

Contenidos multimedia en la Web: Animaciones.

Caso práctico

Como todas las semanas, se reúnen los miembros del equipo de trabajo de la empresa **BK programación** para comentar las incidencias de la semana anterior.

Ana ha realizado un storyboard del proyecto de animación para la presentación de la panadería "Migas amigas" y se lo muestra a sus compañeros para que le den su opinión.

—Tened en cuenta que esto es sólo una planificación previa que sirve para hacerse una idea del orden en el que se van presentar los diferentes elementos —dice **Ana** mientras se lo va mostrando a sus compañeros.

—¿Quiere decir eso que no son las escenas definitivas? —pregunta **María**.

—¡Claro! —le responde **Ana**, que continúa diciendo: —Aquí falta todo el colorido que queramos añadir, las texturas, el movimiento, el fondo de la escena, y todo lo que se nos ocurra.

Ana interviene diciendo: —A mí me parece que has hecho un guión muy original y a la vez muy detallado. No creo que tengáis problema para desarrollarlo, no queda ningún detalle que añadir salvo lo que ya has dicho. ¿Tienes alguna idea en mente?

—La verdad es que tengo algunas, pero no me he decidido todavía por ninguna en concreto —responde **Ana**.

—¿Qué te parece si me las comentas y realizamos las dos que creamos mejores? —pregunta **Carlos** y añade —yo puedo ayudarte a hacer una de ellas.

—Me parece estupendo, gracias **Carlos**, me serás de gran ayuda —dice **Ana**.

Ana, sabe que realizar una animación es una labor divertida pero que lleva muchas horas de trabajo. Siempre anima a todo el equipo en las labores que realizan valorando sus esfuerzos y procurando realizar críticas constructivas que sirvan para futuros proyectos. En este caso les recuerda que tratándose de una animación en una página principal hay que facilitar al usuario el que se salte la animación, ya que a los usuarios no les gusta tener que ver siempre la misma animación cuando acceden a un sitio Web; y como todas las semanas da por terminada la reunión diciendo: —Pues a trabajar todos.



Elaboración propia. **Ana**. (Uso educativo no comercial).

1.- Las animaciones en la Web.

Caso práctico

Carlos está muy contento de que **Ana** le permita colaborar en la realización de una animación. El ya ha explorado algunas herramientas para realizar animaciones y se ha centrado en la herramienta Google Web Designer.

Ha hecho algunas animaciones de prueba, pero nada serio, y está deseando realizar algo que realmente tenga alguna utilidad.



Elaboración propia. Carlos. (Uso educativo no comercial.)

Según la definición de la wikipedia: animación es un proceso utilizado para dar la sensación de movimiento a imágenes o dibujos.

El objetivo de esta unidad es conocer la manera de incluir y generar animaciones para un sitio web, mostrando la tecnología y las tendencias actuales en este campo, pero sin profundizar en el uso de una herramienta concreta.

Las animaciones web se usan para muchos fines: para mostrar un contenido, para publicidad, como banners, detalles de diseño, objetos animados, etcétera. Actualmente la web ofrece muchas alternativas para la creación de estas animaciones, algunas son implementadas por todos los navegadores, otras son exclusivas de unos pocos, algunas siguen el estándar W3C y otras no. La variedad es amplia, y es el desarrollador quien tiene que elegir en cada momento la solución que mejor se adapte a sus necesidades, a la tecnología empleada en el desarrollo del sitio web y a la tecnología que permiten los navegadores. Aunque siempre es recomendable seguir y utilizar estándares propuestos por la W3C y que puedan ser visualizados en el mayor número de navegadores.



Pezibear. Lobo aullando. ([Licencia Pixabay](#))

Todas estas tecnologías y recursos podríamos clasificarlos en tres grandes grupos.

- ✓ Imágenes con formato de animación.
- ✓ Herramientas de programación o desarrollo.
- ✓ Aplicaciones, servicios online, librerías, API, etcétera.

Podría pensarse que el último punto podría incluirse en las herramientas de programación o desarrollo, pero hemos querido separar el desarrollo directo (o genérico) utilizando CSS, SVG, CANVAS, etcétera, de aquellas herramientas que nos facilitan una interfaz o la utilización de librerías o API que nos generarán una aplicación para incorporarla a nuestro desarrollo. En el apartado dos se realiza la animación de una forma directa.

Recuerda que, como ya hemos dicho más de una vez, los recursos gráficos se emplean mucho en la Web y que, utilizados adecuadamente, pueden mejorar el aprendizaje del usuario y añadir valor a nuestro sitio pero que si se utilizan de forma incorrecta producen el efecto contrario y causan un enorme rechazo del usuario.

