienzo.fill();
```

- Confirmamos que dibuje el perímetro de la figura:

```
lienzo.stroke();
```



NOTA

Tengamos en cuenta que si el punto final no coincide con el punto de origen, el método fill traza este último segmento.

5.3.3.7. Arcos rellenos y lineales

Para dibujar arcos disponemos del método arc, al que debemos pasarle los siguientes parámetros:

`arc(x, y, radio, ángulo de comienzo, ángulo final, sentido antihorario).`

Con los dos parámetros primeros indicaremos el punto central del arco, el tercer parámetro es el radio. Seguidamente indicaremos el ángulo de comienzo y el ángulo final en radianes. Y por último, le pasaremos “true” en caso que el arco se dibuje en sentido antihorario y “false” en caso contrario.

Con un ejemplo veremos como hacer para dibujar distintos tipos de arcos lineales y rellenos:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }
    </script>
</head>
<body>
    <div id="lienzo"></div>
    <script type="text/javascript">
        var lienzo = devolverLienzo("lienzo");
        if (lienzo) {
            lienzo.beginPath();
            lienzo.arc(250, 250, 200, Math.PI, Math.PI * 2, true);
            lienzo.fill();
            lienzo.stroke();
            lienzo.closePath();
        }
    </script>
</body>
</html>
```

```

window.onload = function dibujar() {
    var lienzo = devolverLienzo("lienzo1");
    if (lienzo) {
        lienzo.strokeStyle = "rgb(255,0,0)";
        lienzo.beginPath();
        lienzo.arc(100, 100, 50, 0, Math.PI, true);
        lienzo.stroke();
        //sentido horario
        lienzo.strokeStyle = "rgb(0,255,0)";
        lienzo.beginPath();
        lienzo.arc(100, 200, 50, 0, Math.PI, false);
        lienzo.stroke();
        //sentido antihorario
        lienzo.fillStyle = "rgb(0,0,288)";
        lienzo.beginPath();
        lienzo.arc(200, 100, 50, 0, Math.PI, true);
        lienzo.fill();
        // arco relleno de amarillo
        lienzo.fillStyle = "rgb(255,255,0)";
        lienzo.beginPath();
        lienzo.arc(200, 200, 50, 0, Math.PI * 2, true);
        lienzo.fill();
    }
}
</script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Arcos llenos y lineales</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <canvas id="lienzo1" width="350" height="350">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>

```

En este ejemplo vamos a pintar 4 arcos diferentes. En el primer arco vamos a pintar solo el perímetro de color rojo y comenzaremos en el ángulo cero y lo finalizaremos en los 180 grados (en radianes es PI), y le indicamos que avance en sentido antihorario:

```
lienzo.strokeStyle = "rgb(255,0,0)";
lienzo.beginPath();
lienzo.arc(100, 100, 50, 0, Math.PI, true);
lienzo.stroke();
```

En el segundo arco, será similar al arco anterior pero indicándole que lo dibuje en sentido horario:

```
lienzo.strokeStyle = "rgb(0,255,0)";
lienzo.beginPath();
lienzo.arc(100, 200, 50, 0, Math.PI, false);
lienzo.stroke();
```

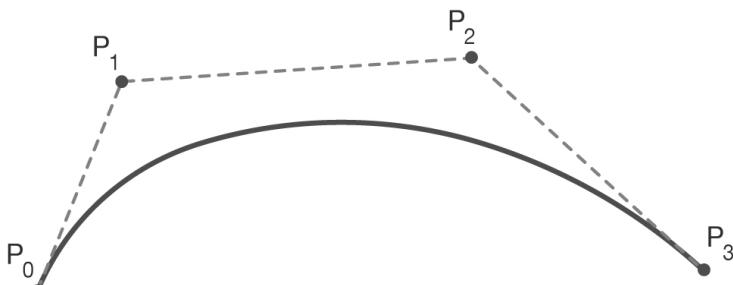
Dibujamos el tercer arco y después le indicamos que se rellene llamando al método fill en lugar de stroke:

```
lienzo.fillStyle = "rgb(0,0,288)";
lienzo.beginPath();
lienzo.arc(200, 100, 50, 0, Math.PI, true);
lienzo.fill();
```

Y por último, dibujamos un círculo relleno de color amarillo:

```
lienzo.fillStyle = "rgb(255,255,0)";
lienzo.beginPath();
lienzo.arc(200, 200, 50, 0, Math.PI * 2, true);
lienzo.fill();
```

5.3.3.8. Curva de Bézier



El concepto de curva de Bézier lo podemos leer en la Wikipedia:

"Se denomina curva de Bézier a un sistema que se desarrolló hacia los años 1960, para el trazado de dibujos técnicos, en el diseño aeronáutico y de automóviles. Su denominación es en honor a Pierre Bézier, quien ideó un método de descripción matemática de las curvas que se comenzó a utilizar con éxito en los programas de CAD."

"Las curvas de Bézier fueron publicadas, por primera vez en 1962 por el ingeniero francés Pierre Bézier, que las usó posteriormente, con profusión, en el diseño de las diferentes partes de los cuerpos de un automóvil, en sus años de trabajo en la Renault. Las curvas fueron desarrolladas por Paul de Casteljau usando el algoritmo que lleva su nombre. Se trata de un método numéricamente estable para evaluar las curvas de Bézier."

Posteriormente, los inventores del PostScript, lenguaje que permitió el desarrollo de sistemas de impresión de alta calidad desde el ordenador, introdujeron en ese código el método de Bézier para la generación del código de las curvas y los trazados. El lenguaje PostScript sigue empleándose ampliamente y se ha convertido en un estándar de calidad universal; por ello, los programas de diseño vectorial como Adobe Illustrator, el extinto Macromedia FreeHand, Corel Draw, tres de los más importantes programas de dibujo vectorial y otros como Inkscape, denominan como "bézier" a algunas de sus herramientas de dibujo, y se habla de "Trazados bézier", "pluma bézier", "lápiz bézier", etc. Su facilidad de uso la ha estandarizado en el diseño gráfico, extendiéndose también a programas de animación vectorial como Adobe Flash, y retoque fotográfico (bitmap) como Photoshop y Gimp, donde se usa para crear formas cerradas o selecciones.

La idea de definir geométricamente las formas no es demasiado compleja: un punto del plano puede definirse por coordenadas. Por ejemplo, un punto A tiene unas coordenadas (x_1, y_1) y a un punto B le corresponde (x_2, y_2). Para trazar una recta entre ambos basta con conocer su posición.

Si en lugar de unir dos puntos con una recta se unen con una curva, surgen los elementos esenciales de una curva Bézier: los puntos se denominan puntos de anclaje o nodos. La forma de la curva se define por unos puntos invisibles en el dibujo, denominados puntos de control, manejadores o manecillas."

Quizás sea poco útil, dependiendo de la utilidad que le vayáis a dar a vuestra página, pero es interesante saber hacerla, por si algún día necesitáis usarla.

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.strokeStyle = "rgb(255,0,0)";
                lienzo.beginPath();
                lienzo.moveTo(0, 150);
                lienzo.bezierCurveTo(100, 50, 200, 250, 290,
150);
                lienzo.stroke();
            }
        }
    </script>
</head>
<body>
    <div id="lienzo1" style="border: 1px solid black; width: 300px; height: 300px;">
```

```

        }
    }
</script>
</head>
<body>
<header>
    <h1>CANVAS</h1>
</header>
<nav>
    <ul>
        <li>Curva Beizer</li>
    </ul>
</nav>
<aside>
</aside>
<section>
    <article>
        <canvas id="lienzo1" width="350" height="350">
            Su navegador no permite utilizar canvas.
        </canvas>
    </article>
</section>
<footer>
</footer>
</body>
</html>
```

Usaremos el método strokeStyle para decirle que será únicamente el contorno lo que pintaremos. Despu s inicializamos el primer punto fijo llamando al m todo moveTo:

```
lienzo.moveTo(0, 150);
```

El segundo punto fijo est  indicado en los dos \'ltimos par metros del m todo bezierCurveTo (es decir 290,150), en tanto que los dos puntos invisibles son los valores (100,50) y (200,250):

```
lienzo.bezierCurveTo(100, 50, 200, 250, 290, 150);
```

5.3.3.9. M todo quadraticCurveTo

El concepto de este tipo de curva es similar a la de B ezier. La diferencia con el m todo anterior, es que solo hay un punto de atracci n.

Veamos un ejemplo de su implementaci n que mostrar  una curva de este tipo y desplazaremos el punto de atracci n en forma vertical, y cada vez que lo desplacemos, vamos a borrar el canvas y volveremos a pintar, d ndole un efecto el stico:

```
<!DOCTYPE html>

<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                lienzo.clearRect(0, 0, 350, 350);
                lienzo.moveTo(0, 150);
                lienzo.bezierCurveTo(100, 50, 200, 250, 290, 150);
                lienzo.stroke();
            }
        }
    </script>
</head>
<body>
    <h1>CANVAS</h1>
    <nav>
        <ul>
            <li>Curva Beizer</li>
        </ul>
    </nav>
    <aside>
        
    </aside>
    <section>
        <article>
            <canvas id="lienzo1" width="350" height="350">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
        
    </footer>
</body>
</html>
```

```

        return lienzo;
    }
    else
        return false;
}

var fila = 0;
function dibujar() {
    var lienzo = devolverLienzo("lienzo1");
    if (lienzo) {
        lienzo.clearRect(0, 0, 300, 300);
        lienzo.strokeStyle = "rgb(255,0,0)";
        lienzo.beginPath();
        lienzo.moveTo(0, 150);
        lienzo.quadraticCurveTo(150, fila, 300, 150);
        lienzo.stroke();
        fila++;
        if (fila > 300)
            fila = 0;
    }
}

window.onload = function iniciar() {
    setInterval(dibujar, 10);
}
</script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Metodo quadraticCurveTo</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <canvas id="lienzo1" width="350" height="350">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>
```

En este ejemplo, la función que se ejecutará al cargar la página, no va a ser la de dibujar(). La función setInterval() será la que se ejecute y le pasaremos como parámetro la función que queremos que se ejecute cada 10 milisegundos:

```
function iniciar() {
    setInterval(dibujar, 10);
}
```

Cuando ejecutemos la función dibujar, lo primero que haremos será borrar el contenido del canvas:

```
lienzo.clearRect(0, 0, 300, 300);
```

Una vez que hemos borrado el canvas, primero definimos el color de la línea, inicializamos el camino con beginPath(), definimos el primer punto fijo de la curva mediante el método moveTo() y llamamos al método quadraticCurveTo(), pasándole como parámetro el punto móvil y con los dos últimos parámetros, le indicamos el segundo punto fijo de la curva:

```
lienzo.strokeStyle = "rgb(255,0,0)";
lienzo.beginPath();
lienzo.moveTo(0, 150);
lienzo.quadraticCurveTo(150, fila, 300, 150);
lienzo.stroke();
```

Después incrementamos la variable global que permite modificar en cada paso la ubicación del punto de atracción, y cuando la variable fila llegue a 300, volverá a asignarle el valor 0 :

```
fila++;
if (fila > 300)
    fila = 0;
```

5.3.3.10. Pintar texto

Canvas nos provee de diversos métodos y propiedades para tratar el texto. Por ejemplo, para imprimir texto disponemos de:

```
fillText("Texto", x, y);
```

Con el método fillText() imprimiremos un String en las coordenadas “x” e “y” indicadas. Debemos tener en cuenta que esta última coordenada corresponde con la parte inferior del texto. Es decir, no se mostrará si indicamos el valor cero en y.

Con el método strokeText() solo pintaremos el contorno de cada letra:

```
strokeText("Texto", x, y)
```

Para seleccionar la fuente a aplicar la inicializaremos con la propiedad Font. Por ejemplo:

```
lienzo.font = "bold 25px Arial";
```

Con la propiedad `textAlign` podremos hacer que comience la impresión a partir de la coordenada indicada o que este sea el punto central o final, etc.

Los valores posibles que podemos asignarle a esta propiedad son:

- “start”.
- “end”.
- “left”.
- “right”.
- “center”.

Por defecto, si no definimos esta propiedad se inicializará con “start”.

```
lienzo.textAlign = "right";
```

También tenemos un método para conocer el ancho en píxeles de un String según la fuente actual, es el método `measureText()`:

```
lienzo.measureText("Texto");
```

Vamos a implementar todos estos métodos con un ejemplo de manejo del texto dentro del canvas:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.fillStyle = "rgb(255,0,0)";
                lienzo.font = "bold 25px Arial";
                lienzo.fillText("SEAS", 150, 50);
                lienzo.textAlign = "center";
                lienzo.fillText("SEAS", 150, 100);
                lienzo.textAlign = "right";
                lienzo.fillText("SEAS", 150, 150);
                var anchopx = lienzo.measureText("SEAS");
                lienzo.textAlign = "start";
            }
        }
    </script>
</head>
<body>
    <div id="lienzo1" style="width: 300px; height: 300px; border: 1px solid black; position: relative; margin: auto; margin-top: 100px;"></div>
</body>

```

```

        lienzo.fillText(anchopx.width, 150, 200);
        lienzo.strokeStyle = "rgb(0,0,255)";
        lienzo.strokeText("Fin", 150, 250);
    }
}
</script>
</head>
<body>
<header>
    <h1>CANVAS</h1>
</header>
<nav>
    <ul>
        <li>Pintar texto</li>
    </ul>
</nav>
<aside>
</aside>
<section>
    <article>
        <canvas id="lienzo1" width="350" height="350">
            Su navegador no permite utilizar canvas.
        </canvas>
    </article>
</section>
<footer>
</footer>
</body>
</html>
```

Expliquemos que hemos hecho con cada método:

- Definimos el color rojo para el texto:

```
lienzo.fillStyle = "rgb(255,0,0);
```

- Definimos el tipo de fuente Arial de 25 píxeles y estilo bold:

```
lienzo.font = "bold 25px Arial";
```

- Pintamos el texto “SEAS”:

```
lienzo.fillText("SEAS", 150, 50);
```

- Cambiamos el tipo de alineación del texto:

```
lienzo.textAlign = "center";
```

- Imprimimos el texto centrado con respecto a la columna 150:

```
lienzo.fillText("SEAS", 150, 100);
```

- Cambiamos la alineación nuevamente:

```
lienzo.textAlign = "right";
```

- Imprimimos el texto y como tenemos alineación a derecha el texto finaliza en la columna 150:

```
lienzo.fillText("SEAS", 150, 150);
```

- Llamamos al método `measureText()` y le pasamos como parámetro el texto que imprimimos. Luego este método nos devolverá un objeto de la clase `TextMetrics`. Será la propiedad `width` la que almacenará el ancho en píxeles del texto que pasamos anteriormente:

```
var anchopx = lienzo.measureText("SEAS");
```

- Cambiamos el tipo de alineación:

```
lienzo.textAlign = "start";
```

- Imprimimos la propiedad `width`, que era la que almacenaba el ancho en píxeles:

```
lienzo.fillText(anchopx.width, 150, 200);
```

- Inicializamos de nuevo el color de línea:

```
lienzo.strokeStyle = "rgb(0,0,255)";
```

- Y por último, dibujamos una línea con el método `strokeText`:

```
lienzo.strokeText("Fin", 150, 250);
```

5.3.4. Pintar imágenes

En Canvas disponemos del método `drawImage` para pintar imágenes. Debemos saber que este método se va a comportar de diferente manera según la cantidad de parámetros que le pasemos. Las más usadas son las siguientes:

- **`drawImage(objetolImagen, x, y)`**: dibujamos la imagen con el tamaño real de la imagen, en la coordenada x e y.
- **`drawImage(objetolImagen, x, y, ancho, alto)`**: redimensionamos la imagen con los valores indicados en ancho y alto.
- **`drawImage(objetolImagen, x1, y1, ancho1, alto1, x2, y2, ancho2, alto2)`**: los primeros cuatro parámetros después del `ObjetolImagen` nos indican el trozo de la imagen y los cuatro restantes es el lugar donde se debe pintara ese trozo de imagen.

Veamos las tres formas anteriores en un ejemplo:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.fillStyle = "rgb(100,0,0)";
                lienzo.fillRect(0, 0, 600, 600);
                img1 = new Image();
                img1.src = "../imagenes/logotipo.png";
                img1.onload = function () {
                    lienzo.drawImage(img1, 150, 0);
                    lienzo.drawImage(img1, 300, 150, 100, 50);
                    lienzo.drawImage(img1, 0, 0, 150, 40, 100, 350, 72,);
                }
            }
        }
    </script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Dibujar Imagen</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <canvas id="lienzo1" width="600" height="600">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>
```

Lo que hemos hecho en este ejemplo es lo siguiente:

- Primero creamos en el lienzo un cuadrado para pintar encima las imágenes:

```
lienzo.fillStyle = "rgb(100,0,0)";
lienzo.fillRect(0, 0, 600, 600);
```

- Necesitamos crear un objeto de la clase Image:

```
img1 = new Image();
```

- Inicializamos la propiedad src con la fuente de la imagen a cargar:

```
img1.src = "imagenes/logotipo.png";
```

- Despues inicializamos la propiedad onload del objeto de la clase Image con una función anónima que se ejecuta cuando se finaliza la carga de la imagen en el navegador:

```
img1.onload = function () {
    lienzo.drawImage(img1, 150, 0);
    lienzo.drawImage(img1, 300, 150, 100, 50);
    lienzo.drawImage(img1, 0, 0, 150, 40, 100, 350, 72, 72);
}
```

Con esta función anónima realizamos lo siguiente:

- Primero dibujamos la imagen en su tamaño original:

```
lienzo.drawImage(img1, 0, 0);
```

- Despues la dibujamos con un ancho de 100 píxeles y 50 del alto:

```
lienzo.drawImage(img1, 300, 150, 100, 50);
```

- Por ultimo, la vamos a dibujar extrayendo de la imagen original 150 píxeles de ancho y 40 de alto y dibujándola en la coordenada (100, 350) con un ancho y alto de 72 píxeles.

```
lienzo.drawImage(img1, 0, 0, 150, 40, 100, 350, 72, 72);
```

5.3.4.1. Transparencias

Con el canvas también podemos realizar transparencias inicializando la propiedad fillStyle. Para ello debemos pasar a la función rgba los valores para el rojo, verde, azul y el grado de transparencia que será un valor entre 0 y 1.

Si pasamos el valor 1, la figura se volverá totalmente opaca, es decir, no se verá absolutamente nada de lo que se encuentra detrás de la misma. A medida que disminuimos dicho valor, la figura se volverá más transparente.

Veamos un ejemplo donde implementamos como activar la transparencia:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.fillStyle = "rgb(255,0,0)";
                lienzo.fillRect(10, 10, 250, 250);
                lienzo.fillStyle = "rgba(0,255,0,0.3)";
                lienzo.fillRect(200, 200, 250, 250);
            }
        }
    </script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Transparencias</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <canvas id="lienzo1" width="600" height="600">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>
```

En este ejemplo, vamos a pintar dos cuadrados, uno rojo y otro verde. Ambos tendrán en común una zona, que será en la que se mezclen ambos colores. El segundo de ellos tendrá una transparencia de 0,3 para que se pueda realizar la mezcla. Veamos como realizarlo:

- Lo primero activamos el color rojo:

```
lienzo.fillStyle = "rgb(255,0,0)";
```

- Dibujamos un cuadrado de ese color.

```
lienzo.fillRect(10, 10, 250, 250);
```

- Después activamos el color verde y con un nivel de transparencia de 0.3:

```
lienzo.fillStyle = "rgba(0,255,0,0.3)";
```

- Y por último, dibujamos el cuadrado superpuesto al primero:

```
lienzo.fillRect(200, 200, 250, 250);
```

5.3.4.2. Sombras

El elemento Canvas nos permite administrar las sombras de una figura. Para ello tenemos una serie de propiedades que debemos inicializar antes de comenzar a pintar.