El que las personas que se dedican al diseño de interfaces puedan usar las tecnologías a su alcance y tengan mucha experiencia en el uso de estas tecnologías, no significa que deban usarlas siempre.

El desarrollador y diseñador de un sitio web debe conocer todas las alternativas actuales para la creación de animaciones de sitios web, sin embargo, esto no significa que tenga la creatividad suficiente para que lo que diseña sea atractivo.

1.1.- Formatos de animaciones.

En la unidad de trabajo de imágenes vimos que el formato GIF era un formato empleado en las imágenes que tienen entre 2 y 256 colores y que uno de sus usos principales seguía siendo el mostrar imágenes animadas en las páginas Web, aunque se solía emplear en la creación de los iconos que acompañan a los enlaces y los logotipos y, en general, en aquellas imágenes con grandes áreas de color sólido.



Beatriz Eugenia Buyo Pérez. Muñeco. (Elaboración propia.)

Hay distintos formatos de archivo que permiten visualizar animaciones en la Web, algunos como el GIF, Webp y SVG ya mencionados en unidades anteriores, otros menos conocidos como MNG y otros ya en desuso que fueron muy populares en su día como SWF y que citamos aquí más que por su uso por su conocimiento por la importancia que en su día tuvieron.

✓ Formato GIF

Un gif animado consiste en una serie de imágenes, en formato gif que están colocadas consecutivamente y se muestran en pantalla durante un intervalo de tiempo determinado.

Las ventajas de un gif animado es su rápida descarga, su nitidez, el uso de transparencias, y que se puede insertar en HTML como cualquier otra imagen. Pero, tiene como desventaja que las imágenes deben tener un número fijo de colores (un máximo de 256). Al tratarse de un formato de mapa de bits si la animación tiene muchas imágenes estáticas el tamaño del fichero resultante puede ser excesivo para que sea práctico.

Existen en la red multitud de sitios web que ofrecen gifs animados ya creados. Y también hay herramientas para crearlos y convertirlos a otros formatos (como vimos anteriormente). Algunas son las siguientes:

- ✓ **Microsoft GIF Animator** es una aplicación diseñada para crear presentaciones animadas en formato de imagen GIF.
- ✓ **GIF Construction Set Professional**: es una herramienta muy utilizada para crear gifs y conversión entre diferentes formatos.
- ✓ **Picasion**: es una herramienta online que permite crear gifs animados a partir de un máximo de 10 imágenes, aunque su funcionalidad es muy limitada.

✓ Formato SVG

Es un formato de imágenes vectorial que se definen como un fichero XML, el cual puede ser editado por un editor de texto. Como hemos comentado en varias ocasiones este formato está tomando cada día más importancia dentro del mundo web y con relación a nuestra unidad, indicar que soporta animaciones. Además se le pueden aplicar filtros, degradados, estilos (CSS), etcétera. Este formato es abierto y soportado por la mayoría de los navegadores y se caracteriza principalmente por el poco espacio que ocupa en disco.

✓ Formato MNG

El formato MNG es un formato de fichero libre de derechos para imágenes animadas y es una extensión del formato de imagen PNG. Hoy en día no es recomendado para la web, aunque es soportado por muchos navegadores. En el siguiente enlace podemos ver el soporte de este tipo de formato.

✓ Formato Webp

Como ya comentamos este formato es de código abierto y propiedad de Google, en la unidad anterior explicamos su ventajas y desventajas y aquí lo citamos ya que soporta también animaciones (imágenes en movimiento) además de trasparencias. Recordar que ofrece compresión con y sin pérdidas, se basa en el formato de video VP8. Comparándolo con el formato GIF, da mejores prestaciones, además de soportar una paleta de colores mayor.

En el siguiente enlace podemos encontrar algunos ejemplos de este tipo de formatos: [Giphy.com](https://giphy.com).

✓ Formatos Flash FLV(.fla) y Shockwave SWF(Small Web Format)

Aunque son un formato en desuso para la web, los citamos por la importancia que tuvieron, así durante más de una década, han sido los formatos más atractivos de la web

Un archivo FLV es básicamente un archivo de vídeo que se ha formateado para transmitir a través del reproductor de Flash. Su extensión de archivo global es una abreviatura de su nombre formal, Flash Video. Los archivos flash (.flv) son creados, inicialmente, con la herramienta Adobe Flash (antes Macromedia Flash), aunque existen otras herramientas con las que también pueden crearse. flv es un formato abierto. Los archivos .fla son los archivos editables.