La propiedad **shadowOffsetX** nos indica como se desplaza la sombra con respecto a "x". Inicializaremos dos parámetros:

- Un valor positivo indicará que la sombra se verá del lado derecho de la figura
- Un valor negativo hará que la sombra aparezca a la izquierda.

La propiedad **shadowOffsetY** hace la misma operación pero respecto a "y":

- Un valor positivo indicará que la sombra se verá en la parte inferior de la figura.
- Un valor negativo hará que aparezca en la parte superior la sombra.

Con la propiedad **shadowColor** fijaremos el color de la sombra.

Y, por último, con la propiedad **shadowBlur** fijaremos el difuminado de la sombra. Si le indicamos un valor nulo, no existirá el difuminado. Con valores positivos aumentará el difuminado, y con valores negativos disminuirá el difuminado de la sombra.

Implementemos un ejemplo para verlo mejor. Vamos a pintar un cuadrado y un rectángulo de color azul, a los que les pintaremos las sombras, al rectángulo, en la parte inferior e izquierda y al cuadrado, en la superior y en la derecha de la figura:

```

<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.shadowOffsetX = -10;
                lienzo.shadowOffsetY = 5;
                lienzo.shadowBlur = 10;
                lienzo.shadowColor = "rgba(0, 0, 255, 0.5)";
                lienzo.fillStyle = "rgb(0,0,255)";
                lienzo.fillRect(20, 20, 150, 100);
                lienzo.shadowOffsetX = 5;
                lienzo.fillRect(250, 20, 100, 100);
            }
        }
    </script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Sombras</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <canvas id="lienzo1" width="600" height="600">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>

```

Analicemos el código línea a línea:

- Indicamos que la sombra aparezca en el lado izquierdo:

```
lienzo.shadowOffsetX = -10;
```

- Indicamos que la sombra aparezca en la parte inferior:

```
lienzo.shadowOffsetY = 5;
```

- Definimos el difuminado de la sombra:

```
lienzo.shadowBlur = 10;
```

- Definimos el color de la sombra y su transparencia:

```
lienzo.shadowColor = "rgba(0, 0, 255, 0.5);
```

- Inicializamos el color de la figura:

```
lienzo.fillStyle = "rgb(0,0,255);
```

- Dibujamos el rectángulo con las sombras previamente definidas:

```
lienzo.fillRect(20, 20, 150, 100);
```

- Cambiamos la orientación en “x” de la sombra para que la pinte en la derecha.

```
lienzo.shadowOffsetX = 5;
```

- Cambiamos la orientación en “y” de la sombra para que la pinte en la parte superior:

```
lienzo.shadowOffsetY = -5;
```

- Pintamos el cuadrado con las sombras anteriormente definidas:

```
lienzo.fillRect(250, 20, 100, 100);
```

5.3.4.3. Gradiente

Hasta aquí habíamos visto como llenar una figura con un color liso. Los gradientes nos permiten llenar una figura con dos o más colores.



Gradiente se usa como sinónimo de “degradado”: la transición suave y sin saltos de un color a otro. Por ejemplo, de rojo a verde, de blanco a negro, de azul a morado.

Lo que haremos será indicar dos colores y luego las librerías disponibles nos generan todos los colores intermedios entre los colores indicados.

Veremos primero la gradiente lineal.

5.3.4.3.1. Gradiente lineal

Los métodos necesarios para implementar una gradiente lineal son:

```
objetoCanvasGradient = lienzo.createLinearGradient(x1, y1, x2, y2);
```

Los cuatro parámetros indican dos puntos del canvas, donde comenzará el degradado y donde terminará el mismo. Después en dichas coordenadas pintaremos la figura. Seguidamente debemos llamar al método:

```
objetoCanvasGradient. addColorStop([valor entre 0 y 1],color);
```

Con addColorStop, definiremos el color inicial y el color final del degradado. Por lo que lo instanciaremos dos veces como mínimo, dependiendo de número de colores por los que queremos que pase el degradado.

Con un ejemplo quedará más claro:

```
var gradiente1 = lienzo.createLinearGradient(0, 0, 200, 0);
gradiente1.addColorStop(0, "rgb(255,0,0)");
gradiente1.addColorStop(1, "rgb(0,0,0)");
lienzo.fillStyle = gradiente1;
lienzo.fillRect(0, 0, 200, 200);
```

- Primero creamos un objeto de la clase CanvasGradient al que le pasamos los puntos donde comenzará y acabará el degradado :

```
var gradiente1 = lienzo.createLinearGradient(0, 0, 200, 0);
```

- Definimos el color inicial del degradado con 0:

```
gradiente1.addColorStop(0, "rgb(255,0,0)");
```

- Definimos el color del punto final, asignándole 1:

```
gradiente1.addColorStop(1, "rgb(0,0,0)");
```

- Inicializamos la propiedad fillStyle con la que definíamos el color de un rectángulo y le asignamos el objeto de tipo CanvasGradient:

```
lienzo.fillStyle = gradiente1;
```

- Y ya por último dibujamos el cuadrado:

```
lienzo.fillRect(0, 0, 200, 200);
```

Luego al verlo en ejecución, visualizaremos que el cuadrado comienza con color rojo en el lado de la izquierda y después va variando al negro.

Veamos un ejemplo, en el que vamos a realizar cuatro degradados diferentes:

- El primero irá de rojo a negro, siendo el rojo el color inicial, en la izquierda y el negro, color final, en la derecha, yendo en ese sentido.
- El segundo irá de negro a rojo, pero haciéndolo al revés que el anterior, diciendo que el rojo es el inicial, el negro es el final, pero pintándolo de derecha a izquierda.
- El tercero será igual que el segundo, pero haciéndolo como el primero, es decir, el color inicial será el negro, el color final será el rojo, y los pintará de izquierda a derecha.
- El cuarto será un degradado negro a rojo, pero pasando por el verde.

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                var gradiente1 = lienzo.createLinearGradient(0, 0, 200, 0);
                gradiente1.addColorStop(0, "rgb(255,0,0)");
                gradiente1.addColorStop(1, "rgb(0,0,0)");
                lienzo.fillStyle = gradiente1;
                lienzo.fillRect(0, 0, 200, 200);

                var gradiente2 = lienzo.createLinearGradient(200, 300, 0, 300);
                gradiente2.addColorStop(0, "rgb(255,0,0)");
                gradiente2.addColorStop(1, "rgb(0,0,0)");
                lienzo.fillStyle = gradiente2;
                lienzo.fillRect(0, 300, 200, 200);

                var gradiente3 = lienzo.createLinearGradient(300, 0, 500, 0);
                gradiente3.addColorStop(0, "rgb(0,0,0)");
                gradiente3.addColorStop(1, "rgb(255,0,0)");
                lienzo.fillStyle = gradiente3;
                lienzo.fillRect(300, 0, 200, 200);
            }
        }
    </script>
</head>
<body>
    <div id="lienzo1" style="width: 400px; height: 400px; border: 1px solid black;">
```

```

var gradiente4 = lienzo.createLinearGradient(500, 300,
    300, 300);
gradiente4.addColorStop(0, "rgb(255,0,0)");
gradiente4.addColorStop(0.5, "rgb(0,255,0)");
gradiente4.addColorStop(1, "rgb(0,0,0)");
lienzo.fillStyle = gradiente4;
lienzo.fillRect(300, 300, 200, 200);

}
}

</script>
</head>
<body>
<header>
    <h1>CANVAS</h1>
</header>
<nav>
    <ul>
        <li>Gradiente Lineal</li>
    </ul>
</nav>
<aside>
</aside>
<section>
    <article>
        <canvas id="lienzo1" width="500" height="500">Su
navegador no permite utilizar canvas.
        </canvas>
    </article>
</section>
<footer>
</footer>
</body>
</html>

```

El ejemplo realizado quedará de la siguiente manera:

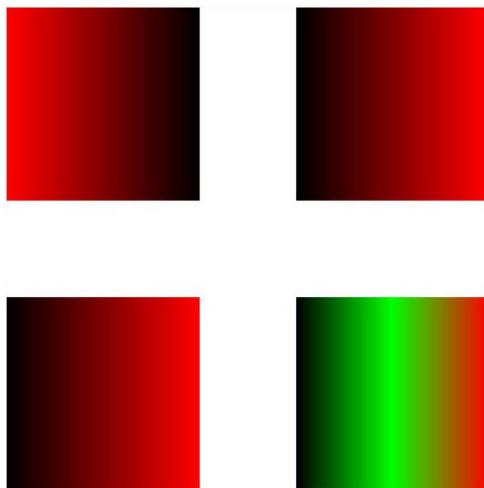


Figura 5.142. Imagen de los 4 degradados.

5.3.4.3.2. Gradiente radial

El método `createRadialGradient` funciona igual que los gradientes radiales, pero especificando el radio de comienzo inicial y radio final.

```
lienzo.createRadialGradient(x1, y1, radioInicial, x2, y2, radioFinal);
```

Veamos un ejemplo donde implementamos dicho ejemplo de degradado:

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Canvas - Gradiente Radial</title>
    <script type="text/javascript">
        function retornarLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            } else
                return false;
        }

        function dibujar() {
            var lienzo = retornarLienzo("lienzo1");
            if (lienzo) {
                var gradiente1 = lienzo.createRadialGradient(200,
200, 0, 200, 200, 200);
                gradiente1.addColorStop(0, "red");
                gradiente1.addColorStop(1, "rgb(0,0,255)");
                lienzo.fillStyle = gradiente1;
                lienzo.arc(200, 200, 100, 0, Math.PI * 2, true);
                lienzo.fill();
            }
        }
    </script>
</head>
<body onload="dibujar()">
    <canvas id="lienzo1" width="600" height="600">
        Su navegador no permite utilizar canvas.
    </canvas>
</body>
</html>
```

La diferencia en el uso respecto al gradiente lineal, es que en este usábamos el método `fillRect()` para pintar el degradado. Con el gradiente radial, usaremos el método `arc`, que explicamos en el punto 5.3.3.7

```
lienzo.arc(200, 200, 100, 0, Math.PI * 2, true);
```

5.3.4.4. Patrones de imágenes

Ya hemos visto en capítulos anteriores, que cuando se pinta el interior de una figura podemos utilizar como relleno:

- Un color fijo.
- Un gradiente.

En este capítulo vamos a aprender a utilizar una imagen como relleno de una figura.

Para poder usar como relleno una imagen primero tenemos que crear un objeto de tipo CanvasPattern, empleando el método createPattern:

```
CanvasPattern createPattern(Object Image, string repetición)
```

Es decir, lo que hacemos es pasar como parámetro un objeto de tipo Image y un string que tomará alguno de estos valores:

- “repeat”: la imagen se repite en “x” y en “y” completando toda la figura.
- “no-repeat”: si la figura es mayor a la imagen, luego quedará vacío parte de la imagen.
- “repeat-x”: la imagen se repite en el eje “x”.
- “repeat-y”: la imagen se repite en el eje “y”.

Crearemos un ejemplo donde pintaremos un cuadrado de 600*600 píxeles y pintaremos su interior utilizando una imagen:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                img1 = new Image();
                img1.src = "../imagenes/logotipo.png";
                img1.onload = function () {

```

```

        var patron = lienzo.createPattern(img1, 'repeat');
        lienzo.fillStyle = patron;
        lienzo.fillRect(0, 0, 600, 600);
    }
}
</script>
</head>
<body>
<header>

    <h1>CANVAS</h1>
</header>
<nav>
    <ul>
        <li>Patrones de Imagenes</li>
    </ul>
</nav>
<aside>
</aside>
<section>
    <article>
        <canvas id="lienzo1" width="600" height="600">
            Su navegador no permite utilizar canvas.
        </canvas>
    </article>
</section>
<footer>
</footer>
</body>
</html>

```

Lo primero que hacemos es crear un objeto de tipo Image e inicializar la propiedad src con la situación de la imagen propiamente dicha:

```

if (lienzo) {
    img1 = new Image();
    img1.src = "../imagenes/logotipo.png";

```

En el momento en el que se termine de cargar por completo la imagen crearemos el patrón y dibujaremos el rectángulo inicializando la propiedad fillStyle con el patrón creado:

```

img1.onload = function () {
    var patron = lienzo.createPattern(img1, 'repeat');
    lienzo.fillStyle = patron;
    lienzo.fillRect(0, 0, 600, 600);
}

```

5.3.4.5. Canvas como patrón

Hasta ahora, y desde el capítulo anterior, podemos llenar el interior de una figura con:

- Un color fijo.
- Un gradiente.
- Una imagen.

En este capítulo descubriremos como usar el contenido de otro Canvas como patrón.

Para poder utilizar como relleno otro canvas, primero debemos crear un objeto de tipo CanvasPattern empleando el método createPattern:

```
CanvasPattern createPattern(HTMLCanvasElement imagen, string repetición)
```

El parámetro repetición podrá tomar los siguientes valores:

- “repeat”: la imagen se repite en “x” y en “y” completando toda la figura.
- “no-repeat”: si la figura es mayor a la imagen, luego quedará vacío parte de la imagen.
- “repeat-x”: la imagen se repite en el eje “x”.
- “repeat-y”: la imagen se repite en el eje “y”.

Veámoslo con un ejemplo en el que realizamos una página que tenga dos CANVAS, en el primero pintaremos una cruz y en el segundo pintaremos un cuadrado que cubra todo el canvas utilizando como patrón el primer canvas:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        window.onload = function dibujar() {
            var lienzol = devolverLienzo("lienzol");
            if (lienzol) {
```

```

        lienzo1.strokeStyle = "rgb(0,0,255)";
        lienzo1.beginPath();
        lienzo1.moveTo(25, 0);
        lienzo1.lineTo(25, 50);
        lienzo1.stroke();
        lienzo1.beginPath();
        lienzo1.moveTo(0, 25);
        lienzo1.lineTo(50, 25);
        lienzo1.stroke();
        var lienzo2 = devolverLienzo("lienzo2");
        if (lienzo2) {
            var patron = lienzo2.createPattern(lienzo1.
            canvas, 'repeat');
            lienzo2.fillStyle = patron;
            lienzo2.fillRect(0, 0, 300, 300);
        }
    }
}

</script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Canvas como patrón</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <p>Primer Canvas (patrón del segundo Canvas)</p>
            <canvas id="lienzo1" width="50" height="50">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
        <article>
            <p>Segundo Canvas</p>
            <canvas id="lienzo2" width="600" height="600">
                Su navegador no permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>

```

Primero tenemos que obtener la referencia del primer canvas y mediante los métodos vistos en capítulos anteriores, pintamos una cruz azul:

```
var lienzo1 = devolverLienzo("lienzo1");
```

```
if (lienzo1) {
    lienzo1.strokeStyle = "rgb(0,0,255)";
    lienzo1.beginPath();
    lienzo1.moveTo(25, 0);
    lienzo1.lineTo(25, 50);
    lienzo1.stroke();
    lienzo1.beginPath();
    lienzo1.moveTo(0, 25);
    lienzo1.lineTo(50, 25);
    lienzo1.stroke();
```

Una vez que ya tenemos creado y pintado nuestro primer Canvas, vamos a obtener la referencia del segundo canvas (lienzo2) y pintamos un cuadrado, pero creando previamente un patrón y utilizando el primer canvas:

```
var lienzo2 = devolverLienzo("lienzo2");
if (lienzo2) {
    var patron = lienzo2.createPattern(lienzo1.canvas, 'repeat');
    lienzo2.fillStyle = patron;
    lienzo2.fillRect(0, 0, 300, 300);
}
```

Y para que se pinte en nuestra página ambos elementos, debemos definir los dos elementos de tipo Canvas:

```
<section>
<article>
    <p>Primer Canvas (patrón del segundo Canvas)</p>
    <canvas id="lienzo1" width="50" height="50">
        Su navegador no permite utilizar canvas.
    </canvas>
</article>

<article>
    <p>Segundo Canvas</p>
    <canvas id="lienzo2" width="600" height="600">
        Su navegador no permite utilizar canvas.
    </canvas>
</article>
</section>
```

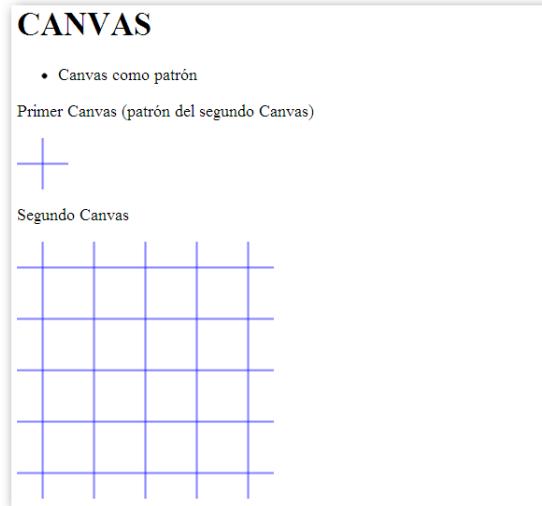


Figura 5.143. Imagen de nuestro ejemplo en ejecución.

5.3.4.6. Grabar y recuperar el estado

Imagináros que en una aplicación, tenemos que hacer un dibujo complejo. Podría ser que necesitásemos guardar el estado de algunas propiedades del canvas para recuperarlas más adelante. Estas dos acciones las realizaremos mediante los métodos:

- `Save()`.
- `Restore()`.

Los valores que se pueden almacenar, entre otras propiedades, son `strokeStyle`, `fillStyle`, `lineJoin`, `lineCap`, `shadowOffsetY`, `shadowOffsetX`, `miterLimit`, `shadowBlur`, `shadowColor`, `globalAlpha`...



Debéis recordar que si llamáis dos veces seguidas al método `save()`, para volver al estado inicial, deberéis llamar dos veces al método `restore()`.