SWF es un formato vectorial muy popular para presentar contenidos animados más complejos que Flash (juegos, entre otros) permitiendo integrar tanto imagen, texto, vídeo y audio. Además, incluye un lenguaje de programación ActionScript que hace que estas animaciones puedan interactuar con el usuario, al reconocer, por ejemplo, eventos de ratón. Sin embargo, tiene como contrapartida que el software Adobe para crear shockwave es más caro que el que se usa para Flash. Además, a la hora de insertarlo en una web, Flash ofrece soluciones más extendidas.

Estos formatos no se utilicen hoy en día y se debe principalmente a estas dos razones:

- ✓ La dificultad de que el contenido de los .SWF puedan ser indexados en su totalidad por los motores de búsqueda.
- ✓ El auge de iPad, Apple se negó a introducir la tecnología Flash en sus dispositivos. Esto hace que los navegadores de estos dispositivos (Safari) no puedan visualizar archivos .SWF. Como veremos más adelante, la solución adoptada ante este problema, ha sido la aparición de HTML5 y conversores de .SWF a HTML5.

Algunas de las herramientas utilizadas para crear .SWF son las siguientes:

- ✓ **Adobe Flash Professional**: se trata de una aplicación multiplataforma que permite la creación y manipulación de gráficos vectoriales destinada a la realización de animaciones y a la producción de contenido interactivo usando el lenguaje Action Script.
- ✓ **SWF Easy**: es una herramienta sencilla con gran funcionalidad para crear archivos .SWF. Permite textos e imágenes estáticas, y además, la posibilidad de usar Action Script para permitir interacción.

Para saber más

En 2005, Macromedia fue absorbida por Adobe Systems. Como resultado de la adquisición, los formatos .FLV y .SWF original de Macromedia se convirtieron en formatos propietarios de Adobe, pero existe una pequeña diferencia que es que Adobe mantiene el formato .FLV como un formato abierto y el formato .SWF tiene una autoría más restringida.

Tanto .FLV como .SWF son formatos contenedores multimedia que puede agrupar múltiples flujos de datos en un único archivo. Realmente, ambos fueron creados por la misma compañía (Macromedia).

Macromedia publicó el formato FLV en 2005

1.2.- Herramientas de programación.

Otra forma de incluir animaciones en nuestra web a parte de los formatos de imágenes animadas, es mediante la utilización de lenguajes de programación, éste será el punto más importante de nuestra unidad y donde nos centraremos. Entre los lenguajes que podemos utilizar están:

- ✓ HTML y JavaScript: utilizando la etiqueta `<canvas></canvas>` de HTML y el lenguaje de programación JavaScript.
- ✓ CSS: el objeto que queremos animar hay que definirlo en HTML.
- ✓ HTML y CSS: utilizando la etiqueta `<svg></svg>` de HTML y pudiendo definir estilos con CSS.
- ✓ JavaScript / jQuery: el objeto que queremos animar hay que definirlo en HTML.

Pasamos a comentar un poco más cada uno de estos apartados, aunque en la unidad se desarrollan de una forma más extensa.



[Ciker-Free-Vector-Images](#). Animación ([Licencia Pixabay](#))

✓ API HTML y JavaScript: CANVAS.

HTML5 introdujo varias APIs, una de estas API permite la creación de dibujos o imágenes como mapa de bits, mediante la etiqueta `<canvas></canvas>`, que denominaremos también como lienzo. Dentro de la etiqueta `<canvas></canvas>` podremos dibujar figuras, mostrar imágenes y videos y mediante JavaScript podremos dotar de movimiento, mostrando y eliminando los elementos que vamos a visualizar. Así, realmente canvas por si solo, no permitiría animaciones, pero con el complemento de JavaScript podremos realizar las animaciones que vamos buscando.

✓ CSS3

CSS3 incorpora la posibilidad de realizar animaciones mediante reglas, así veremos como podemos realizar animaciones muy simples con un estado inicial y final (transiciones) y animaciones con diferentes estados intermedios (además del estado inicial y final) mediante la definición de la regla `@keyframe` y la propiedad `animation`.