Para ver su funcionamiento más claramente, vamos a desarrollar un programa que muestre 3 cuadrados, uno en color rojo, otro con un gradiente del rojo al azul y finalmente otro cuadrado rojo:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
```

```

        if (canvas && canvas.getContext) {
            var lienzo = canvas.getContext("2d");
            return lienzo;
        }
        else
            return false;
    }

    function dibujarCuadradoGradiente(lienzo, x, y, ancho,
alto) {
        lienzo.save();
        var gradiente1 = lienzo.createLinearGradient(x, y, x
+ ancho, y + alto);
        gradiente1.addColorStop(0, "rgb(255,0,0)");
        gradiente1.addColorStop(1, "rgb(0,0,255)");
        lienzo.fillStyle = gradiente1;
        lienzo.fillRect(x, y, ancho, alto);
        lienzo.restore();
    }

    window.onload = function dibujar() {
        var lienzo = devolverLienzo("lienzo1");
        if (lienzo) {
            lienzo.fillStyle = "rgb(255,0,0)";
            lienzo.fillRect(10, 30, 100, 100);
            dibujarCuadradoGradiente(lienzo, 120, 30, 100,
100);
            lienzo.fillRect(230, 30, 100, 100);
        }
    }
</script>
</head>
<body>
<header>
    <h1>CANVAS</h1>
</header>
<nav>
    <ul>
        <li>Save () - Restore ()</li>
    </ul>
</nav>
<aside>
</aside>
<section>
    <article>
        <canvas id="lienzo1" width="600" height="600">Su
navegador no permite utilizar canvas.
    </canvas>
    </article>
</section>
<footer>
</footer>
</body>
</html>

```

Cuando cargue la página completamente, lo primero dibujaremos un cuadrado de color rojo. Usaremos fillStyle para fijar el color y fillRect para llenar el cuadrado:

```
lienzo.fillStyle = "rgb(255,0,0)";
lienzo.fillRect(10, 30, 100, 100);
```

Después llamamos a la función dibujarCuadradoGradiente(), al que le pasaremos 5 parámetros:

```
dibujarCuadradoGradiente(lienzo, 120, 30, 100, 100);
```

Analicemos esta función con detenimiento. Lo primero que hacemos es grabar el estado actual llamando al método save() y después crearemos un objeto de tipo LinearGradient (recordar el capítulo del gradiente lineal) para asignárselo al lienzo mediante fillStyle y pintarlo con fillRect, pasándole los parámetros que le habíamos pasado a la función. Y por último, después de pintar el cuadrado, restauramos el estado actual llamando a la función restore():

```
function dibujarCuadradoGradiente(lienzo, x, y, ancho,
alto) {
    lienzo.save();
    var gradiente1 = lienzo.createLinearGradient(x, y, x
+ ancho, y + alto);
    gradiente1.addColorStop(0, "rgb(255,0,0)");
    gradiente1.addColorStop(1, "rgb(0,0,255)");
    lienzo.fillStyle = gradiente1;
    lienzo.fillRect(x, y, ancho, alto);
    lienzo.restore();
}
```

Una vez que ha pintado el segundo cuadrado, al haber restaurado el estado dentro de la función, le decimos que vuelva a pintar el cuadrado, pasándole las nuevas especificaciones al método fillRect:

```
lienzo.fillRect(230, 30, 100, 100);
```

5.3.4.7. Transformaciones en canvas

El elemento Canvas nos permite realizar una serie de transformaciones, que nos serán útiles a la hora de pintar en la página. Estas son las siguientes:

- Trasladar.
- Rotar.
- Escalar.

En los puntos siguientes veremos como realizar cada una de estas transformaciones.

5.3.4.7.1. Trasladar

Canvas nos permite mover el sistema de coordenadas. Para realizar esto usaremos el método `translate()`. Después de realizar la llamada a este método, la coordenada (0,0) corresponderá al par de valores indicados en los parámetros:

```
lienzo.translate(x, y);
```

Confeccionaremos un programa que implemente una función que dibuje un triángulo, la función recibe la coordenada donde debe dibujarla y la base y altura del triángulo:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        function dibujarTriangulo(lienzo, x, y, base, altura) {
            lienzo.save();
            lienzo.translate(x, y);
            lienzo.fillStyle = "rgb(255,0,0)";
            lienzo.beginPath();
            lienzo.moveTo(base / 2, 0);
            lienzo.lineTo(0, altura);
            lienzo.lineTo(base, altura);
            lienzo.lineTo(base / 2, 0);
            lienzo.fill();
            lienzo.restore();
        }

        window.onload = function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                for (var col = 0; col < 550; col += 30)
                    dibujarTriangulo(lienzo, col, 10, 30, 130);
            }
        }
    </script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Trasladar</li>
        </ul>
    </nav>
    <aside>
    </aside>
</body>
```

```

<section>
    <article>
        <canvas id="lienzo1" width="600" height="600">Su navegador no
        permite utilizar canvas.
    </article>
</section>
<footer>

</footer>
</body>
</html>

```

Cuando se acabe de cargar la página, se ejecutará la función dibujar() que llamará a la función dibujarTriangulo(), tantas veces como triángulos quepan en el tamaño que le hayamos dado al elemento Canvas.

En esta la función dibujarTriangulo lo primero que hacemos es grabar el estado llamando al método save(). De esta manera, quedarán almacenados los valores actuales del punto de origen. Después movemos el punto de origen llamando a translate. Ahora podemos dibujar el triángulo considerando que la coordenada (0, 0) coincide con los parámetros (x, y). Una vez que hemos pintado el triángulo, recuperamos el estado actual llamando al método restore():

```

function dibujarTriangulo(lienzo, x, y, base, altura) {
    lienzo.save();
    lienzo.translate(x, y);
    lienzo.fillStyle = "rgb(255,0,0)";
    lienzo.beginPath();
    lienzo.moveTo(base / 2, 0);
    lienzo.lineTo(0, altura);
    lienzo.lineTo(base, altura);
    lienzo.lineTo(base / 2, 0);
    lienzo.fill();
    lienzo.restore();
}

```

Cada vez que llamemos desde el for de la función dibujar() a la función dibujarTriangulo() le pasaremos una nueva coordenada “x”, para que empiece a pintar el siguiente triángulo donde acabó el anterior:

```

if (lienzo) {
    for (var col = 0; col < 550; col += 30)
        dibujarTriangulo(lienzo, col, 10, 30, 130);

}

```

5.3.4.7.2. Rotar

La segunda transformación que nos permite realizar el elemento Canvas es la rotación:

```
rotate(grados)
```



Debemos saber que los grados de rotación se indican en radianes y que la rotación será en sentido horario y con respecto al punto de origen (0, 0).

Por lo tanto, lo normal será que usemos primero el método `translate()` y luego lo rotemos.

Veamos un ejemplo en el que vamos a rotar un cuadrado de forma indefinida:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.clearRect(0, 0, 600, 600);
                lienzo.save();
                lienzo.fillStyle = "rgb(255,0,0)";
                lienzo.translate(300, 300);
                lienzo.rotate(avance);
                lienzo.fillRect(-100, -100, 200, 200);
                lienzo.restore();
                avance += 0.05;
                if (avance > Math.PI * 2)
                    avance = 0;
            }
        }

        var avance = 0;
        window.onload = function inicio() {
            setInterval(dibujar, 50);
        }
    </script>
```

```

</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Rotación</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <canvas id="lienzo1" width="600" height="600">Su
navegador no permite utilizar canvas.
        </canvas>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>

```

Lo primero que vamos a hacer es definir una variable global que indicará el grado de rotación actual y una función inicio() que se ejecutará nada más que se cargue la página:

```

var avance = 0;
window.onload = function inicio() {
    setInterval(dibujar, 50);
}

```

Con la función inicio() vamos a definir la ejecución de la función dibujar() cada 50 milisegundos.

En esta función, lo primero que haremos será borrar el lienzo, guardar el estado actual con el método save(), definir el color y después trasladar el punto de origen a la coordenada (300, 300), que será el centro del canvas:

```

lienzo.clearRect(0, 0, 600, 600);
lienzo.save();
lienzo.fillStyle = "rgb(255,0,0)";
lienzo.translate(300, 300);

```

Después definimos los grados de rotación, que inicialmente serán 0 radianes.

Y dibujamos un cuadrado teniendo en cuenta la ubicación actual del punto de origen (por eso los parámetros negativos en x1 e y1):

```

lienzo.rotate(avance);
lienzo.fillRect(-100, -100, 200, 200);

```

Y por último, restauramos el estado inicial con el método `restore()` e incrementamos el ángulo avance en 0,05 radianes. Usamos la formula `Math.PI * 2`, que es lo mismo que decir 360º para que nos reinicie el contador a 0:

```

        lienzo.restore();
        avance += 0.05;
        if (avance > Math.PI * 2)
            avance = 0;
    }
}

```

5.3.4.7.3. Redimensionar

La última transformación que nos permite la librería que trae el Canvas es el redimensionado. La realizaremos llamando al método:

```
scale(x, y);
```

Los valores que pueden tener sus parámetros pueden ser:

- Si le pasamos `x=1` e `y=1`, el gráfico tendrá el mismo tamaño que el original.
- Si es menor a uno, estamos reduciendo.
- Si es mayor a 1, estaremos generando una figura mayor a la original.

Veamos un ejemplo que mostrará una imagen girando y en cada giro le iremos aumentando su tamaño:

```

<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Canvas</title>
    <script type="text/javascript">
        function devolverLienzo(x) {
            var canvas = document.getElementById(x);
            if (canvas && canvas.getContext) {
                var lienzo = canvas.getContext("2d");
                return lienzo;
            }
            else
                return false;
        }

        function dibujar() {
            var lienzo = devolverLienzo("lienzo1");
            if (lienzo) {
                lienzo.clearRect(0, 0, 600, 600);
                lienzo.save();
                lienzo.translate(300, 300);
                lienzo.rotate(avance);
                lienzo.scale(tamx, tamy);
                lienzo.drawImage(img1, -125, -125);
                avance += 0.05;
                tamx += 0.01;
                tamy += 0.01;
            }
        }
    </script>

```

```

        if (avance > Math.PI * 2)
            avance = 0;
        if (tamx >= 10) {
            tamx = 0.01;
            tamy = 0.01;
        }
        lienzo.restore();
    }
}

var avance = 0;
var img1;
var tamx = 0.01;
var tamy = 0.01;
window.onload = function inicio() {
    img1 = new Image();
    img1.src = "../imagenes/logotipo.png";
    img1.onload = function () {
        setInterval(dibujar, 100);
    }
}

</script>
</head>
<body>
    <header>
        <h1>CANVAS</h1>
    </header>
    <nav>
        <ul>
            <li>Redimensionar</li>
        </ul>
    </nav>
    <aside>
        <section>
            <article>
                <canvas id="lienzo1" width="500" height="500">Su navegador no
                permite utilizar canvas.
            </canvas>
        </article>
    </section>
    <footer>
        </footer>
    </aside>
</body>
</html>

```

Declaramos cuatro variables globales que definirán el avance, la imagen y el tamaño inicial de la misma. Una vez que se carga la página definimos la función inicio() que declara un objeto de tipo imagen, que una vez cargado, ejecutará cada 100 milisegundos la función dibujar():

```

var avance = 0;
var img1;
var tamx = 0.01;
var tamy = 0.01;
window.onload = function inicio() {
    img1 = new Image();
    img1.src = "../imagenes/logotipo.png";
    img1.onload = function () {
        setInterval(dibujar, 100);
    }
}

```

}

Esta función es muy similar a la anterior, en la que limpiamos el lienzo, guardamos el estado, trasladamos el eje “x” e “y”, rotamos la imagen, la redimensionamos y la pintamos, pero con la diferencia que incrementamos tanto el avance como el tamaño de la imagen:

```
lienzo.clearRect(0, 0, 600, 600);
lienzo.save();
lienzo.translate(300, 300);
lienzo.rotate(avance);
lienzo.scale(tamx, tamy);
lienzo.drawImage(img1, -125, -125);
avance += 0.05;
tamx += 0.01;
tamy += 0.01;
if (avance > Math.PI * 2)
    avance = 0;
if (tamx >= 10) {
    tamx = 0.01;
    tamy = 0.01;
}
```

Hasta aquí hemos visto los elementos principales de la librería que incorpora el canvas. En los siguientes capítulos descubriremos todo lo relacionado con las nuevas propiedades que incorpora CSS3.

5.4. Introducción a CSS3

CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los continentes y su presentación y es imprescindible para crear páginas web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas:

- Obliga a crear documentos HTML/XHTML bien definidos y con un significado completo (también llamados “documentos semánticos”).
- Mejora la accesibilidad del documento.
- Reduce la complejidad de su mantenimiento.
- Permite visualizar el mismo documento en infinidad de dispositivos diferentes.

Cuando realizamos una web, lo primero que haremos será codificar el XHTML, de esa manera ya tendremos marcados los elementos, y luego utilizaremos el lenguaje CSS para definir el aspecto de cada elemento.

Una vez creados los contenidos, se utiliza el lenguaje CSS para definir el aspecto de cada elemento:

- Tamaño y tipo de letra del texto.
- Separación horizontal y vertical entre elementos.
- Color.
- Posición de cada elemento dentro de la página, etc.

5.5. Prefijos del navegador

Para poder especificar las propiedades de CSS3, como norma general hacen falta los *vendor prefixes* o prefijos del navegador. La razón es que los navegadores web implementan las diferentes propiedades de una forma gradual, y por lo tanto, hay que especificar cada una de las propiedades para los distintos navegadores.

La desventaja de tener que realizarlo así es que:

- Nuestro código CSS parecerá caótico.
- Hemos de especificar varias veces los parámetros de una misma propiedad.
- La ventaja de hacerlo así:
 - Daremos un toque diferente a la misma página web, y de esa manera se verá de forma distinto en los distintos navegadores.

Las extensiones más comunes son:

- **-webkit-** se usan en navegadores que utilizan Webkit (*Chrome* y *Safari*).
- **-moz-** se usa en navegadores basados en Gecko (*Firefox*).
- **-o-** es para el navegador Opera.
- **-ms-** es para el navegador Internet Explorer.

Aunque muchas de las etiquetas y propiedades que a continuación veremos, ya las implementan las versiones más actualizadas de los navegadores más usados, podéis comprobar dicha compatibilidad accediendo a la siguiente página web <http://www.findmebyip.com/litmus/>.

	CSS3 Selectors									
	MAC					WIN				
	FIREFOX	OPERA	CHROME	SAFARI	FIREFOX	OPERA	CHROME	SAFARI	IE	
11	11	11, 62	10	5, 1	11	11, 81	10	5, 1	8	7
Begins with	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
Ends with	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
Matches	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Root	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
nth-child	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
nth-last-child	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
nth-of-type	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
nth-last-of-type	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
last-child	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
first-of-type	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
last-of-type	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
only-child	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
only-of-type	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
empty	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
target	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
enabled	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
disabled	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
checked	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
not	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
General Sibling	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
										98%

5.6. Etiquetas CSS3

En este capítulo descubriremos las etiquetas más importantes y más usadas en las hojas de estilos de CSS3.

Para ello, realizaremos los ejemplos de cada uno de los siguientes capítulos, aplicando los estilos en un archivo aparte de la página web.

Dicho archivo tendrá la extensión .css y deberemos alojarlo en una carpeta dentro del sitio web, que en nuestro caso la llamaremos “Styles”. Dicho archivo lo llamaremos “Estilos.css”, pero cada uno puede usar el nombre que quiera.

Después, para poder utilizar dicha hoja de estilos en nuestra página web, deberemos introducir la siguiente línea de código en nuestra cabecera:

```
<link href="Styles/Estilos.css" rel="stylesheet" />
```

Veamos a continuación las etiquetas.

5.6.1. Bordes

CSS3 nos va a permitir crear varios tipos de bordes en los elementos de la página. El borde más habitual es el de una línea lisa, pero también hay otros tipos de bordes que podemos implementar como bordes redondeados, bordes con imágenes, etc.

En los siguientes capítulos veremos como implementarlos.

5.6.1.1. Bordes redondeados

Con la propiedad **border-radius** de CSS3, para redondear las esquinas ya no hace falta que utilicemos ni imágenes ni JavaScript para conseguir esquinas redondeadas.

En la hoja de estilos deberemos introducir el siguiente código:

```
div.rounded
{
    background-color: #eee;
    line-height: 20px;
    text-align: center;
    padding: 20px;
    border-radius: 15px;
    width: 400px;
    height: 30px;
}
```

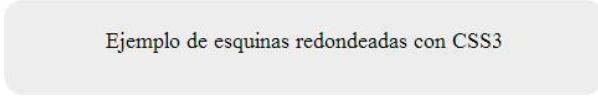
Y en la página HTML introduciremos:

```
<section>
  <article>
    <div class="rounded">
      Ejemplo de esquinas redondeadas con CSS3
    </div>
  </article>
</section>
```

En la hoja de estilos hemos definido los bordes redondeados, predefiniéndolo para los navegadores que aceptan esta etiqueta div. Si no funcionase en alguno de los navegadores que tenéis instalados en el ordenador, podéis probar a insertar estas etiquetas:

```
-moz-border-radius: 15px;
-webkit-border-radius: 15px;
```

Si ejecutamos nuestra página web, veremos el siguiente resultado:



Ejemplo de esquinas redondeadas con CSS3

Figura 5.144. Vista de cuadro con bordes redondeados.

Con el ejemplo anterior hemos definido que todas las esquinas sean redondeadas. Si no queremos que lo sean todas, realizamos lo siguiente:

- Si indicamos 4 valores corresponderán con:

```
border-radius: ArribaIzquierda ArribaDerecha AbajoDerecha
AbajoIzquierda;
```

- Si indicamos 3 valores serán los siguientes:

```
border-radius: ArribaIzquierda AbajoIzquierda+ArribaDerecha
AbajoDerecha;
```

- Si indicamos únicamente 2 valores corresponderán con:

```
border-radius: ArribaIzquierda+AbajoIzquierdaArribaDerecha
+Abajo Derecha;
```

- Si indicamos únicamente 1 solo valor:

```
border-radius: Todos;
```

5.6.1.2. Bordes redondeados de alguno de los vértices

En el caso que queramos redondear solo alguno de los cuatro vértices podemos utilizar alguna de las siguientes propiedades:

- border-top-left-radius: borde superior izquierdo.
- border-top-right-radius: borde superior derecho.
- border-bottom-right-radius: borde inferior derecho.
- border-bottom-left-radius: borde inferior izquierdo.

Veamos un ejemplo donde aparecerán redondeados los vértices superior izquierdo e inferior derecho, en el recuadro superior, y el segundo recuadro muestra redondeado solo el vértice superior derecho. Insertaremos en la hoja de estilos:

```
#recuadro1{
    border-top-left-radius: 20px;
    border-bottom-right-radius: 20px;
    background-color:#ddd;
    width:200px;
    padding:10px;
}
#recuadro2{
    border-top-right-radius: 40px;
    background-color:#aa0;
    width:200px;
    padding:10px;
    margin-top:50px;
}
```

Y para poder verlos en la página, deberemos insertar dos div y asignarles a cada uno la id para que los pinte:

```
<article>
    <div id="recuadro1">
        <h3>Recuadro 1</h3>
        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, </p>
    </div>
    <div id="recuadro2">
        <h3>Recuadro 2</h3>
        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, </p>
    </div>
</article>
```

Si vemos en ejecución la página, veremos cada uno de los estilos que hemos implementado en cada div:

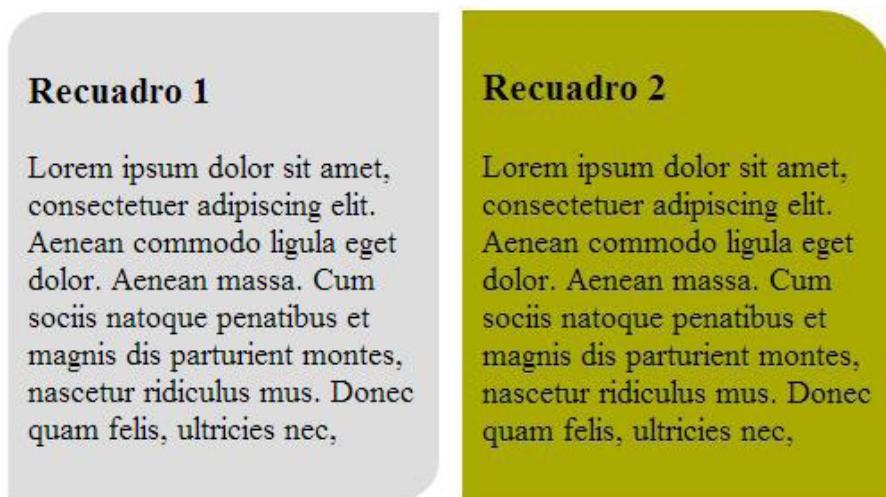


Figura 5.145. Vista de la página con los recuadros con esquinas redondeadas.

5.6.1.3. Bordes con imágenes

Con la propiedad border-image podemos usar imágenes como bordes. Es otra funcionalidad muy interesante para los bordes en CSS. Con esta propiedad puedes definir el uso de una imagen en lugar del borde habitual. Esta funcionalidad está dividida en dos propiedades:

- border-image.
- border-corner-image.

Si sólo queremos especificar uno de los bordes, utilizaremos estas otras etiquetas:

- border-image:
 - border-top-image.
 - border-right-image.
 - border-bottom-image.
 - border-left-image.
- border-corner-image:
 - border-top-left-image.
 - border-top-right-image.
 - border-bottom-left-image.
 - border-bottom-right-image.

La sintaxis a emplear será la siguiente:

```
border-image: url(imagen) %arriba %derecha %abajo %izquierda stretch  
stretch;
```

Expliquemos cada uno de los atributos:

- En imagen habrá que poner la ruta hasta la imagen.
 - Los cuatro siguientes serán los tamaños:
 - Se especificarán en el orden de las agujas del reloj.
 - Se pueden expresar:
 - En porcentajes
 - Sin ninguna unidad de medida para expresar píxeles exactos.
 - La última parte, donde poner stretch, determina como se van a escalar y cortar los bordes:
 - Primer stretch: los lados y superior e inferior de nuestro elemento.
 - Segundo stretch: lados derecho e izquierdo.
- Hay tres posibles valores:
- Stretch (estirar).
 - Repeat (repetir).
 - Round (redondear).
- Si especificamos únicamente un valor, entonces será stretch para todos los lados.
 - El valor por defecto es “stretch”.

Veámoslo con un ejemplo, en el que usaremos una imagen que nosotros hemos llamado borde.gif y sobre un elemento div le pondremos de borde la imagen anterior:

```
div.bordeImage  
{  
    width:300px;  
    height:200px;  
    padding:10px; /* pongo un padding para despegar el texto de los bordes*/  
    border-width: 10px;  
    -webkit-border-image: url(borde.gif) 25% stretch stretch;  
    -moz-border-image: url(borde.gif) 25% stretch stretch;  
}
```

En nuestra página HTML insertaremos la siguiente etiqueta para poder ver el borde sobre el div:

```
<article>
  <div class="bordeImage">
    <p>
      Ejemplo de div al que vamos a aplicar como borde una imagen
    </p>
  </div>
</article>
```

¿Qué hemos hecho con este código?

- En primer lugar hemos definido el grosor del borde en 10px:

`border-width: 10px;`

- Esos 10 píxeles de borde se van a llenar con nuestra imagen borde.gif:

- Para los navegadores que usan webkit (Safari y Chrome):

`-webkit-border-image: url(borde.gif) 25% stretch stretch;`

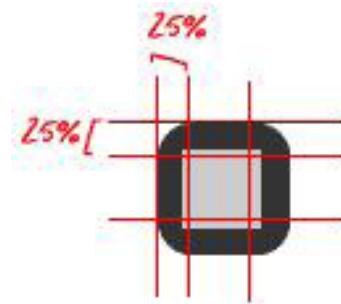
- Para Mozilla (Firefox):

`-moz-border-image: url(borde.gif) 25% stretch stretch;`

- Veamos los parámetros elegidos para border-image:

- `Url(borde.gif)` : indicamos la ruta de la imagen que vamos a usar para crear el borde.

- `25%` : hemos elegido el mismo valor para las cuatro esquinas. Usaremos ese 25% de la imagen para llenar los 10 píxeles de borde de la esquina superior izquierda del DIV, cortando desde la esquina superior izquierda. Y lo mismo para el resto de esquinas:



- `stretch stretch`: alargamos la imagen del borde hasta que se llegue a la siguiente esquina, en todos los lados.

Si ejecutamos veremos el siguiente resultado:

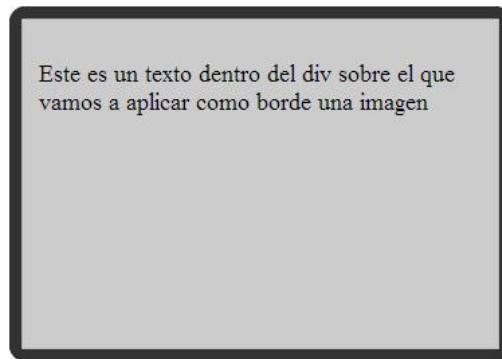


Figura 5.146. Ejecución de la página.

Veamos otro ejemplo para ver mejor como funciona. En este ejemplo usaremos otra imagen para el borde que llamaremos borde2.gif. Si en la hoja de estilos insertamos el siguiente código:

```
div.bordeImage
{
    width: 300px;
    height: 200px;
    padding: 10px;
    border-width: 10px;
    -webkit-border-image: url(borde2.gif) 29% stretch;
    -moz-border-image: url(borde2.gif) 29% stretch;
}
```

El resultado que obtendremos será el siguiente:

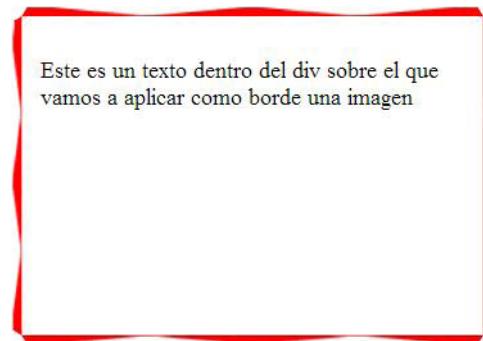


Figura 5.147. Resultado del estilo.

En este ejemplo veréis mejor el efecto de stretch alargando la imagen del borde hasta encontrar la otra esquina. Si lo cambiamos por "round" veremos el siguiente borde:

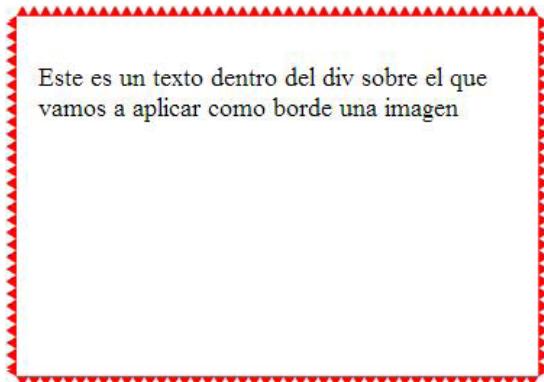


Figura 5.148. Imagen obtenida con un 29%.

Al principio es un poco complicado cogerle el truco, pero haciendo pruebas con los porcentajes y los efectos se pueden lograr cosas interesantes, por ejemplo para crear botones.

5.6.2. Sombras

La propiedad **box-shadow** permite a los diseñadores web el implementar fácilmente sombras tanto dentro como fuera de elementos de caja, especificando valores como el color, tamaño, borrosidad (blur) y offset.

El soporte de los navegadores está aumentando con Mozilla (Firefox), Webkit (Safari y Chrome), Opera e IE9. Aun así, Mozilla y Webkit necesitan todavía sus prefijos -moz- y -webkit- respectivamente (nota: Mozilla Firefox 4.0+ ya no requiere el prefijo -moz-).

```
div.boxshadow
{
    background-color: #eee;
    -moz-box-shadow: 10px 10px 5px #888;
    -webkit-box-shadow: 10px 10px 5px #888;
    box-shadow: 10px 10px 5px #888;
    margin-right: 20px;
    margin-top: 10px;
    padding: 20px;
    text-align: center;
    border-radius: 8px;
    width: 500px;
    height: 20px;
}
```

Expliquemos como funciona esta etiqueta. La propiedad tiene 3 valores y un color, los valores son los siguientes:

- El desplazamiento horizontal de la sombra:
 - Si es positivo significa que la sombra se encuentra a la derecha del objeto.
 - Si es un desplazamiento negativo pondrá la sombra a la izquierda.

- El desplazamiento vertical de la sombra:
 - Si es negativo, la sombra será en la parte superior del objeto.
 - Si es positivo, la sombra estará por debajo.
- El tercer parámetro es el radio de desenfoque:
 - Si se pone a 0, la sombra será fuerte y con color liso
 - Si es más grande el número, más borrosa será.
- El último valor es el color a aplicar a la sombra.

Para introducirlo en nuestra página lo haremos de la siguiente forma:

```
<article>
  <div class="boxshadow">
    Ejemplo de recuadro con bordes redondeados y una sombra gris.
  </div>
</article>
```

Si ejecutamos la página veremos el siguiente resultado:

Ejemplo de recuadro con bordes redondeados y una sombra gris.

Figura 5.149. Vista de recuadro en ejecución.

5.6.2.1. Sombras múltiples

Con la propiedad box-shadow también podemos aplicar múltiples sombras a un objeto. Para esto debemos aplicar la siguiente sintaxis:

```
box-shadow: [desplazamiento en x] [desplazamiento en y] [desenfoque]
           [color] , [desplazamiento en x] [desplazamiento en y] [desenfoque]
           [color] ...
```

Veámoslo con un ejemplo en el que haremos que aparezca una sombra de color rojo en la parte superior izquierda y una sombra verde en la parte inferior derecha:

```
div.boxshadow
{
  background-color: #eee;
  box-shadow: -30px -30px 20px #f00, 30px 30px 20px #0f0;
  border-radius: 8px;
  margin-left: 50px;
  margin-right: 20px;
  margin-top: 50px;
  padding: 20px;
  text-align: center;
  width: 500px;
  height: 20px;
}
```

Si ejecutamos el archivo, veremos el siguiente resultado:

CSS3

- Ejemplo Esquinas Redondeadas y Sombras Multiples

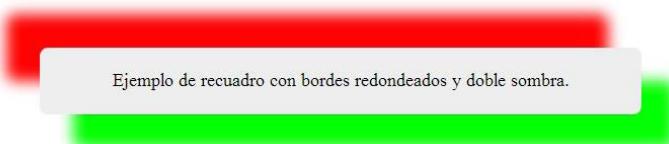


Figura 5.150. Ejecución del ejemplo.

5.6.2.2. Sombras interiores

También podemos implementar la sombra interior al objeto, para esto debemos anteceder a los valores la palabra inset. Si queremos un recuadro con sombra interior de color rojo y sombra exterior de color amarilla, podemos aplicar los siguientes valores:

```
div.boxshadow
{
    background-color: #ddd;
    box-shadow: inset 0px 0px 40px #f00, 0px 0px 40px #ff0;
    border-radius: 20px;
    margin-left: 50px;
    margin-right: 20px;
    margin-top: 50px;
    padding: 20px;
    text-align: center;
    width: 200px;
    padding: 10px;
}
```

Veamos como queda el efecto si ejecutamos:

CSS3

- Ejemplo de recuadro con bordes redondeados y sombra interior roja y exterior amarilla

Ejemplo de recuadro con
bordes redondeados y sombra
interior roja y exterior
amarilla.

Figura 5.151. Ejecución del ejemplo.

5.6.2.3. Sombras con transparencia

Otra posibilidad que nos ofrece la propiedad box-shadow es definir un valor de transparencia de la sombra. Para conseguir este efecto debemos utilizar la función rgba, ejemplo:

```
box-shadow: 30px 30px 30px rgba(255,0,0,0.3);
```

Con esta función debemos indicar el color de la sombra:

- Con los primeros tres parámetros indicamos el color.
- El cuarto parámetro es el índice de transparencia y podemos utilizar cualquier valor entre 1 y 0:
 - 0 indica que es totalmente transparente la sombra.
 - 1 es totalmente opaca.

Hagamos un ejemplo donde realizaremos un recuadro con sombra de color rojo y un índice de transparencia del 50%:

```
div.boxshadow
{
    box-shadow: 30px 30px 30px rgba(255,0,0,0.5);
    border-radius: 20px;
    background-color: #ddd;
    width: 200px;
    padding: 10px;
}
```

Si ejecutamos nuestro ejemplo, veremos el recuadro definido por nuestra función:

Ejemplo de recuadro con sombra de color rojo y un índice de transparencia del 50%

Figura 5.152. Ejemplo transparencia.

5.6.2.4. Sombras de texto

Con la propiedad **text-shadow** podremos definir una sombra a un texto.



Es importante tener en cuenta que esta propiedad no la soporta el Internet Explorer 9.

La sintaxis más común es la siguiente:

```
Elemento {  
    text-shadow: desplazamientoX desplazamientoY radio-de-  
    desenfoque color;  
}
```

La propiedad posee 4 valores:

- El desplazamiento horizontal de la sombra o desplazamientoX:
 - Un desplazamiento negativo pondrá la sombra a la izquierda.
 - Un desplazamiento positivo pondrá la sombra a la derecha.
- El desplazamiento vertical o desplazamientoY:
 - Un valor negativo dispone la sombra en la parte superior del texto.
 - Con un valor positivo la sombra estará por debajo del texto.
- El tercer parámetro es el radio de desenfoque:
 - Si se pone a 0 la sombra, será fuerte y con color liso.
 - Para números más grandes, más borrosa será.
- El último parámetro es el color de la sombra.

Vamos a hacer tres ejemplos diferentes:

1. Que un texto tenga una sombra en la parte inferior y a la derecha, con un pequeño desenfoque de color gris. Aplicaremos la propiedad “#titulo1”
2. En este ejemplo la sombra estará en la parte superior izquierda de cada letra. Aplicaremos la propiedad “#titulo2”
3. En este ejemplo aplicaremos varias sombras al texto con la propiedad “#titulo3”

Los estilos serán los siguientes:

```
h1  
{  
    font-size: 50px;  
}  
body  
{  
    background: white;  
    margin: 20px;  
}  
#titulo1 {  
    text-shadow: 5px 5px 5px #aaa;  
}  
#titulo2 {  
    text-shadow: -5px -5px 5px #aaa;  
}
```

```
#titulo3 {
    text-shadow: 3px 3px 5px #f00,
                 6px 6px 5px #0f0,
                 9px 9px 5px #00f;
}
```

Para implementarlo usaremos la cabecera de nuestra página, donde debemos usar las etiquetas h1, pero con cuidado:

```
<header>
    <h1 id="titulo1">Titulo sombreado</h1>
    <h1 id="titulo2">Titulo sombreado</h1>
    <h1 id="titulo3">Titulo sombreado</h1>
</header>
```

Si ejecutamos, veremos los 3 títulos diferentes con sombra en los textos:



Titulo sombreado

Titulo sombreado

Titulo sombreado

Figura 5.153. Ejecución de la página con los textos sombreados.

5.6.3. Transformaciones 2D

Todas las transformaciones en CSS3 las conseguiremos por medio de la propiedad **transform**, la cual tiene como valor diversos métodos.

```
transform: <método>+;
```

El método actuará como una función interna. Lo escribiremos como si hiciésemos una llamada a una función, tal como hacemos en JavaScript:

```
transform: método(a,b);
```

Donde método será siempre una palabra clave y a, b serán los parámetros o argumentos que le pasaremos al método. En los siguientes capítulos, veremos los distintos métodos de la propiedad transform.

Debéis saber que las transformaciones todavía no están definidas como un estándar en todos los navegadores, por lo que será necesario agregar los prefijos del navegador que la implementa:

```
Elemento {  
    -ms-transform: metodo(valor(es));      /* Internet Explorer */  
    -webkit-transform: metodo(valor(es));   /* WebKit */  
    -moz-transform: metodo (valor(es));     /* Firefox */  
    -o-transform: metodo (valor(es));      /* Opera */  
}
```

En un futuro próximo, la propiedad de transformación 2D definitiva será:

```
Elemento {  
    transform: metodo(valor(es));  
}
```

5.6.3.1. Rotación

La primera función de transformación que vamos a ver será la de rotar un elemento HTML. Esta función se llama rotate y necesita que se le pase un parámetro que será el que indique los grados a rotar. Esta rotación será en sentido horario y si le indicamos un valor negativo, lo hará en sentido contrario a las agujas del reloj.

Por ejemplo, vamos a rotar un recuadro 45 grados en sentido de las agujas del reloj y para que nos funcione en la mayoría de los navegadores deberemos implementar el siguiente código:

```
div.transform  
{  
    -ms-transform: rotate(45deg);  
    -webkit-transform: rotate(45deg);  
    -moz-transform: rotate(45deg);  
    -o-transform: rotate(45deg);  
    transform: rotate(45deg);  
    border-radius: 20px;  
    background-color: #ddd;  
    width: 200px;  
    padding: 10px;  
    margin-left: 50px;  
}
```

Dependiendo del navegador en el que se procese la página, se ejecutará una u otra propiedad:

- Si es el Internet Explorer -ms-transform.
- Si es Safari o Chrome se ejecutará -webkit-transform.
- Para un futuro, hemos agregado la propiedad transform: rotate(45deg) que será la que en un futuro todos los navegadores interpretarán.

El resultado visual es el siguiente:

Recuadro rotado



Figura 5.154. Imagen de la página en ejecución.

Os habréis fijado que el punto de rotación coincide con el centro del recuadro. Y podemos rotar en sentido antihorario indicando el valor del grado con un valor negativo:

```
div.trasformAntihorario{
    -ms-transform: rotate(-45deg);
    -webkit-transform: rotate(-45deg);
    -moz-transform: rotate(-45deg);
    -o-transform: rotate(-45deg);
    transform: rotate(-45deg);
    border-radius: 20px;
    background-color:#ddd;
    width:200px;
    padding:10px;
}
```

Necesitaremos insertar un segundo div en la página para que cargue dicho recuadro y darle el efecto deseado:

```
<article>
    <div class="trasform">
        <p>
            Este es un texto dentro del div sobre el que vamos a aplicar
            una rotación de 45 grados
        </p>
    </div>
    <div class="trasformAntihorario">
        <p>
            Este es un texto dentro del div sobre el que vamos a aplicar
            una rotación de 45 grados antihorario
        </p>
    </div>
</article>
```

Recuadro rotado

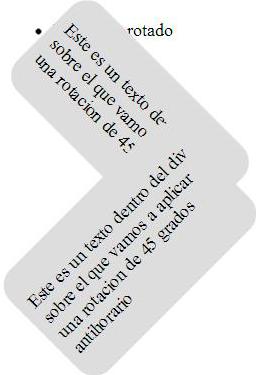


Figura 5.155. Imagen de los cuadros con rotación antihoraria.

Habréis observado que cuando la rotación se ejecuta, no ocupa más espacio el elemento HTML, realmente se solapa con otros elementos de la página.

5.6.3.2. Punto de origen

En el capítulo anterior, ya vimos que cuando rotamos un elemento HTML siempre hay un punto fijo que por defecto es el punto central del recuadro.

Ese punto se puede variar y ubicar en cualquiera de los cuatro vértices del recuadro, haciéndolo con la propiedad transform-origin y cualquiera de las siguientes opciones:

- Left top.
- Right top.
- Left bottom.
- Right bottom.

Imaginad que queremos dejar fijo el vértice superior izquierdo y rotar 10 grados en sentido horario, entonces deberíamos insertar el siguiente código:

```
div.trasform{
    -ms-transform: rotate(10deg);
    -webkit-transform: rotate(10deg);
    -moz-transform: rotate(10deg);
    -o-transform: rotate(10deg);
    transform: rotate(10deg);

    -ms-transform-origin: left top;
    -webkit-transform-origin: left top;
    -moz-transform-origin: left top;
    -o-transform-origin: left top;
    transform-origin: left top;
```

```

border-radius: 20px;
background-color:#ddd;
width:200px;
padding:10px;
margin-left:50px;
}

```

Es como si clavásemos en la esquina superior izquierda y después girásemos en el sentido de las agujas de un reloj 10 grados. Veamos el resultado:

Recuadro rotado

- Recuadro rotado

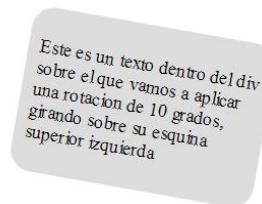


Figura 5.156. Imagen del recuadro rotado.

Con lo que hemos visto hasta ahora, podemos poner el punto de origen en cinco lugares, el centro y las cuatro esquinas, pero también podemos trasladar a cualquier punto dentro del div creado, el punto del origen con la siguiente sintaxis:

```
transform-origin: 0% 50%;
```

En este código, el primer valor representa las “x” y el segundo representa las “y”. Si queremos poner el punto de rotación en la esquina superior derecha, podemos hacerlo de dos formas:

- Como ya lo habíamos hecho: `transform-origin: left top`.
- Mediante porcentajes: `transform-origin: 100% 0%`.

Después, mediante porcentajes podemos en “x” e “y” mover el punto de origen a cualquier parte dentro del div.

Veamos como poner el punto de origen de la rotación en la mitad del lado izquierdo:

```

div.trasform
{
  -ms-transform: rotate(10deg);
  -webkit-transform: rotate(10deg);
  -moz-transform: rotate(10deg);
  -o-transform: rotate(10deg);
  transform: rotate(10deg);
  -ms-transform-origin: 0% 50%;
  -webkit-transform-origin: 0% 50%;
  -moz-transform-origin: 0% 50%;
  -o-transform-origin: 0% 50%;
}

```

```

    transform-origin: 0% 50%;
    border-radius: 20px;
    background-color: #ddd;
    width: 200px;
    padding: 10px;
    margin-top: 50px;
}

```

Si ejecutamos, observamos como el punto de rotación lo realiza sobre la mitad del recuadro en el lado izquierdo:

Recuadro rotado

- Recuadro rotado

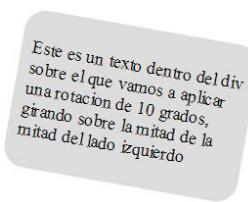


Figura 5.157. Rotación sobre la mitad del lado izquierdo.

5.6.3.3. Escalado

Mediante CSS3 podemos agrandar o reducir el tamaño del elemento, utilizando la función **transform:scale**. La sintaxis que usaremos es la siguiente:

```

Elemento {
    transform: scale(valorx ,valory);
}

```

- Valor x indica la escala para x:
 - Valor 1 el elemento queda como está.
 - Valores mayores de 1, lo haremos más grande.
 - Valores menores de 1 lo haremos más pequeño.
- Valor y es para la escala en y: funciona de la misma manera que valorx.

Si por ejemplo queremos que sea dos veces su ancho y la mitad de alto, pondremos lo siguiente:

```
transform: scale(2 ,0.5);
```

Si lo queremos un 20% más pequeño de ancho y 20% más de alto, definiremos lo siguiente:

```
transform: scale(0.8 , 1.2);
```

Ahora vamos a implementarlo en un ejemplo. Para ello definiremos también el punto de origen en la esquina superior izquierda. Para hacerlo correctamente, debemos definir todos los prefijos para que funcione en la mayoría de los navegadores:

```
div.trasform
{
    /*definimos el escalado*/
    -ms-transform: scale(0.8, 1.2);
    -webkit-transform: scale(0.8, 1.2);
    -moz-transform: scale(0.8, 1.2);
    -o-transform: scale(0.8, 1.2);
    transform: scale(0.8, 1.2);

    /*definimos el punto de origen*/
    -ms-transform-origin: left top;
    -webkit-transform-origin: left top;
    -moz-transform-origin: left top;
    -o-transform-origin: left top;
    transform-origin: left top;

    /*definimos otras propiedades como el color, bordes, ancho...*/
    border-radius: 20px;
    background-color: #ddd;
    width: 200px;
    padding: 10px;
}
```

Si ejecutamos nuestro script, veremos la imagen deformada, con los nuevos porcentajes de su tamaño:

Recuadro escalado

- Recuadro escalado

Este es un texto dentro del div sobre el que vamos a aplicar un escalado

Figura 5.158. Imagen del recuadro con las nuevas dimensiones.

También tenemos la posibilidad de escalar la imagen eligiendo o la altura o la anchura, usando únicamente la función scaleX o scaleY:

```
transform: scaleX(2);
transform: scaleY(0.5);
```

Si usamos un valor negativo, en cualquiera de los dos valores conseguiremos una reflexión del elemento, es decir, un efecto espejo. Lo realizaremos en el valor:

```
div.trasform
{
    -ms-transform: scale(1.20,-1);
    -webkit-transform: scale(1.20,-1);
    -moz-transform: scale(1.20,-1);
    -o-transform: scale(1.20,-1);
    transform: scale(1.20,-1);

    border-radius: 20px 40px 40px 20px;
    background-color: #aa0;
    width: 200px;
    padding: 10px;
    margin-top: 10px;
    margin-left: 50px;
}
```

Cuya imagen será cuando ejecutemos:

Recuadro escalado

- Recuadro escalado (Reflexion)

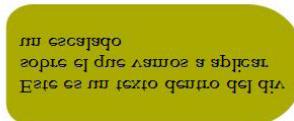


Figura 5.159. Vista del recuadro en reflexión.

5.6.3.4. Traslación

Con el método **transform:translate** cambiaremos de posición el elemento moviéndolo hacia donde indican los parámetros:

```
transform: translate(valorx, valory);
```

Dentro del paréntesis pondremos dos parámetros:

- Valorx es la traslación que se produce en horizontal, hacia la derecha:
 - Si el valor es positivo se desplazará hacia la derecha.
 - Si su valor es negativo el elemento se desplazara hacia la izquierda.
- Valory es la traslación en vertical, hacia abajo.
 - Si el valor es positivo, se desplazará hacia abajo.
 - Si su valor es negativo, el elemento se desplazara hacia arriba.

Veamos el siguiente ejemplo, le aplicamos a un elemento el siguiente código:

```

div.trasform
{
    transform: translate(150px,20px);
    -moz-transform: translate(150px,20px);
    -webkit-transform: translate(150px,20px);
    -o-transform: translate(150px,20px);
    -ms-transform: translate(150px,20px);

    border-radius: 20px 20px 20px 20px;
    background-color: #aa0;
    width: 200px;
    padding: 10px;
    margin-top: 10px;
    margin-left: 50px;
}

```

Veamos el resultado, en la primera imagen vemos el div original, y en la segunda imagen vemos la traslación aplicada:

Recuadro trasladado Recuadro trasladado

- Recuadro trasladado (translate)
- Recuadro trasladado (translate)

Esto es un texto dentro de div
sobre el que vamos a aplicar
una translación

Esto es un texto dentro de div
sobre el que vamos a aplicar
una translación

Figura 5.160. Imágenes antes y después de aplicar las translación.

También disponemos de los métodos `translateX` y `translateY`, que desplazan el elemento de su posición original. Estos métodos funcionan igual que el método `translate` pero sólo en una dirección, por eso sólo tienen un parámetro:

- Para desplazar el elemento a lo largo del eje X:

```
transform: translateX(valor);
```

- Para desplazar el elemento a lo largo del eje Y

```
transform: translateY(valor);
```

5.6.3.5. Torcer

Usando el método `transform:skew` podremos torcer el elemento HTML en los ejes “x” e “y”, la sintaxis es la siguiente:

```
transform: skew(gradosx ,gradosy);
```

Veamos por ejemplo como torcer en “x” 15 grados un recuadro:

```
div.transform
{
    transform: skew(15deg,0deg);
    -ms-transform: skew(15deg,0deg);
    -webkit-transform: skew(15deg,0deg);
    -moz-transform: skew(15deg,0deg);
    -o-transform: skew(15deg,0deg);

    border-radius: 20px;
    background-color: #ddd;
    width: 200px;
    padding: 10px;
    margin-left:10px;
}
```

Si queremos torcerlo en el eje "y" deberemos cambiar los comandos por:

```
-ms-transform: skew(0deg,15deg);
-webkit-transform: skew(0deg,15deg);
-moz-transform: skew(0deg,15deg);
-o-transform: skew(0deg,15deg);
transform: skew(0deg,15deg);
```

Si aplicamos el giro en ambos ejes, escribiríremos los siguientes comandos:

```
-ms-transform: skew(15deg,15deg);
-webkit-transform: skew(15deg,15deg);
-moz-transform: skew(15deg,15deg);
-o-transform: skew(15deg,15deg);
transform: skew(15deg,15deg);
```

Si vemos los 3 ejemplos en ejecución veremos las diferencias entre ellos:

Recuadro torcido Recuadro torcido Recuadro torcido

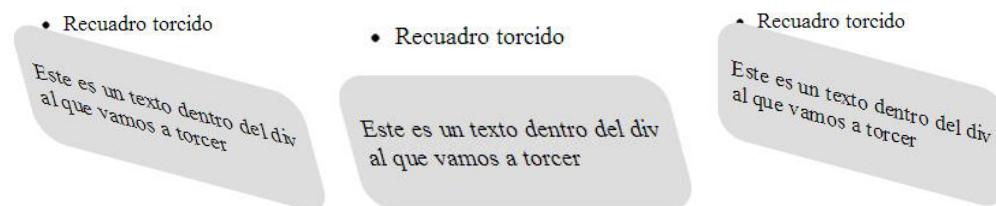


Figura 5.161. Diferencias entre aplicar diferentes tipos de skew.

5.6.3.6. Múltiples transformaciones

Con la sintaxis de CSS3 podemos definir en la propiedad transform múltiples transformaciones en forma simultánea.



Debemos dejar al menos un espacio en blanco entre cada llamada a una función de transformación.

Su sintaxis será la siguiente:

```
transform: rotate(grados) scale(valorx, valory) translate(valorx,
                                                       valory) skew(gradosx, gradosy);
```

Por ejemplo apliquemos múltiples transformaciones a un recuadro:

```
div.trasform
{
    -moz-transform: rotate(15deg) scale(0.9,0.6) translate(10px,10px)
    skew(5deg,0deg);
    -ms-transform: rotate(15deg) scale(0.9,0.6) translate(10px,10px)
    skew(5deg,0deg);
    -webkit-transform: rotate(15deg) scale(0.9,0.6) translate(10px,10px)
    skew(5deg,0deg);
    -o-transform: rotate(15deg) scale(0.9,0.6) translate(10px,10px)
    skew(5deg,0deg);
    transform: rotate(15deg) scale(0.9,0.6) translate(10px,10px)
    skew(5deg,0deg);
    border-radius: 20px;
    background-color: #ddd;
    width: 200px;
    padding: 10px;
    margin-left: 10px;
}
```

En este caso hemos:

- Rotado un recuadro 15 grados en sentido de las agujas del reloj.
- Lo hemos escalado un 10% más pequeño de ancho y un 40% más pequeño de alto.
- Lo hemos trasladado 10px en los ejes “x” e “y”.
- Lo hemos torcido 5 grados en el eje “x”.

Si ejecutamos la página, veremos el siguiente resultado:

Transformaciones 2D

- Múltiples Transformaciones

Este es un texto dentro del div al que vamos a aplicar múltiples transformaciones

Figura 5.162. Múltiples transformaciones a un div.

5.6.4. Opacidad (opacity)

Debemos saber que los elementos HTML poseen la propiedad opacity para definir cuál es su opacidad.



La opacidad es una característica de los objetos de no dejar pasar la luz. Cuanto más opaco es un objeto, menos deja pasar la luz.

La sintaxis para definir la opacidad es la siguiente:

```
opacity: valor;
```

El método será opacity y su valor estará comprendido entre 0 y 1.

- 0 será totalmente transparente, y aunque luego no se verá nada en pantalla, ese espacio ocupado por el elemento HTML quedará reservado.
- 1 significa que es totalmente opaco y no dejará pasar la luz.



Debemos tener en cuenta que cuando asignemos una opacidad a un elemento HTML, todos los elementos que estén contenidos en dicho elemento, heredaran también dicha opacidad.

Veamos varios ejemplos de opacidad, en el que en un recuadro vamos a insertar una imagen y un texto:

```
div.opacidad
{
    background-image: url("logotipo.jpg");
    opacity: 0.3;
    color: #f00;
    width: 500px;
    height: 250px;
```

```

border-radius: 15px;
font-size: 30px;
}

```

Si ejecutamos nuestra página, veremos que la opacidad de la imagen, también la hereda el texto insertado:

Opacidad

- Opacidad al 0.3



Figura 5.163. Opacidad del 0.3.

Si modificamos únicamente la opacidad al 0.6, `opacity: 0.6;` obtendremos el siguiente resultado:

Opacidad

- Opacidad al 0.6



Figura 5.164. Opacidad del 0.6.

Y si por último, modificamos únicamente la opacidad al 0.9, `opacity: 0.9;` obtendremos el siguiente resultado, casi opaco sin dejar pasar casi la luz:

Opacidad

- Opacidad al 0.9



Figura 5.165. Opacidad del 0.9.

5.6.5. Opacidad (color)

Con CSS3 además de poner elementos semitransparentes, también podemos poner la transparencia en el color. Cada color y cada elemento pueden adquirir un grado de

transparencia que va desde el color sólido al transparente total. La forma de definirlo es la siguiente:

```
color: rgba(rojo, verde, azul, opacidad);
```

Para definir un color indicaremos cuatro valores:

- Los tres primeros son valores enteros entre 0 y 255, que indican la cantidad de rojo, verde y azul.
- El cuarto valor es un número entre 0 y 1 que indica la opacidad que se aplica al color.
 - 1 se pinta el color totalmente opaco (es el valor por defecto).
 - Valores menores de 1 hará más transparente el gráfico.

Veamos un ejemplo, en el que pondremos un texto dentro de un recuadro que tiene una imagen, en la que utilizaremos la función `rgba` para definir la opacidad del texto:

```
div.opacidad
{
    background-image: url("logotipo.jpg");
    color: #f00;
    width: 700px;
    height: 450px;
    border-radius: 15px;
    font-size: 45px;
    margin-top: 10px;
}
.texto
{
    color: rgba(0,0,255,0.3);
}
```

Y en el archivo HTML definiremos los div:

```
<article>
    <div class="opacidad">
        <p class="texto">
            Este es un texto dentro del div al que vamos a aplicar
            múltiples opacidades
        </p>
    </div>
</article>
```

Si ejecutamos la página, veremos como reducimos la opacidad al texto, pero no a la imagen:

Opacidad

• Opacidad al 0.3



Figura 5.166. Cambio de opacidad en el texto.

5.6.6. Múltiples columnas

Otra de las novedades de CSS3 es poder distribuir contenido, ya sea texto, imágenes, etc., de un elemento en varias columnas. No se crearán nuevas cajas para las columnas, aunque el texto y demás elementos de la caja aparezcan distribuidos en columnas. Por lo tanto, las propiedades que se apliquen a la caja, se aplicarán a todas las columnas.

Cuando se crean varias columnas en una caja, todas tendrán la misma anchura, por lo que el espacio se distribuirá de manera uniforme entre las diferentes columnas. El texto también intentará distribuirse de manera uniforme en altura, pero a veces no es posible, por ejemplo, si se incluyen imágenes. En este caso la altura será la de la columna más alta, y dejará en las demás, al final de las mismas, un espacio en blanco para completarlas.

Veamos a continuación las propiedades para distribuir en columnas.

5.6.6.1. Número de columnas

Con esta propiedad, colocaremos el bloque de texto en múltiples columnas con la única indicación de la cantidad de columnas que queremos que lo divida:

```
column-count: cantidadColumnas;
```

La propiedad column-count distribuye el contenido en el número de columnas indicado. Su valor debe ser siempre un número positivo entero distinto de 0.

Veámoslo con un ejemplo. Generamos un cuadro con tres columnas. Insertaremos lo siguiente en nuestra hoja de estilos:

```
div.columnas
{
    -moz-column-count: 4;
    -webkit-column-count: 4;
    column-count: 4;
    border-radius: 20px;
    background-color: #ddd;
    width: 600px;
    padding: 10px;
}
```

Y el ejemplo de página HTML será el siguiente:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>CSS3</title>
    <link href="Styles/Estilos.css" rel="stylesheet" />
</head>

<body>
    <header>
        <h1>Columnas Multiples</h1>
    </header>
    <nav>
        <ul>
            <li>Cuadro con 4 columnas</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <div class="columnas">
                <p>Este es el texto que está contenido en un div,  

                al cual le vamos a asignar un número determinado de columnas.  

                De esta manera comprobaremos el resultado de la propiedad  

                column-count.</p>
                <p>Este es el texto que está contenido en un div,  

                al cual le vamos a asignar un número determinado de columnas.  

                De esta manera comprobaremos el resultado de la propiedad  

                column-count.</p>
                <p>Este es el texto que está contenido en un div,  

                al cual le vamos a asignar un número determinado de columnas.  

                De esta manera comprobaremos el resultado de la propiedad  

                column-count.</p>
                <p>Este es el texto que está contenido en un div,  

                al cual le vamos a asignar un número determinado de columnas.  

                De esta manera comprobaremos el resultado de la propiedad  

                column-count.</p>
            </div>
        </article>
    </section>
    <footer>
    </footer>
</body>
```

```
</html>
```

Si ejecutamos el ejemplo anterior, veremos el texto organizado en 4 columnas:

Columnas Multiples

- Cuadro con 4 columnas

Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas. De esta manera comprobaremos el resultado de la propiedad column-count.	a asignar un numero determinado de columnas. De esta manera comprobaremos el resultado de la propiedad column-count.	manera comprobaremos el resultado de la propiedad column-count.	propiedad column-count.
Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas. De esta manera comprobaremos el resultado de la propiedad column-count.	Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas. De esta manera comprobaremos el resultado de la propiedad column-count.	Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas. De esta manera comprobaremos el resultado de la propiedad column-count.	Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas. De esta manera comprobaremos el resultado de la propiedad column-count.
Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas. De esta manera comprobaremos el resultado de la propiedad column-count.			

Figura 5.167. Imagen del recuadro con 4 columnas.

5.6.6.2. Columnas dinámicas

Otra de las propiedades que nos permite generar múltiples columnas es column-width. Con esta propiedad, si modificamos el tamaño de la ventana del navegador su contenido se relocaliza.

Especificaremos el ancho de la columna en píxeles, porcentaje, etc. y luego se generan tantas columnas como entren en el contenedor.

Su sintaxis es la siguiente:

```
column-width: ancho;
```

Veamos como crear un recuadro con texto y que cada columna ocupe 150 píxeles. Insertamos en nuestra hoja de estilos lo siguiente:

```
div.columns
{
    -moz-column-width: 200px;
    -webkit-column-width: 200px;
    column-width: 200px;
    border-radius: 20px;
    background-color: #ddd;
    padding: 10px;
}
```

Usaremos el mismo ejemplo anterior de página HTML para ejecutarlo:

Columnas Multiples

- Cuadro con columnas de 300px

Este es el texto que esta contenido en un div, al cual le vamos a asignar un ancho determinado a la columna. De esta manera comprobaremos el resultado de la propiedad column-width.

Este es el texto que esta contenido en un div, al cual le vamos a asignar un ancho determinado a la columna. De esta manera comprobaremos el resultado de la propiedad

column-width.

Este es el texto que esta contenido en un div, al cual le vamos a asignar un ancho determinado a la columna. De esta manera comprobaremos el resultado de la propiedad column-width.

Este es el texto que esta contenido en un div, al cual le vamos a asignar un ancho determinado a la columna. De

esta manera comprobaremos el resultado de la propiedad column-width.

Este es el texto que esta contenido en un div, al cual le vamos a asignar un ancho determinado a la columna. De esta manera comprobaremos el resultado de la propiedad column-width.

Figura 5.168. Imagen de nuestra página con columnas de ancho 300px.

El resultado variará dependiendo las dimensiones de la ventana del navegador, dependiendo de si hemos creado o no una página web fluida, es decir, sin un ancho fijo. Podemos mejorar la disposición del texto dentro de las columnas inicializando la propiedad text-indent y margin y asignando justificado a la propiedad text-align. Añadimos a nuestro código anterior las siguientes líneas:

```
text-indent: 1em;
margin: 0;
text-align: justify;
```

Vemos como alinea el texto en formato justificado:

Columnas Multiples

- Cuadro con columnas de 300px

Este es el texto que esta contenido en un div, al cual le vamos a asignar un ancho determinado a la columna. De esta manera comprobaremos el resultado de la propiedad column-width.

Este es el texto que esta contenido en un div, al cual le vamos a asignar un ancho determinado a la columna. De esta manera comprobaremos el resultado de la propiedad

column-width.