✓ HTML (y CSS): SVG

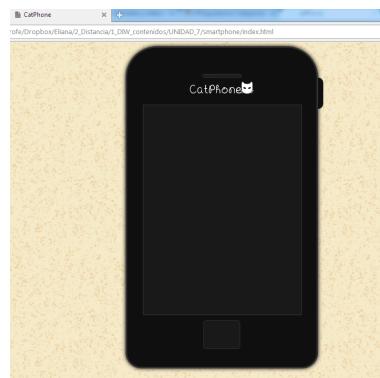
De la misma forma que la etiqueta `<canvas></canvas>` nos permite definir dibujos como mapa de bits, la etiqueta `<svg></svg>` permite la creación de dibujos o formas básicas como líneas, rectángulos, círculos, etcétera, pero se definen como gráficos vectoriales. SVG es un lenguaje de marcas creado por el W3C. Una de las ventajas de esta etiqueta frente a `<canvas></canvas>` es que los elementos que se definen dentro de la etiqueta `<svg></svg>` pueden ser referenciados por CSS y forman parte del DOM, por eso en el epígrafe hemos puesto SVG y CSS, ya que además de las propias animaciones que se definen con SVG a cada uno de los elementos que conforma este dibujo se le pueden aplicar animaciones también con CSS.

✓ JavaScript / jQuery

Otra alternativa para crear animaciones que permite la interacción con los usuarios es combinar HTML con, por ejemplo, JavaScript. La potencia de JavaScript como lenguaje de programación (con eventos), ayuda a crear animaciones fácilmente soportables por los navegadores de forma nativa.

Si no queremos usar directamente JavaScript, podemos usar alguna librería alternativa como jQuery, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. jQuery es software libre y de código abierto. jQuery ahorra tiempo y espacio en el desarrollo de animaciones comparando con usar JavaScript directamente. Incluimos en este apartado jQuery y no en el siguiente punto ya que en la siguiente unidad ("Contenidos web interactivos") se estudiará esta librería de forma más detallada.

A continuación mostramos un ejemplo de animaciones con jQuery, donde se simula el comportamiento de los elementos de la pantalla de un dispositivo móvil, al desbloquearse la misma, seleccionar alguno de los elementos ó volver al menú principal. Este código ha sido desarrollado por la alumna Florina Roxana Marinca de Diseño de Interfaces Web en la modalidad presencial.



Florina Roxana Marinca. Animaciones con jQuery (Dominio público)

[SmartPhone](#). (zip - 0,23 MB)

Para saber más

En el siguiente enlace podemos ver como hacer animaciones directamente con JavaScript.

Animaciones con JavaScript

El mejor resumen de lo que es jQuery lo podemos encontrar en el lema de su propia página web: "La librería JavaScript para escribir menos y hacer más". Existen multitud de tutoriales y fuentes donde se puede encontrar la sintaxis de esta librería, en este caso empezamos por la propia.

jquery.com

Autoevaluación

Selecciona la/s afirmacion/es correcta/s

- La regla @keyframe está relacionada con las animaciones en CSS
- SVG crea gráficos y animaciones de tipo bitmap (mapa de bits).
- La API canvas dispone del método animation para realizar animaciones
- JavaScript permite realizar animaciones e interactuar con el usuario.

Correcto, @keyframe permite definir los estados de una animación.

Incorrecto, SVG permite crear gráficos y animaciones basados en elementos vectoriales

Incorrecto, la API canvas no dispone de métodos propios para realizar animaciones, deberemos utilizar JavaScript.

Correcto.

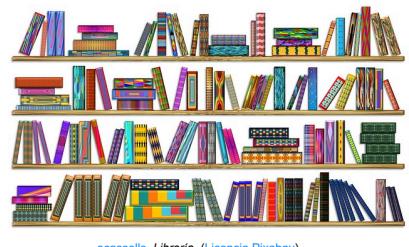
Solución

1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

1.3.- Entornos gráficos, librerías y/o API de animaciones.

En este apartado nos vamos a centrar en mencionar productos y librerías que están enfocados directamente con la realización o tratamiento de animaciones, dicho esto, podríamos pensar que jQuery debería estar en este apartado, pero jQuery es una librería genérica de JavaScript y en este apartado se citarán librerías o APIs específicas de animaciones.

Actualmente en el mercado de la Web existe un amplio abanico de software para la realización de animaciones, el cual está en constante evolución, el uso de uno u otro programa estará determinado por el tipo de animación a realizar y el presupuesto que tengamos.



ecassells, Librería. (Licencia Pixabay)