Este es el texto que esta contenido en un div, al cual le vamos a asignar un ancho determinado a la columna. De esta manera comprobaremos el resultado de la propiedad

column-width.

Este es el texto que esta contenido en un div, al cual le vamos a asignar un ancho determinado a la columna. De esta manera comprobaremos el resultado de la propiedad

column-width.

Figura 5.169. Texto justificado en cada una de las columnas.

5.6.6.3. Separación entre columnas

Con la propiedad column-gap podemos especificar la separación entre las columnas. Su sintaxis es:

```
column-gap: valor;
```

Para indicar la separación entre columnas utilizaremos cualquier unidad, ya sea en “px”, o en “em” etc.). Por ejemplo, vamos a ver el ejemplo anterior con una separación de 40px entre las columnas. Insertaremos el siguiente código en nuestra hoja de estilos:

```
div.columns
{
    -moz-column-count: 3;
    -webkit-column-count: 3;
    column-count: 3;

    -moz-column-gap: 40px;
```

```

-webkit-column-gap: 40px;
column-gap: 40px;

border-radius: 20px;
background-color: #ddd;
padding: 10px;
width: 600px;

text-indent: 1em;
margin: 0;
text-align: justify;

}

```

Si ejecutamos, veremos como quedan las 3 columnas y una separación de 40px entre ellas:

Columnas Multiples

- Cuadro con 3 columnas y separación de 40px

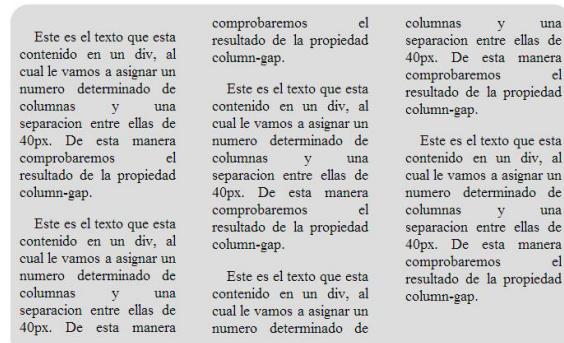


Figura 5.170. Resultado del código anterior.

5.6.6.4. Línea separadora de columnas

A parte de definir la separación entre columnas, podemos configurar una línea separadora entre ellas, usando la propiedad column-rule. La sintaxis es la siguiente:

```
column-rule: grosor estilo color;
```

Veamos un ejemplo, en el que asignamos distintos valores a esta propiedad. Definimos una línea de un píxel, con trazo sólido de color rojo:

```

div.columns
{
    /*definimos número de columnas*/
    -moz-column-count: 3;
    -webkit-column-count: 3;
    column-count: 3;
    /*definimos la linea de separación*/
    -moz-column-rule: 1px solid #f00;
    -webkit-column-rule: 1px solid #f00;
    column-rule: 1px solid #f00;
}

```

```

border-radius: 20px;
background-color: #ddd;
width: 600px;
padding: 10px;
}

```

Veamos como quedan:

Columnas Multiples

- Cuadro con 3 columnas y separación de linea entre columnas

Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas y una separacion entre ellas de lineas diferentes. De esta manera comprobaremos el resultado de la propiedad column-rule.	Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas y una separacion entre ellas de lineas diferentes. De esta manera comprobaremos el resultado de la propiedad column-rule.	Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas y una separacion entre ellas de lineas diferentes. De esta manera comprobaremos el resultado de la propiedad column-rule.
---	---	---

Figura 5.171. Línea continua de separación roja.

Si modificamos únicamente las líneas donde definimos el tipo de línea e introducimos el siguiente código:

```

/*definimos la linea de separación*/
-moz-column-rule: 3px dotted #f00;
-webkit-column-rule: 3px dotted #f00;
column-rule: 3px dotted #f00;

```

Veremos una línea discontinua de color rojo:

Columnas Multiples

- Cuadro con 3 columnas y separación de linea entre columnas

Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas y una separacion entre ellas de lineas diferentes. De esta manera comprobaremos el resultado de la propiedad column-rule.	Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas y una separacion entre ellas de lineas diferentes. De esta manera comprobaremos el resultado de la propiedad column-rule.	Este es el texto que esta contenido en un div, al cual le vamos a asignar un numero determinado de columnas y una separacion entre ellas de lineas diferentes. De esta manera comprobaremos el resultado de la propiedad column-rule.
---	---	---

Figura 5.172. Línea discontinua de separación roja.

Los diferentes valores que pueden tener el estilo son los mismos que para el borde en la propiedad border-style:

```
none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset;
```

De igual forma que en anteriores propiedades, se pueden asignar de manera independiente el grosor, estilo y color del separador:

```

column-rule-width: grosor;
column-rule-style: estilo;
column-rule-color: color;

```



Recordar siempre insertar los prefijos admitidos por los navegadores: -moz-, -webkit-, -o- y -ms-

5.6.7. Importar fuentes

Mediante la propiedad font-face podemos importar fuentes no disponibles en el navegador que se descargará de un servidor web o las incluiremos en nuestro sitio para que se descargue cuando se cargue la página.

Ahora mismo existe un problema, y es que existen varios formatos de fuentes específicos de cada navegador:

- El IE8 e inferiores solo admite el formato *EOT* que es un formato propietario.
- Las fuentes *True Type* es soportado por los navegadores más actualizados.
- Otros formatos son el *OpenType*, *SVG* y *WOFF*.

Por este motivo, debemos intentar especificar la mayor cantidad de formatos de fuentes posibles para nuestro sitio web.

Vamos a ver como importar una fuente para que después pueda ser utilizada por una página web, mediante la regla @font-face, en la cual,

- Primero indicaremos el nombre de la fuente a importar.
- Segundo indicaremos la dirección web de donde descargarla o la dirección donde se encuentra en nuestro sitio web.

La regla @font-face para realizarlo será de la siguiente manera:

```
@font-face
{
    font-family: <nombre>;
    scr: url(<ruta>);
    font-weight: tamaño fuente;
    font-style: estilo fuente;
}
```

- `@font-face` no es una propiedad, es una regla completa en la que `@font-face` actúa como selector. Entre los corchetes incluiremos:

- `font-family: <nombre>;` Damos un nombre al nuevo tipo de letra, elegiremos cualquier nombre y este será el nombre de la fuente.
- `scr: url(<ruta>);` Indicamos la ruta del archivo donde guardamos la fuente.
- `font-weight` y `font-style`:

- Estas propiedades son opcionales
- sus posibles valores son para indicar el grosor y estilo de letra.

Veamos como incluir por ejemplo una fuente de Windows. Hemos buscado un archivo .ttf en nuestro equipo y hemos seleccionado una al azar, y la hemos copiado en el directorio nuevo llamado /fuentes. Hemos elegido la fuente “tahomabd.ttf”.

Para incluirla en la página escribiremos en la hoja de estilos lo siguiente:

```
@font-face
{
    font-family: mifuente;
    src: url(fuentes/DroidSans-Bold.ttf);}
```

Una vez definida, podemos utilizar el nombre de fuente como valor para la propiedad font-family:

```
p
{
    font-family: mifuente;
}
```

Debemos tener en cuenta lo siguiente:

- El nuevo tipo de letra se verá en todos los navegadores, excepto en IE, ya que sólo reconoce los archivos de fuente con la extensión “.eot”;
- Los iPhone y los iPad necesitan los archivos de fuentes con extensión: “.svg”.

Como la mayoría de los archivos de fuentes tienen la extensión .ttf o .otf, debemos convertir el archivo de fuente a los formatos eot y svg.

Para convertirlos, tenemos herramientas en internet que nos facilitan esta tarea. Por ejemplo, en la página <http://www.fontsquirrel.com/fontface/generator>, disponemos de dicha herramienta. Cuando accedamos a la página, veremos el siguiente cuadro:

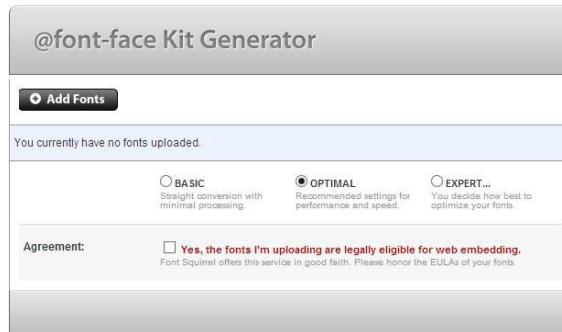


Figura 5.173. Cuadro para añadir las fuentes.

Cuando pulsemos en el botón “Add Fonts” nos saldrá un cuadro de búsqueda, en el que seleccionaremos el archivo de fuente en nuestro ordenador, y lo subimos a la página. Si en la página no aparece el botón “add fonts” prueba a abrir la página con otro navegador, ya que en Firefox y Opera posiblemente no se vea el botón.

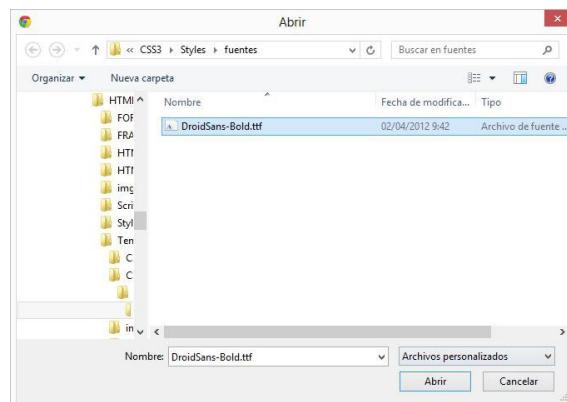


Figura 5.174. Seleccionamos la fuente.

Una vez mandado marcamos el botón “Agreement:”, y en la parte de debajo del cuadro saldrá un botón de descarga. Pulsamos en “Download Your Kit” y descargamos y guardamos el archivo:



Figura 5.175. Pulsamos en el botón descargar.

Se descargará un archivo en formato .zip; Lo descomprimimos y vemos que consta de varios archivos con fuentes en distintos formatos, y aparte, contiene un archivo CSS con el código de la regla @font-face para que se vean las fuentes en todos los navegadores.

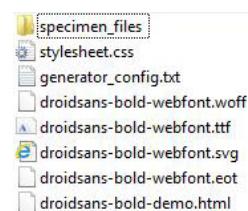


Figura 5.176. Contenido del archivo descargado.

Coparemos los archivos de fuentes con extensión .ttf, .eot y .svg, en nuestro directorio de fuentes, y después añadiremos el siguiente código a nuestro archivo CSS.

```

@font-face
{
    font-family: mifuente;
    src: url(fuentes/droidsans-bold-webfont.ttf);
}

@font-face
{
    font-family: mifuenteIE;
    src: url(fuentes/droidsans-bold-webfont.eot);
}

@font-face
{
    font-family: mifuenteIphone;
    src: url(fuentes/droidsans-bold-webfont.svg);
}

#capa
{
    font: normal 1em mifuente, mifuenteIE, mifuenteIphone;
    width: 350px;
    border: 1px solid blue;
    padding:10px;
}

```

Y para cargar en la página web introducimos el siguiente código:

```

<article>
    <div>
        Este texto tiene el estilo normal de la página web
        <p id="capa">Este texto tiene aplicada la nueva fuente</p>
    </div>
</article>

```

Si ejecutamos la página, veremos como queda la nueva fuente importada:

Importar fuentes

- Aplicamos la fuente nueva

Este texto tiene el estilo normal de la pagina web

Este texto tiene aplicada la nueva fuente

Figura 5.177. Ejecución de la página.

5.6.8. Tratamiento de Imágenes

En los siguientes capítulos, vamos a ver como tratar e importar las imágenes desde CSS3 y su hoja de estilos. Veremos como colocar varias imágenes, como cambiar su tamaño, definir su posición, etc.

5.6.8.1. Imágenes de fondo

Con las versiones anteriores de CSS, si queríamos poner varias imágenes de fondo en un sólo espacio la única solución era hacer varias capas o divs superpuestos en los que ponemos una imagen en cada uno de ellos.

Ahora, con CSS3, no necesitamos crear varias capas, ya que podemos poner varias imágenes de fondo en una sola capa. La sintaxis para insertar varias imágenes será:

```
background-image: url("imagen1"), url("imagen2"), ...;
```

Por lo tanto el código HTML será el siguiente:

```
<article>
  <div id="capa1">
    <p>
      Esta área de la página tiene dos imágenes de fondo distintas
    </p>
  </div>
</article>
```

Con la propiedad background:

- Podremos poner distintas imágenes separadas por comas.
- El orden va de externa a la interna, la primera imagen estará por encima de todas y la última debajo de todas.

En la hoja de estilos insertaremos el siguiente código:

```
#capa1
{
  width: 700px;
  height: 200px;
  background-image: url(logotipo.png) , url(seas.png);
  background-repeat: no-repeat;
}
```

Hemos elegido dos imágenes, una más grande que la otra, para que la grande este en el fondo, por lo tanto, será la última, y una más pequeña, que pondremos la primera en nuestro código. Hemos inicializado la propiedad background-repeat con el valor no-repeat, para que no se repita la imagen por toda el recuadro.

Si ejecutamos la página, el resultado será:

Imagenes

- Varias Imagenes de fondo



Figura 5.178. Imagen logotipo.png sobre la imagen de seas.png.

5.6.8.2. Imágenes: definir posición

Con la propiedad background-position podemos indicar la posición de la imagen con respecto al contenedor.

La sintaxis es la siguiente:

```
Elemento {
    background-image: url("imagen1"), url("imagen2", ...);
    background-position: posx posy , posx posy;
}
```

Si os fijáis los valores que indicamos con posx y posy, coinciden con la posición de las imágenes que especificamos en la propiedad background-image.

Vamos a realizar un ejemplo en el que mostraremos la imagen anterior, que tiene un tamaño de 712x188 y dentro de esta la imagen del logotipo, y lo ubicaremos en los cuatro vértices de la imagen más grande.

Definimos en la hoja de estilos el siguiente código:

```
#recuadro1{
    background-image: url("logotipo.png"),
                    url("logotipo.png"),
                    url("logotipo.png"),
                    url("logotipo.png"),
                    url("seas.png");
    background-position: 0% 0%,
                        100% 0%,
                        0% 100%,
                        100% 100%;
    background-repeat: no-repeat;
    width:721px;
    height:188px;
}
```

Recordar que la primera imagen que se muestra, es la que especificamos la última.

Después en la propiedad background-position indicaremos en el mismo orden la ubicación de cada imagen.

- Las cuatro primeras imágenes definimos las posiciones donde queremos que aparezcan.
- La quinta imagen le indicaremos la posición ya que el ancho y alto de la imagen coincide con los valores asignados a las propiedades width y height.

Veamos en ejecución nuestra página:

Imagenes

- Varias Imagenes de fondo definiendo su posicion y tamaño

Esta área de la página tiene tres imágenes de fondo distintas



Figura 5.179. Repetimos la imagen logotipo 4 veces sobre la imagen principal.

5.6.8.3. Imágenes escaladas

Con la propiedad background-size podremos escalar una imagen insertada como fondo de un elemento. Su sintaxis es:

```
background-size: ancho alto;
```

Las longitudes se pueden especificar en píxeles, porcentajes, etc.

Veamos un ejemplo, en el que usaremos las anteriores imágenes, pero colocando el logotipo en la parte derecha de la pantalla. Este último tiene unas dimensiones de 70x106, y en la página multiplicaremos su tamaño por 1,5 veces.

```
#re cuadro1
{
    background-image: url("logotipo.png"), url("seas.png");
    background-position: 100% 0%, 50% 50%;
    background-size: 105px 159px, 721px 188px;
    background-repeat: no-repeat;
    width: 721px;
    height: 188px;
}
```

El resultado de la página será el siguiente:

Imagenes

- Varias Imagenes de fondo definiendo su posicion y tamaño

Esta área de la página tiene tres imágenes de fondo distintas



Figura 5.180. Imagen redimensionada y situada.

Para que una imagen tenga el tamaño original, podemos utilizar la palabra clave “auto”, por ejemplo, para mostrar la imagen primera, escribiremos:

```
background-size: 105px 159px, auto auto;
```

5.6.8.4. Imágenes estableciendo el origen

Con la propiedad background-origin podemos establecer el punto donde se comenzará a dibujar el fondo. Puede tomar 3 valores:

- **border-box:** es el valor por defecto. El origen está en la esquina superior izquierda del borde de la caja, es decir, dentro del borde.
- **padding-box:** El origen está en la esquina superior izquierda, pero sin incluir los bordes.
- **content-box:** El origen está en la esquina superior, pero sin incluir ni los bordes, ni el relleno (padding) de la caja.

Su sintaxis será la siguiente:

```
background-origin: border-box/padding-box/content-box;
```

Vamos a ver cada uno de ellos con un ejemplo, para que sepáis diferenciarlos. Para ello, pintaremos un div de 200x150px, con un borde de 20px de color negro y un padding de 20px. Dentro de este pintaremos el logotipo en los 3 diferentes sitios de origen:

```
#recuadro1
{
    background-origin: border-box;
    border: 20px solid #000;
    padding: 20px;
    background-image: url(logotipo.png);
    background-color: #ff0;
    background-repeat: no-repeat;
    width: 200px;
    height: 150px;
}
```

Si cambiamos la propiedad background-origin, y le damos los otros valores:

```
background-origin: padding-box;
background-origin: content-box;
```

Si ejecutamos cada uno de los ejemplos, veremos las diferencias entre ellos:



Figura 5.181. Página con las propiedades border-box/padding-box/content-box.

5.6.9. Transiciones

Definamos primero que es una transición.



Una transición es un cambio progresivo de un elemento. Aplicado a los estilos CSS significa que dentro de una propiedad, para pasar de un valor a otro, lo haremos de manera progresiva, pero pasando por los valores intermedios.

De esta manera, parecerá que hemos realizado un efecto de animación, en propiedades que tengan que ver con la posición, el tamaño o el color.

Las transiciones sólo podrán darse en las propiedades CSS que tengan valores numéricos. Estas serán creadas por una serie de propiedades que controlarán los elementos, las propiedades CSS y el tiempo entre otras cosas.

5.6.9.1. Cambio de valor en propiedades

Para que haya una transición es necesario un cambio de valor en una propiedad. Esta operación la podemos realizar de dos maneras:

- Mediante JavaScript
- Mediante la pseudoclase: *hover*.

Para realizarlo con JavaScript, lo veremos en puntos posteriores, y ahora nos centraremos en realizar los ejemplos con *hover*. Lo normal es ver esta pseudoclase usada con los hipervínculos, no es exclusiva de ellos y se puede usar con cualquier elemento.

Hover funciona de la siguiente manera, hace que cambien el elemento cuando se le pasa el ratón por encima. Aunque el cambio se realiza instantáneamente, nosotros le aplicaremos una transición para que sea más progresivo.

5.6.9.2. Propiedades

Veamos primero las propiedades que podemos aplicar en esta pseudoclase para después ver diferentes ejemplos.

- `transition-property`: indica a qué propiedades se aplicaran las transiciones. Su sintaxis es:

`transition-property: none | all | <propiedad>+;`

Los valores que puede tener esta propiedad son:

- `none`: con este valor indicaremos que no se aplican las transiciones a ninguna propiedad. Es el valor por defecto.

- `all`: indica que las transiciones se aplicarán a todas las propiedades que puedan admitirlas.
- `<propiedad>+`: indicamos el nombre de las propiedades que queramos. Si hay más de una éstas deben ir separadas por comas.
- `transition-duration`: esta propiedad nos indica el tiempo que tardará en realizarse la transición. Se mide en segundos ("s") o milisegundos ("ms"). Su sintaxis es:

```
transition-duration: <tiempo>;
```



Con estas dos propiedades definidas hasta ahora ya podríamos hacer funcionar la transición.

- `transition-timing-function`: con esta propiedad indicaremos los cambios de la velocidad mientras se produce la transición. Su sintaxis es:

```
transition-timing-function: linear | ease | ease-in | ease-out |  
ease-in-out | cubic-bezier(n,n,n,n);
```

Los valores que puede tomar son:

`linear`: la velocidad será constante durante toda su duración.

`ease`: es el valor por defecto.

- Lenta al principio.
- Rápida en el medio.
- Finaliza mucho más lenta.

`ease-in`: la transición es:

- Lenta al principio.
- Al final va a máxima velocidad.

`ease-out`: la transición es:

- Rápida al principio.
- Disminuye su velocidad para ser lenta al final.

`ease-in-out`: la transición va:

- Lenta al principio.
- Rápida a la mitad.
- Acaba lenta.

- `cubic-bezier(n,n,n,n)`: la velocidad de la transición en base a una curva cúbica de Bezier. Ya hablamos de ella en capítulos anteriores. Para ello incorporamos cuatro valores “n” que serán números decimales entre el 0 y el 1.
- `transition-delay`: con esta propiedad indicaremos un retraso en el tiempo en empezar la transición. El valor se expresa con un número seguido por la letra “s” para segundos, o “ms” si está en milisegundos. Su sintaxis es:

```
transition-delay: <tiempo>;
```

Veamos todo esto con un ejemplo, en el que pintaremos un recuadro al que modificaremos su ancho y cambiaremos de color. Necesitamos definir en la hoja de estilos:

```
#recuadro1
{
    border-radius: 20px;
    background-color: #ff0;
    width: 200px;
    padding: 10px;

    transition-property: width;
    transition-duration: 3s;
    transition-property: background-color;
    transition-duration: 8s;

    -moz-transition-property: width;
    -moz-transition-duration: 5s;
    -moz-transition-property: background-color;
    -moz-transition-duration: 8s;

    -ms-transition-property: width;
    -ms-transition-duration: 5s;
    -ms-transition-property: background-color;
    -ms-transition-duration: 8s;

    -webkit-transition-property: width;
    -webkit-transition-duration: 5s;
    -webkit-transition-property: background-color;
    -webkit-transition-duration: 8s;

    -o-transition-property: width;
    -o-transition-duration: 5s;
    -o-transition-property: background-color;
    -o-transition-duration: 8s;
}

#recuadro1:hover
{
    width: 300px;
    background-color: #f00;
}
```

Definimos la transición indicándole a la propiedad width con una duración de un segundo y la propiedad background-color con un valor de 8 segundos.

Cuando pasemos el ratón por encima se lanzarán ambas transiciones:

- Cambiará de 200 píxeles a 300 píxeles en un segundo
- Cambiará del color amarillo al rojo en forma gradual en 8 segundos

Insertamos en nuestra página html el siguiente código, con el texto que ocupará el cuadro que después se modificará:

```
<article>
  <div id="recuadro1">
    <p>Una transición es un cambio progresivo de un elemento. Aplicado a los estilos CSS significa que dentro de una propiedad, para pasar de un valor a otro, lo haremos de manera progresiva, pero pasando por los valores intermedios</p>
  </div>
</article>
```

Ejecutamos la página, y cuando cargue completamente veremos un cuadro de la siguiente forma:

Transiciones

- Transicion de tamaño y color

Una transición es un cambio progresivo de un elemento. Aplicado a los estilos CSS significa que dentro de una propiedad, para pasar de un valor a otro, lo haremos de manera progresiva, pero pasando por los valores intermedios

Figura 5.182. Cuadro inicial

Como ocurre en la mayoría de las propiedades en CSS3, las propiedades anteriores pueden agruparse en una sola propiedad de tipo “shorthand”:

```
transition: <transition-property> || <transition-duration> ||
            <transition-timing-function> || <transition-delay>;
```

Hay que seguir el siguiente orden lógico cuando usemos este tipo de sintaxis:

- El nombre de la propiedad (transition-property).
- El tiempo (transition-duration).
- El tipo de velocidad de transición (transition-timing-function).
- El retraso en la ejecución (transition-delay).

El ejemplo anterior quedará definido de la siguiente manera:

```
#recuadro2
{
    border-radius: 20px;
    background-color: #ff0;
    width: 200px;
    padding: 10px;
    transition: width 5s, background-color 8s;
    -moz-transition: width 5s, background-color 8s;
    -ms-transition: width 5s, background-color 8s;
    -webkit-transition: width 5s, background-color 8s;
    -o-transition: width 5s, background-color 8s;
}

#recuadro2:hover
{
    width: 300px;
    background-color: #f00;
}
```

El efecto es el mismo, pero con un código muchísimo más abreviado.

5.6.9.3. Transiciones con JavaScript

Con el uso de JavaScript vamos a poder cambiar las propiedades de los elementos. Si primero aplicamos una transición a un elemento, y con JavaScript le cambiamos las propiedades, tendremos como resultado que el cambio se produce con una transición. De esta manera podremos crear efectos muy parecidos a las animaciones.

Veamos un ejemplo en el que daremos opción al usuario para controlar la posición de un elemento. Para realizar esto, debemos crear un elemento contenedor, y dentro de este, un cuadrado y dos botones con los que controlaremos el cuadrado:

```
<article>
    <div id="fondocaja">
        <div id="cuadrado">Pulsa el botón que quieras</div>
    </div>
    <p>
        <input type="button" value="Hacia la dcha" onclick="ida()" />
        <input type="button" value="Hacia la izda" onclick="vuelta()" />
    </p>
</article>
```

Cada botón tiene asociado mediante el evento onclick las funciones JavaScript que cambiarán el cuadrado. Las vemos a continuación:

```
<script type="text/javascript">
    window.onload = function () {
        //buscar elemento en el DOM
        elemento = document.getElementById("cuadrado")
    }
    //ida: cambiar posición y color de fondo.
    function ida() {
        elemento.style.left = "500px";
    }
    //vuelta: cambiar posición y color de fondo.
    function vuelta() {
        elemento.style.left = "150px";
    }
</script>
```

```

        elemento.style.backgroundColor = "#90c";
    }
//vuelta: volver a posición y color original.
function vuelta() {
    elemento.style.left = "20px";
    elemento.style.backgroundColor = "yellow";
}
</script>

```

Primero buscamos la referencia al elemento del DOM y después, mediante las funciones `ida()` y `vuelta()`, cambiamos al elemento, la propiedad `style` y le definimos los diferentes tamaños del margen izquierdo. De esta manera cambiarán las propiedades cuando pulsemos los botones.

Después creamos el código CSS:

```

#fondocaja
{
    width: 600px;
    height: 120px;
    border-radius: 20px;
    background-color: aquamarine;
    position: relative;
}

#cuadrado
{
    width: 80px;
    height: 80px;
    border-radius: 10px;
    background-color: yellow;
    position: relative;
    top: 20px;
    left: 20px;
    transition: all 2s linear;
    -moz-transition: all 2s linear;
    -webkit-transition: all 2s linear;
    -o-transition: all 2s linear;
}

```

A parte de definir y posicionar los elementos, aplicaremos la transición para el cuadrado.

Si ejecutamos la página veremos un cuadrado grande con otro interior más pequeño, cambiará de posición y de color cada vez que pulsemos el botón:

Transiciones

- Transición con JavaScript



Figura 5.183. Imagen de la transición.

5.6.10. Animaciones

Otra de las características que nos ofrece CSS3, es la posibilidad de crear animaciones de una manera sencilla.



Una **animación** es cualquier elemento o grupo de elementos que se mueve o cambia de aspecto progresivamente.

Para realizar las animaciones que después se insertaban en las web, debíamos recurrir a hacerlas con JavaScript o con algún programa externo como “Flash” o “Java”.

Las animaciones con CSS3 son bastante más sencillas que realizarlas con otros programas, aunque utilizan un método usado en programas como “flash”, llamado de los *fotogramas clave*.

5.6.10.1. Fotograma clave

Para definir este método, debemos definir primeramente que son los fotogramas clave.



Los fotogramas clave son los fotogramas que van a marcar el estado del elemento animado al principio, al final, o cuando éste se tenga que variar de una forma no previsible.

Para explicarlos de una forma más sencilla, imaginaremos que queremos que un elemento cambie su posición de izquierda a derecha. Lo que tendríamos que hacer es lo siguiente:

- Colocamos en el primer fotograma el elemento en un punto a la izquierda
- En el segundo fotograma, lo colocamos a la derecha.
- Despues definimos el tiempo que tiene que tardar de ir de la izquierda a la derecha
- Si posteriormente, queremos que el elemento se mueva hacia abajo.
- Lo colocamos en un tercer fotograma clave, que estará en un punto debajo del segundo.

Lo que hará este elemento será recorrer en línea recta del primer fotograma clave al segundo y después, del segundo fotograma al tercero. Entre los fotogramas 1, 2 y 3, existen unos estados intermedios que se van a calcular automáticamente.

Esto es lo que vamos a aprender a hacer con CSS3.

5.6.11. Sintaxis de una animación

En CSS los fotogramas clave se crean mediante la regla `@keyframes` la cual tiene la siguiente estructura. La sintaxis básica para una animación será la siguiente:

```
Elemento
{
    animation-name: [nombre de la animación];
    animation-duration: [tiempo de duración];
}

@keyframes [nombre de la animación]
{
    % / from
    {
        [propiedades y valores del estado inicial de la animación];
    }
    % / to
    {
        [propiedades y valores del estado final de la animación];
    }
}
```

Expliquemos que hemos hecho hasta ahora:

- Lo primero escribimos la palabra clave `@keyframes` seguido del nombre que le daremos a la animación.
- Dentro de las llaves pondremos los fotogramas clave. Estos deben seguir unas reglas de CSS compuestas de:
 - Selector: estos elemento realizan lo siguiente:
 - Indican el punto en el que se coloca el fotograma clave.
 - Se indican en tantos por ciento.
 - El primero será el 0% e indica el principio de la animación.
 - El último será el 100% e indica el final de la misma.
 - Se pueden colocar fotogramas intermedios indicando en % el punto en el que estará el fotograma.
 - Si únicamente hay dos fotogramas se usarán las palabras clave:
 - From para el primero
 - To para el último.
 - Declaración: serán las propiedades CSS que se le aplicaran al elemento cuando se visualice el fotograma. Estas propiedades cambiarán del primero al último de una forma gradual, durante el tiempo que dure la animación.

Lo que está dentro del elemento, lo entenderemos mejor después de explicar las propiedades.

5.6.11.1. Propiedades de una animación

Las propiedades que debemos conocer para realizar una animación son las siguientes:

- **animation-name:** es obligatoria para que funcione correctamente, ya que será la que nos diga a qué elemento le aplicaremos los fotogramas clave de la regla @keyframes:

```
animation-name: <nombre>;
```

El valor de <nombre> debe ser el mismo que se ha declarado en el selector de la regla @keyframes. De esta manera, los fotogramas se aplicarán al elemento al que se refiere esta regla.

- **animation-duration:** esta propiedad también es obligatoria y nos indica el tiempo que durará la animación.

```
animation-duration: <tiempo>;
```

Los valores de <tiempo> se expresan siempre en segundos ("s") o en milisegundos ("ms").

- **animation-direction:** esta propiedad nos indica si al llegar al final la animación acabará o va a volver en sentido inverso:

```
animation-direction: normal | alternate;
```

Los dos posibles valores son:

- **normal:** es el valor por defecto y una vez concluido el tiempo finaliza.
- **alternate:** indica se repite en sentido inverso, repitiéndose la secuencia de atrás a adelante.

No es una propiedad obligatoria.

- **animation-iteration-count:** indica las veces que se repetirá la animación, e irá de 1 hasta infinito.

```
animation-iteration-count: <entero_positivo> | infinite;
```

Su valor por defecto es 1, por lo que si no se pone, la animación se ejecutará una vez.

- **animation-delay:** indica el tiempo de retraso en empezar después de cargarse la página:

```
animation-delay: <tiempo>;
```

Los valores de <tiempo> se expresan igual que en la propiedad animation-duration.

- **animation-timing-function:** con esta propiedad controlaremos los cambios en la velocidad mientras se produce la animación.

```
animation-timing-function: linear | ease | ease-in | ease-out |  
ease-in-out | cubic-bezier(n,n,n,n);
```

Los valores que puede tomar son los mismos que en la propiedad transition-timing-function, de las propiedades de las transiciones:

- **linear:** la velocidad será constante durante toda su duración.
- **ease:** es el valor por defecto.
 - Lenta al principio.
 - Rápida en el medio.
 - Finaliza mucho más lenta.
- **ease-in:** la transición es:
 - Lenta al principio.
 - Al final va a máxima velocidad.
- **ease-out:** la transición es:
 - Rápida al principio.
 - Disminuye su velocidad para ser lenta al final.
- **ease-in-out:** la transición va:
 - Lenta al principio.
 - Rápida a la mitad.
 - Acaba lenta.
- **cubic-bezier(n,n,n,n):** la velocidad de la transición en base a una curva cúbica de Bezier. Ya hablamos de ella en capítulos anteriores. Para ello incorporamos cuatro valores “n” que serán números decimales entre el 0 y el 1.
- **animation-play-state:** indica si la animación está ejecutándose o está en pausa.

```
animation-play-state: paused | running;
```

Sus posibles valores son:

- **paused:** la animación está en pausa

- `running`: la animación se está ejecutando. Éste es el valor por defecto.

Lo normal es usar esta propiedad con JavaScript, para que mediante algún botón, el usuario pueda pausar o poner en marcha la animación.

- `animation` tipo "shorthand". Como todas las propiedades en CSS3, las propiedades anteriores pueden reunirse en una sola de tipo "shorthand":

```
animation: <animation-name> || <animation-duration> ||  
<animation-timing-function> || <animation-delay> || <animation-  
iteration-count> || <animation-direction>]
```

Como valores pondremos:

- Primero el nombre `animation-name`.
- Despues el tiempo de duración `animation-duration`.
- Despues el tipo de velocidad `animation-timing-function`.
- Seguimos con el tiempo de retardo de comienzo `animation-delay`.
- Despues el número de repeticiones `animation-iteration-count`.
- Por ultimo, la propiedad de dirección `animation-direction`.

Propiedades de animación y JavaScript: tenemos que seguir usando JavaScript si queremos interactuar con el usuario. Si queremos que pare o ponga en marcha la animación, o para que se pueda regular la velocidad, etc., tendremos que seguir usando JavaScript.

Estas propiedades se usan como cualquier otra propiedad CSS. Por ejemplo `animation-delay` se escribirá: `animationDelay`. Recordar de anteponer los prefijos para que funcionen correctamente:

- Firefox con el prefijo `-Moz-`
- Safari y Chrome con el prefijo `-webkit-`.

Por ejemplo, una vez obtenido el elemento del DOM escribiremos:

- `elemento.style.WebkitAnimationDelay = "paused";`
- `elemento.style.MozAnimationDelay = "paused";`

La primera línea activa la propiedad en Safari y Chrome, y la segunda en Firefox.

Y por último, debemos recordar las recomendaciones que nos hace la W3C.



- Las propiedades no obligatorias pueden omitirse.
- El orden puede cambiar únicamente en las propiedades no obligatorias.
- Las propiedades animation-name y animation-duration serán siempre las primeras.

5.6.11.2. Mi primera animación

Veamos un ejemplo donde emplearemos la sintaxis expuesta en los capítulos anteriores.



Debéis tener instalado cualquiera de los navegadores en su última versión para poder ver estos ejemplos en ejecución.

Aun así, desarrollaremos los ejemplos, insertando los prefijos para asegurarnos que funcionan en versiones anteriores.

Vamos a mostrar un recuadro con un texto que parte de la columna 30px y avance hasta la columna 300px y que cambie del color gris al color amarillo. Que se repita de forma indefinida y que se repita al revés.

Definimos en nuestra hoja de estilos el siguiente código:

```
#recuadro1
{
    left: 30px;
    position: relative;
    border-radius: 20px;
    background-color: #ddd;
    width: 200px;
    height: 100px;
    padding: 4px;

    animation-name: animacion1;
    animation-duration: 4s;
    animation-iteration-count: infinite;
    animation-direction: alternate;
    -moz-animation-name: animacion1;
    -moz-animation-duration: 4s;
    -moz-animation-iteration-count: infinite;
    -moz-animation-direction: alternate;
    -webkit-animation-name: animacion1;
    -webkit-animation-duration: 4s;
    -webkit-animation-iteration-count: infinite;
    -webkit-animation-direction: alternate;
```

```
}
```

Con este código hemos definido el tamaño del cuadro, su posición, su color, etc. Y hemos definido las propiedades:

```
animation-name: animacion1;
animation-duration: 4s;
animation-iteration-count: infinite;
animation-direction: alternate;
```

Con las que hemos definido el nombre de la regla, su duración, las veces que se repetirá y si queremos que se repita hacia atrás cuando llegue al final. Sigamos definiendo las reglas @keyframes:

```
@keyframes animacion1
{
    from
    {
        left: 30px;
        background-color: #ddd;
    }

    to
    {
        left: 300px;
        background-color: #ff0;
    }
}

@-moz-keyframes animacion1
{
    from
    {
        left: 30px;
        background-color: #ddd;
    }

    to
    {
        left: 300px;
        background-color: #ff0;
    }
}

@-webkit-keyframes animacion1
{
    from
    {
        left: 30px;
        background-color: #ddd;
    }

    to
    {
        left: 300px;
        background-color: #ff0;
    }
}
```

}

Lo que hemos hecho es decirle donde estará el primer fotograma a 30px del margen izquierdo y el color inicial que tendrá: #ddd. Y donde acabara a 300px del margen izquierdo y su color final.

Si os fijáis bien, las 3 son completamente iguales, lo único que cambia es la forma de definir la regla, para que funcione en todos los navegadores existentes:

```
@keyframes animacion1  
@-moz-keyframes animacion1  
@-webkit-keyframes animacion1
```

Y por último, necesitamos definir la etiqueta dentro de nuestra página HTML, que será el contenedor:

```
<article>  
  <div id="recuadro1">  
    <h3>Título</h3>  
    <p>Esta va a ser nuestra primera animación</p>  
  </div>  
</article>
```

Si ejecutamos la página en el navegador, veremos como el cuadro va y viene cambiando de color.

Para mejorar un poco más nuestra animación, vamos a implementarle dos botones, que pondrán en marcha y pararán el cuadro a nuestro antojo, y un desplegable con varias velocidades para poder seleccionar y aplicar.

Esto lo realizaremos mediante JavaScript. Insertamos el siguiente script en nuestra página:

```
<script type="text/javascript">  
//Buscamos el elemento animado en el DOM.  
window.onload = function () {  
    cuadro = document.getElementById("recuadro");  
}  
//parar la animación: propiedad animation-play-state: paused;  
function parar() {  
    cuadro.style.animationPlayState = "paused";  
    cuadro.style.MozAnimationPlayState = "paused";  
    cuadro.style.WebkitAnimationPlayState = "paused";  
}  
//reiniciar la animación: propiedad animation-play-state: running;  
function seguir() {  
    cuadro.style.animationPlayState = "running";  
    cuadro.style.MozAnimationPlayState = "running";  
    cuadro.style.WebkitAnimationPlayState = "running";  
}
```

```
//Cambiar la velocidad: propiedad animation-duration: "num"
function velocidad(num) {
    valor = num + "s";
    cuadro.style.animationDuration = valor;
    cuadro.style.MozAnimationDuration = valor;
    cuadro.style.WebkitAnimationDuration = valor;
}
</script>
```

Una vez que hemos capturado el objeto cuadro del DOM, tenemos tres funciones JavaScript:

- La función parar(), que detiene la animación.
- La función seguir() que reinicia la animación.
- La función velocidad() que indica el cambio de velocidad en la animación. Los números que pasamos como parámetros son los segundos que tarda la animación en producirse una vez.