- ✓ **Entornos gráficos:** para crear animaciones visualmente. Luego, estos entornos permiten guardar (exportar) las animaciones creadas en HTML5, CSS3 y JavaScript. Es decir, de la misma manera que se pueden guardar en un formato determinado, se puede generar código HTML5 correspondiente. Actualmente grandes empresas como Google o Adobe ofrecen herramientas gráficas para este fin:
 - ◆ **Adobe Edge:** es un programa que sirve para crear animaciones web y otros contenidos interactivos mediante HTML5, CSS3 y JavaScript. Tiene una interfaz intuitiva y clara, muy inspirada en Adobe Flash. Permite importar documentos HTML, CSS y soporta los formatos de imagen más populares (svg, png, jpeg y gif). Esta herramienta no es gratuita.
 - ◆ **Google web designer:** permite la creación de contenido animado en HTML5, mejorando las posibilidades de crear publicidad.
 - ◆ **Hype:** es otra herramienta con similares características destinada a ordenadores Mac OS X. Esta es otra herramienta no gratuita.
 - ◆ **Inkscape:** es un editor de gráficos en formato vectoriales **SVG** (Scalable Vector Graphics, Gráficos Vectoriales Escalables) gratuito, libre y multiplataforma.
- ✓ **Aplicaciones de conversión de archivos Flash (swf) a HTML5, CSS3 y JavaScript:** este tipo de aplicaciones de conversión se presentan como herramientas de transición entre archivos Flash y HTML5. Muchas web actuales con animaciones Flash no pueden ser visualizadas correctamente en algunos dispositivos móviles. Por ello, estas herramientas se presentan como solución temporal de cambio entre ambas tecnologías.
 - ◆ **Wallaby:** es una aplicación Adobe para convertir archivos (.fla, Flash editable) a HTML5.
 - ◆ **Swiffy:** es una utilidad gratuita de Google para convertir .swf a HTML5.

Además existen librerías y frameworks destinadas o enfocadas solo a la realización de animaciones, a continuación nombramos algunas de ellas:

- ✓ animeJS: en el siguiente enlace se puede ver un [ejemplo realizado con la librería animeJS](#).
- ✓ GreenSock: framework en JavaScript para realizar animaciones.
- ✓ Velocity.js
- ✓ Animate.css

Para saber más

A continuación mostramos algunas herramientas que podemos utilizar para desarrollar los contenidos de esta unidad:

Para generar animaciones podemos utilizar "Google Web Designer" la información oficial relacionada con este software se puede encontrar en el siguiente enlace:

[Google Web Designer](#).

En el siguiente enlace se pueden consultar más librerías y frameworks relacionados con animaciones:

[Librerías y frameworks sobre animaciones](#)

Para generar gráficos vectoriales, podemos trabajar con "Inkscape" Toda la información que se proporciona de forma oficial sobre este software se puede encontrar en el siguiente enlace:

[Página oficial de Inkscape](#)

Autoevaluación

¿Google Web Designer es una aplicación web que permite crear animaciones y banner publicitarios en HTML5?

- Verdadero Falso

Verdadero

Correcto, Google Web Designer es una aplicación web avanzada que permite diseñar y compilar anuncios HTML5, imágenes, vídeos y otros contenidos web para su empresa en una interfaz de código y visual integrada.

2.- CSS3

Como se ha comentado existen diferentes formas de mostrar una animación en nuestra página web, en este apartado nos centraremos en las posibilidades que nos ofrece CSS, hoy en día este sea quizás el método más utilizado, el cual nos permitirá realizar animaciones complejas y profesionales. Antes de CSS se podrían realizar animaciones con herramientas externas o bien directamente con JavaScript, aunque esto puede llevar complejidad en el desarrollo.

Con la evolución de CSS y las posibilidades que éste nos ofrece, se ha convertido en el lenguaje preferente para incluir animaciones dentro de nuestros desarrollos web.

Básicamente una animación en CSS nos permitirá definir mediante reglas las propiedades de un objeto, así definimos un conjunto de estados, inicial y final o inicial, intermedios y final y el navegador se encargará de realizar la renderización del objeto para partiendo del estado inicial llegar al final, realizando si se han definido, los estados intermedios.

Entre las principales ventajas que encontramos al realizar las animaciones con CSS, es la facilidad para implementarlas y por otra parte que es el propio navegador el que se encarga de realizar la renderización y la optimización de la animación.

Posteriormente analizaremos otro tipo de tecnología como es SVG, pues bien, CSS nos permitirá animar también objetos o parte de estos definidos con SVG.

Uno de los inconvenientes o carencias que puede tener esta tipología de animaciones, es que si queremos interactuar con ellas, tendremos que utilizar JavaScript. Usaremos este lenguaje entre otras cosas, para saber si una animación está en ejecución, ha finalizado o queremos que se comience a ejecutarse.

A día de hoy la mayoría de navegadores soportan las animaciones definidas con CSS, pero si queremos mantener compatibilidad con versiones más antiguas, quizás tengamos que poner prefijos a algunas de las propiedades que vamos a estudiar:

- ✓ webkit-: Chrome y Safari.
- ✓ -moz-: Firefox.
- ✓ -o-: Opera.