Y en el código HTML de la página, introduciremos los 3 botones y el desplegable, que será donde seleccionemos la velocidad:

```
<article>
    <div id="recuadro">
        <h3>Título</h3>
        <p>Esta va a ser nuestra primera animacion</p>
    </div>
    <p>
        <input type="button" name="stop" value="stop" onclick=
            "parar()" />
        <input type="button" name="play" value="play" onclick=
            "seguir()" />
        <b>Elige la velocidad:</b>
        <select name="vel">
            <option selected="selected" onclick="velocidad(4)">
                Normal</option>
            <option onclick="velocidad(10)">Muy lento</option>
            <option onclick="velocidad(8)">Lento</option>
            <option onclick="velocidad(2)">Rápido</option>
            <option onclick="velocidad(1)">Muy Rápido</option>
        </select>
    </p>
</article>
```

Con este código hemos creado dos botones y una lista desplegable. En estos elementos hemos insertado los eventos de JavaScript que nos llevan a las funciones que cambian el comportamiento de la animación.

Si ejecutamos la página, veremos el siguiente resultado:

ANIMACIONES

- Animaciones con CSS3



Figura 5.184. Ejemplo en ejecución.

Vamos a seguir implementando acciones en nuestra aplicación. Ahora vamos a poner más puntos de movimiento de nuestro recuadro. Para realizar esto, en las reglas de los @keyframes insertamos el siguiente código:

```
@keyframes animacion1
{
    0%
    {
        left: 30px;
        top: 5px;
    }

    25%
    {
        left: 300px;
        top: 5px;
    }

    50%
    {
        left: 300px;
        top: 100px;
    }

    75%
    {
        left: 30px;
        top: 100px;
    }

    100%
    {
        left: 30px;
        top: 5px;
    }
}
```

Recordad que esto lo tenemos que cambiar en todos los keyframes. Además, quitaremos la línea:

```
animation-direction: alternate;
```

De esta manera, cuando llegue al final, no volverá a hacer el camino contrario, y el efecto será de un bucle completo. Además, cambiaremos el orden de los divs para que los botones aparezcan sobre el cuadro en vez de debajo:

```
<article>
  <div>
    <p>
      <input type="button" name="stop" value="stop" onclick="parar()"/>
      <input type="button" name="play" value="play" onclick="seguir()"/>
      <b>Elige la velocidad:</b>
      <select name="vel">
        <option selected="selected" onclick="velocidad(4)">
          Normal</option>
        <option onclick="velocidad(10)">Muy lento</option>
        <option onclick="velocidad(8)">Lento</option>
        <option onclick="velocidad(2)">Rápido</option>
        <option onclick="velocidad(1)">Muy Rápido</option>
      </select>
    </p>
  </div>
  <div id="recuadro">
    <h3>Título</h3>
    <p>Esta va a ser nuestra primera animación</p>
  </div>
</article>
```

Si ejecutamos la página en el navegador, veremos que el recuadro realiza un movimiento rectangular.

5.6.11.3. Avanzando un poco más

En este último punto vamos a ver como realizar una animación usando dos imágenes. La sensación final será de un avión que está volando por encima de una playa. Sobre un fondo en movimiento vamos a colocar la figura de un avión, esto dará la sensación de que es el avión el que se mueve cuando en realidad retrocede el fondo.

Para conseguir este efecto, la imagen de fondo tiene que dar la sensación de ser una tira continua que no se acaba. La técnica para conseguir esto es sencilla. Necesitamos un dibujo de fondo en el que su borde izquierdo coincida o sea muy parecido al borde derecho. Después, la ventana en la que lo mostraremos debe medir la mitad o menos del tamaño de la imagen.

Lo que haremos será pasar por el visor la imagen de derecha a izquierda viendo siempre una parte del mismo. Cuando se acaba la imagen, vuelve a empezar y siempre estaremos viendo el dibujo pasando de derecha a izquierda. El avión es una imagen con el fondo transparente, en formato png.

Puedes usar las imágenes que tú quieras o te las puedes descargar desde la plataforma. Las imágenes son las siguientes:



Figura 5.185. Playa.jpg.



Figura 5.186. Avión.png.

A continuación vamos a colocar los elementos en la página. Para ello necesitamos un “div” donde colocaremos los dibujos. Dentro del “article” ponemos el siguiente código:

```
<article>
  <h2>Avión volando:</h2>
  <div id="visor">
    
    
  </div>
</article>
```

Ahora vamos a crear la animación en el código CSS. Primero vamos a definir y posicionar ambas imágenes, por lo que tenemos que insertar lo siguiente:

```
#visor
{
  width: 400px;
  height: 400px;
  background-color: silver;
  overflow: hidden;
  position: absolute;
}
```

Hemos definido el visor donde mostraremos después la imagen. Le hemos dado un tamaño de 400x400px, y le hemos dado un `overflow: hidden;` para que no se vea el resto del fondo que permanece oculto.

Al haber dado un posicionamiento absoluto al visor, las posiciones del resto de las imágenes se miden desde el propio visor, de esa manera, si movemos el visor de sitio, las imágenes no quedarán descolocadas.

```
#playa
{
    position: relative;
    width: 1200px;
    height: 400px;
}
```

La imagen de fondo es tres veces más ancha que el ancho del visor. Así crearemos la sensación de un fondo continuo que no se acaba.

```
#avion
{
    position: absolute;
    top: 30px;
    left: 140px;
    width: 50px;
}
```

Al avión lo hemos redimensionado al tamaño que más nos convenía para colocar sobre la imagen del fondo.

Para crear la animación, lo primero es definir la regla @keyframes:

```
@keyframes playa {
    from { left: 0px; }

    to { left: -600px; }
}

@-moz-keyframes playa {
    from { left: 0px; }

    to { left: -600px; }
}

@-webkit-keyframes playa {
    from { left: 0px; }

    to { left: 600px; }
}
```

Debéis recordar, que por los problemas de compatibilidad con los navegadores, debemos repetirla varias veces. Seguimos en el código CSS, ahora incorporamos las propiedades de estilo al fondo de la playa:

```
#playa
{
    animation-name: playa;
    animation-duration: 5s;
    animation-iteration-count: infinite;
    animation-timing-function: linear;
```

```
-moz-animation-name: playa;  
-moz-animation-duration: 5s;  
-moz-animation-iteration-count: infinite;  
-moz-animation-timing-function: linear;  
  
-webkit-animation-name: playa;  
-webkit-animation-duration: 5s;  
-webkit-animation-iteration-count: infinite;  
-webkit-animation-timing-function: linear;  
}
```

Las propiedades indican que:

- La duración será de 5 segundos.
- Se repetirá indefinidamente.
- La velocidad es constante.

De esta manera, ya tendríamos definida nuestra animación. Si ejecutamos la página, veremos como se mueve el fondo de una forma continua y parece que el avión está sobrevolando la playa.

ANIMACIONES

• Animaciones con CSS3

Avion volando:



Figura 5.187. Página en ejecución.

5.6.12. Animaciones 3D

Aunque las llamamos animaciones son una variación de las transformaciones, ya que podemos crear animaciones en tres dimensiones a partir de algunos métodos de transform.

Debéis saber que los métodos de transform para 3D no están disponibles para todos los navegadores. A día de hoy (Diciembre 2012), únicamente en Safari y Chrome, y con el prefijo -webkit-. Veamos cuáles son estos métodos.

5.6.12.1. Traslación

El método **translate()** en 3D tiene la siguiente sintaxis:

```
transform: translate3d(<medidaX>, <medidaY>, <medidaZ>);
```

Es decir, lo único que hacemos es añadirle las letras “3d” y también un parámetro más, que es el que corresponde a la profundidad o al eje Z. A los métodos `translateX` y `translateY` se añade también el método:

```
transform: translateZ(<medidaZ>);
```

El cual indica la transformación en el eje Z o de profundidad. Tenéis que pensar que al ser imágenes planas, no se va a ver el movimiento a través del eje Z, por lo que no lo implementaremos en nuestros ejemplos.

Veamos con un ejemplo como realizar el movimiento mediante esta propiedad. Pintaremos 3 cuadros que desplazaremos en los ejes X e Y y uno de forma diagonal, combinando los dos anteriores. Usaremos el siguiente código HTML para mostrar nuestros cuadros:

```
<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>CSS3</title>
    <link href="Styles/Estilos.css" rel="stylesheet" />
</head>
<body>
    <header>
        <h1>Animaciones 3D</h1>

    </header>
    <nav>
        <ul>
            <li>Movimientos 3D</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <div id="capa1">translateX (100px)</div>
            <div id="capa2">translateY (100px)</div>
            <div id="capa3">translateXY (100px)</div>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>
```

En el archivo de la hoja de estilos insertamos el siguiente código:

```
@-webkit-keyframes cuadro1
{
    from
    {
        -webkit-transform: translateX(0px);
    }

    to
    {
        -webkit-transform: translateX(100px);
    }
}

@-webkit-keyframes cuadro2
{
    from
    {
        -webkit-transform: translateY(0px);
    }

    to
    {
        -webkit-transform: translateY(100px);
    }
}

@-webkit-keyframes cuadro3
{
    from
    {
        -webkit-transform: translateX(0px) translateY(0px);
    }

    to
    {
        -webkit-transform: translateX(-100px) translateY(-100px);
    }
}

#capa1, #capa2, #capa3
{
    width: 100px;
    height: 100px;
    margin: 50px;
    border-radius: 20px 20px 20px 20px;
    background-color: #aa0;
    padding: 10px;
    float: left;
    font: bold 14px arial;
}

#capa1
{
    -webkit-animation: cuadro1 3s alternate infinite;
}

#capa2
{
    -webkit-animation: cuadro2 3s alternate infinite;
}

#capa3
{
    -webkit-animation: cuadro3 3s alternate infinite;
}
```

Lo que hemos hecho con este código es simplemente asignar a cada cuadro una animación diferente, mediante las reglas de los @keyframes. Definimos el punto inicial, y el final, y el solo se encarga de realizar y pintar los puntos que faltan.

Vemos como quedaría nuestra página en ejecución.

Animaciones 3D

- Rotaciones 3D

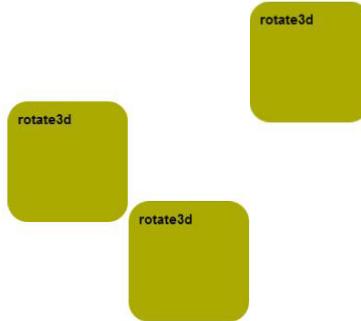


Figura 5.188. Animaciones con translate().

5.6.12.2. Escalado

El método scale sigue los mismos pasos que el anterior para sus propiedades en 3D, es decir le añadimos las letras “3d” y un parámetro más:

```
transform: scale3d(<numX>, <numY>, <numZ>);
```

Como en la traslación también tenemos el método:

```
-webkit-transform: scaleX(<numX>);  
-webkit-transform: scaleY(<numY>);  
-webkit-transform: scaleZ(<numZ>);
```

Debemos asignar un tamaño primero al elemento que queremos escalar, para después poderle aplicar el escalado. Los valores que puede tomar van de cero a infinito:

- Si asignamos 0, lo reducirá a nada.
- Si le damos 1, le dejará el tamaño original.
- Si asignamos 2, duplicará el tamaño del mismo y así sucesivamente.

Veamos un ejemplo con 3 cuadros que irán aumentando y disminuyendo su tamaño. Para realizarlo, utilizaremos el mismo código HTML que para la traslación, y únicamente variaremos las reglas @keyframes, cambiándolas por estas:

```
@-webkit-keyframes cuadro1 {  
    from { -webkit-transform: scaleX(1); }  
    to { -webkit-transform: scaleX(2); }  
}
```

```

@-webkit-keyframes cuadro2 {
    from { -webkit-transform: scaleY(1); }
    to { -webkit-transform: scaleY(2); }
}

@-webkit-keyframes cuadro3 {
    from { -webkit-transform: scale3d(1,1,1); }
    to { -webkit-transform: scale3d(2,2,2); }
}

```

El resto de la hoja de estilos no es necesaria que la toquemos. Si ejecutamos, veremos el siguiente resultado:

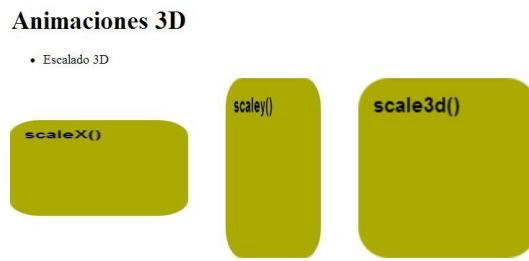


Figura 5.189. Animación de escalado.

5.6.12.3. Rotación

La rotación es el método que más variaciones tiene en 3D, ya que permite rotar un elemento en los tres ejes del plano, por lo tanto tenemos:

```

-webkit-transform: rotateX('angle');
-webkit-transform: rotateY('angle');
-webkit-transform: rotateZ('angle');

```

También tenemos el método rotate3d(), que en este caso varía un poco de los anteriores, ya que tiene 4 parámetros.

```
-webkit-transform: rotate3d(<numX>, <numY>, <numZ>, <angulo>);
```

Con los tres primeros parámetros indicaremos un eje de coordenadas del espacio, (X, Y, Z), que crearán un vector que marca una línea recta en el espacio. Este será el eje sobre el cual el elemento se inclinará los grados marcados en el cuarto parámetro.

Veamos un ejemplo de cada una de las rotaciones en 3d, en el que vamos a crear tres cuadros que rotarán en los distintos ejes que existen.

Para este ejemplo tomaremos el siguiente código HTML:

```

<!DOCTYPE html />
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>CSS3</title>
    <link href="Styles/Estilos.css" rel="stylesheet" />
</head>
<body>
    <header>
        <h1>Animaciones 3D</h1>

    </header>
    <nav>
        <ul>
            <li>Rotaciones 3D</li>
        </ul>
    </nav>
    <aside>
    </aside>
    <section>
        <article>
            <div id="capa1">rotate3d</div>
            <div id="capa2">rotate3d</div>
            <div id="capa3">rotate3d</div>
        </article>
    </section>
    <footer>
    </footer>
</body>
</html>

```

En nuestra hoja de estilos insertaremos el siguiente código:

```

@-webkit-keyframes cuadro1 {
    from {
        -webkit-transform: rotateX(0deg);
    }

    to {
        -webkit-transform: rotateX(360deg);
    }
}

@-webkit-keyframes cuadro2 {
    from {
        -webkit-transform: rotateY(0deg);
    }

    to {
        -webkit-transform: rotateY(360deg);
    }
}

@-webkit-keyframes cuadro3 {
    from {
        -webkit-transform: rotateZ(0deg);
    }

    to {
        -webkit-transform: rotateZ(360deg);
    }
}

```

Lo primero definimos las reglas @keyframes y definimos el inicio y el fin de cada uno de los fotogramas clave. El inicial será lo 0 grados y el final los 360°, que será la circunferencia completa. Recordad que CSS se encargará de rellenar automáticamente las imágenes que faltan entre el inicio y el fin.

```
#capa1, #capa2, #capa3
{
    width: 100px;
    height: 100px;
    margin: 50px;
    border-radius: 20px 20px 20px 20px;
    background-color: #aa0;
    padding: 10px;
    float: left;
    font: bold 14px arial;
}
```

Aquí hemos definido el estilo de los cuadros, que será el mismo para todos.

```
#capa1 {
    -webkit-animation: cuadro1 3s linear infinite;
}

#capa2 {
    -webkit-animation: cuadro2 3s linear infinite;
}

#capa3 {
    -webkit-animation: cuadro3 3s linear infinite;
}
```

Y por último, hemos incorporado las propiedades de estilo a cada una de los cuadros:
Las propiedades indican que:

- La duración será de 3 segundos.
- Se repetirá indefinidamente.
- La velocidad es constante.

Si ejecutamos nuestra página, veremos 3 cuadros rotando sobre los tres ejes:

Animaciones 3D

• Rotaciones 3D



Figura 5.190. Ejecución de las animaciones en 3D.

Veamos otros tres movimientos, combinando rotaciones en dos ejes en cada cuadro.
Para eso cambiaremos únicamente las @keyframes, definiendo las siguientes:

```

@-webkit-keyframes cuadro1 {
    from {
        -webkit-transform: rotateX(0deg) rotateY(0deg);
    }
    to {
        -webkit-transform: rotateX(360deg) rotateY(360deg);
    }
}

@-webkit-keyframes cuadro2 {
    from {
        -webkit-transform: rotateX(0deg) rotateZ(0deg);
    }
    to {
        -webkit-transform: rotateX(360deg) rotateZ(360deg);
    }
}

@-webkit-keyframes cuadro3 {
    from {
        -webkit-transform: rotateY(0deg) rotateZ(0deg);
    }
    to {
        -webkit-transform: rotateY(360deg) rotateZ(360deg);
    }
}

```

Si ejecutamos la página ahora, veremos otra animación diferente en cada uno de los cuadros:

Animaciones 3D

- Rotaciones 3D



Figura 5.191. Otras rotaciones diferentes.

Veamos ahora el uso de la propiedad `rotate3d()`, con otros tres ejemplos diferentes. Volvemos a modificar las @keyframes:

```

@-webkit-keyframes cuadro1
{
    from {
        -webkit-transform: rotateX(0deg) rotateY(0deg)
rotateZ(0deg);
    }
    to {
        -webkit-transform: rotateX(360deg) rotateY(360deg)
rotateZ(360deg);
    }
}

```

```

@-webkit-keyframes cuadro2
{
    from
    {
        -webkit-transform: rotate3d(9,3,0,0deg);
    }

    50%
    {
        -webkit-transform: rotate3d(-9,-3,0,180deg);
    }

    to
    {
        -webkit-transform: rotate3d(9,3,0,360deg);
    }
}

@-webkit-keyframes cuadro3
{
    from
    {
        -webkit-transform: rotate3d(1,1,1,0deg);
    }

    50%
    {
        -webkit-transform: rotate3d(-1,-1,-1,180deg);
    }

    to
    {
        -webkit-transform: rotate3d(1,1,1,360deg);
    }
}

```

Si ejecutamos, veremos como los cuadros giran en todos los ángulos:

Animaciones 3D

- Rotaciones 3D

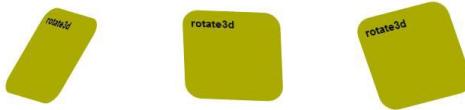


Figura 5.192. Animacion con rotate3d().



La animación CSS está creciendo y aún está en fase experimental, por lo que puede que en un futuro tenga variaciones. Procuraremos estar al tanto para poder incorporarlas.

RESUMEN

- Hemos conseguido tener una visión general de HTML5 y CSS3, además de aprender a implementar las diferentes etiquetas existentes, para dar a nuestras páginas un aspecto más profesional.
- Hemos realizado una introducción a HTML5 y CSS3, ampliando los conocimientos previos que teníamos de las versiones anteriores de ambas notaciones.
- Ahora conocemos las etiquetas más utilizadas de HTML5, y la forma de introducirlas, también hemos trabajado con la etiquetas audio, vídeo y el elemento canvas.
- Ya sabemos cuales son las limitaciones de los navegadores y los prefijos necesarios para usar las etiquetas en cada uno de ellos.
- Despues de trabajar con las etiquetas de CSS3, ya sabemos:
 - Modificar los bordes.
 - Aplicar sombras.
 - Realizar transformaciones en 2D.
 - Cambiar la opacidad de los elementos.
 - Insertar múltiples columnas de texto.
 - Importar fuentes externas al navegador.
 - Usar imágenes de una forma sencilla.
 - Realizar transiciones.
 - Realizar animaciones en 2D y 3D.