Para finalizar comentamos algunas de las ventajas que presenta hacer animaciones con CSS:

- ✓ Compatibilidad con la mayoría de navegadores.
- ✓ Facilidad de implementación.
- ✓ Es un lenguaje que ya es familiar para los desarrolladores front-end.

Este apartado se divide en tres subapartados donde veremos como realizar: transformaciones, transiciones y animaciones.



Pexels. Animaciones con CSS. ([Licencia Pixabay](#))

2.1.- Transformaciones.

Las transformaciones en CSS nos permitirán modificar el comportamiento visual de un elemento, así las transformaciones más habituales las podremos realizar en 2D, aunque también pueden realizarse en 3D. Para poder realizar una transformación utilizaremos la propiedad transform.

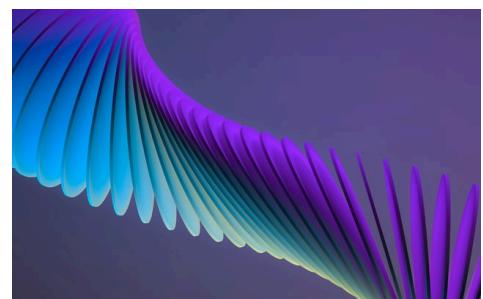
Las operaciones de transformación que podemos realizar a un objeto serán: rotaciones, desplazamiento (en diferentes ejes), escalados, deformaciones, etcétera. Además pueden combinarse realizando por ejemplo un desplazamiento y una rotación.

A continuación vamos a analizar las principales transformaciones que podemos realizar.

Los ejemplos que se muestran se realizan sobre un div, así se define un div con un identificador (id) y posteriormente se le aplica a este una operación de transformación. En la imagen se muestra el div original y como quedaría con la transformación. Al div transformado se le aplica un fondo para diferenciarlo.

Translación: Traslada un elemento. Esta translación puede realizarse, en el eje x, en el eje y o en ambos a la vez.

- ✓ translateX(valorx)
- ✓ translateY(valory)
- ✓ translate(valorx, valory)

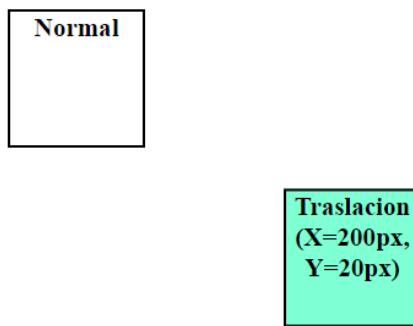


Milad Fakurian, Transformaciones ([Licencia Unsplash](#))

Translación (Código).

```
#traslacion { transform: translate(200px, 20px); }
```

Resultado de una translación.



Elaboración propia. Translación ([CC BY-NC-SA](#))

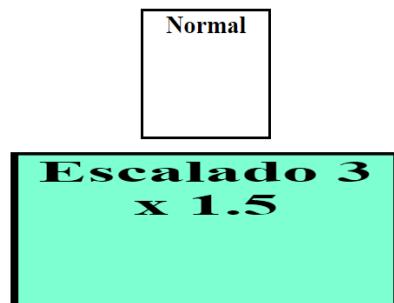
Escalado: Escala un elemento. Este escalado puede realizarse en el eje x, en el eje y o en ambos. Como parámetro indicamos cuanto queremos escalar nuestro elemento. Así un valor de 2 significa que queremos doblar el tamaño de nuestro objeto.

- ✓ scaleX (aumentoX)
- ✓ scaleY (aumentoY)
- ✓ scale (aumentoX, aumentoY)

Escalado (código).

```
#escalado {  
    transform: scale(3, 1.5);  
}
```

Resultado de un escalado.



Elaboración propia. Escalado ([CC-BY-NC-SA](#))

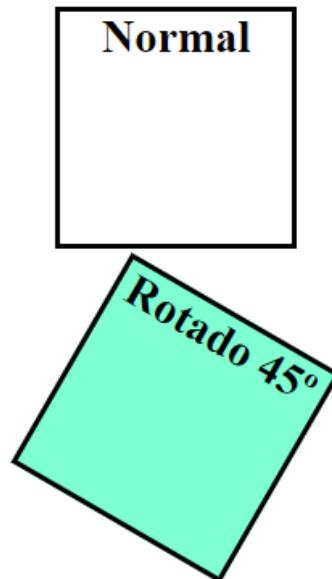
Rotación: Permite girar un elemento un determinado número de grados.

- ✓ rotate (grados)

Rotación (código).

```
#rotado {  
    transform: rotate(45deg);  
}
```

Resultado de una rotación.



Elaboración propia. Rotación ([CC BY-NC-SA](#))

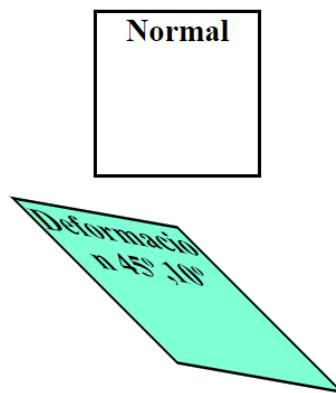
Deformación: Permite aplicar una deformación de un objeto en el eje x, en el eje y o en ambos a la vez. Como parámetro se pasa los grados (grados sexagesimales) de deformación que queremos aplicar a un objeto determinado.

- ✓ skewX(gradoX)
- ✓ skewY(gradoY)
- ✓ skewXY(gradoX,gradoY)

Deformación (código).

```
#deformacion {
    transform: skew(45deg, 10deg);
}
```

Resultado de una deformación.



Elaboración propia. Deformación ([CC BY-NC-SA](#))

A continuación se adjunta el ejemplo completo con todas las transformaciones aplicadas, para que puedas practicar con él.

Transformaciones (código)

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Transformaciones</title>
    <meta charset="UTF-8">
    <style type="text/css">

        *{
            box-sizing: border-box;
            margin:0px;
        }

        .objeto_contenedor{
            width:90vw;
            margin:10px;
            text-align: center;
        }

        .objeto{
            width:100px;
            height:100px;
            border:solid 2px black;
            background: aquamarine;
            margin:10px;
            margin-left:100px;
            overflow-wrap: break-word;
        }

        /* Objeto normal sin ninguna trasformación */
        #normal {
            background: white;
        }

        /* Objeto al cual se le aplica una traslación en el eje X de 200px y 20px en eje Y*/
        #traslacion {
            transform: translate(200px, 20px);
        }

        #traslacion:active::before {
            content:"Realizamos una traslación en el eje x de 200px y 20px en el eje y";
            font-weight: bold;
            color:red;
        }

        /* Objeto al cual se le aplica una rotación de 45% */
        #rotado {
            background: aquamarine;
            transform: rotate(30deg);
        }

        /* Objeto al cual se le aplica un escalado de 1,5 en el eje y 3 en el eje x */
        #escalado {
            position:relative;
            left:500px;
            background: aquamarine;
            transform: scale(3, 1.2);
        }

        /* Objeto al cual se le aplica una deformación de 45º en el eje X y 10 grados en el eje Y */
        #deformacion {
            background: aquamarine;
            transform: skew(45deg, 10deg);
        }
    </style>

```

```

        }

    #varios {
        background: purple;
        transform: rotate(30deg) scale(2, 0.8) translate(100px, -150px);
    }

    </style>
</head>

<body>
    <h1> Ejemplo de transformaciones</h1>
    <div class="objeto_contenedor">
        <div id="normal" class="objeto">
            <h3>Normal</h3>
        </div>

        <div id="traslacion" class="objeto">
            <h3>Traslacion (X=200px, Y=20px)</h3>
        </div>

        <div id="rotado" class="objeto">
            <h3>Rotado 45º</h3>
        </div>

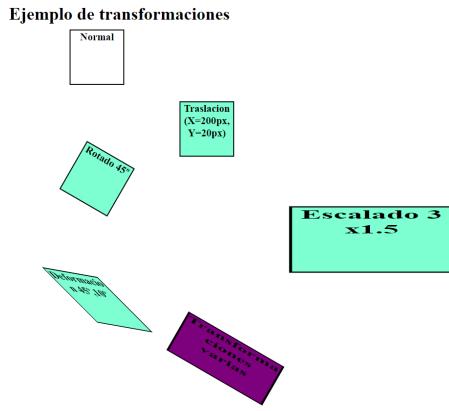
        <div id="escalado" class="objeto">
            <h3>Escalado 3 x1.5</h3>
        </div>

        <div id="deformacion" class="objeto">
            <h3>Deformacion 45º, 10º</h3>
        </div>

        <div id="varios" class="objeto">
            <h3>Transformaciones varias</h3>
        </div>
    </div>
</body>
</html>

```

Resultado del código.



Elaboración propia.. Transformaciones ([CC BY-NC-SA](#))

Las transformaciones en 3D llevan una complejidad mayor que las 2D, por lo que solo comentaremos las principales propiedades y mostraremos algún ejemplo.

Hasta ahora hemos visto cómo transformar un objeto en 2D, pero CSS también nos permite realizar transformaciones en 3D, así tendremos además de los ejes X e Y el eje Z, por lo tanto necesitaremos de propiedades adicionales para poder realizar transformaciones en este eje.

Estas propiedades son:

- ✓ translate3d: igual que en 2D pero podemos definir los tres ejes X, Y y Z.
- ✓ scale3d: igual que en 2D pero podemos definir los tres ejes X, Y y Z.
- ✓ rotate3d: igual que en 2D pero podemos definir los tres ejes X, Y y Z.
- ✓ skew3d: igual que en 2D pero podemos definir los tres ejes X, Y y Z.
- ✓ perspective(n): nos permite indicar la distancia que existirá entre el observador y el elemento al cual vamos aplicar la transformación.
- ✓ perspective-origin: nos permite definir el origen del observador indicando su posición en el eje X e Y. Por defecto sus valores son 50% 50%, es decir, están en el centro del elemento. Podemos indicar diferentes medidas como px, %, también podemos modificar la propiedad con los valores left, right y center para el eje X y top, bottom y center para el eje Y.
- ✓ transform-origin: nos permite desplazar el eje del objeto sobre el que estamos aplicando la transformación.
- ✓ transform-style: permite los valores preserve-3d y flat. El primero hará que si el objeto contiene objetos hijos se mantenga la transformación 3D para ellos y el segundo no.
- ✓ backface-visibility : permite ver el dorso de un objeto. Es una propiedad que permitirá los valores visible (por defecto) o hidden.

Para saber más

Dejamos un enlace donde encontramos un generador donde se puede ver mejor como funciona la propiedad perspective

[Transformaciones 3D](#)

A continuación dejamos un enlace donde se muestra como se puede animar un objeto en 3D, usando para ello transformaciones 3D.

[Animación 3D](#)

2.2.- Transiciones.

Una transición nos permitirá que un objeto o una propiedad experimente un cambio, donde nosotros podemos indicar (entre otras características) como se realiza ese cambio y el tiempo que deseamos que dure. Para crear una transición tendremos que definir un estado inicial y un estado final (asignando valores a propiedades que conformarán reglas CSS, además de otras propiedades o características como son, tiempo que durará, tiempo de retraso y ritmo).

La transición **no** se puede aplicar sobre todas las propiedades de un objeto, generalmente podremos aplicarlas sobre aquellas en las que podemos indicar una unidad de medida (algo cuantificable), colores y opciones de visibilidad e invisibilidad. En el siguiente enlace podemos consultar las propiedades sobre las cuales podemos aplicar nuestra transición.

[Propiedades que pueden utilizarse en una transición.](#)

Para aplicar una transición a una, varias o todas las propiedades de un elemento tendremos que utilizar la propiedad transition. Esta propiedad controlará entre otras cuestiones la duración, tiempo de retraso en comenzar el cambio y el ritmo. Todas estas características podemos definirlas en la la propiedad transition de forma simultanea, aunque nosotros vamos a analizarlas primero de forma individualizada.

Propiedades relacionadas con las transiciones:

- ✓ transition-property: indicamos la propiedad o propiedades (separados por comas) sobre las que queremos realizar la transición.
 - ◆ transition-property: color; /* estamos indicando que se va a producir la transición sobre la propiedad color*/
 - ◆ transition-property: color, border; /* estamos indicando que se va a producir la transición sobre las propiedades color y border*/
- ✓ transition-duration: Indicamos el tiempo en segundos que aplicaremos a cada propiedad que participa de la transición.
 - ◆ transition-duration: 3s; /* indicamos que la transición durará 3 segundos para todas las propiedades */
 - ◆ transition-duration: 3s 1s; /* indicamos que la transición durará 3 segundos para el color y 1 segundo para el borde */
- ✓ transition-timing-function: indicamos el ritmo del cambio y puede tomar los siguientes valores:
 - ◆ ease: especifica un efecto de transición con un inicio lento, después rápido, y al final de nuevo lento (valor predeterminado).
 - ◆ linear: especifica un efecto de transición con la misma velocidad de inicio a fin.
 - ◆ ease-in: especifica un efecto de transición con un inicio lento.
 - ◆ ease-out: especifica un efecto de transición con un final lento.
 - ◆ ease-in-out: especifica un efecto de transición con un inicio y final lento.
 - ◆ cubic-bezier(n,n,n,n): permite definir los valores propios en una función cúbica-Bezier. En este tipo de funciones cada uno de los valores (n), serán valores numéricos comprendidos entre 0 y 1. En el siguiente enlace podemos ver los valores que puede tomar esta propiedad.
 - transition-timing-function: linear; /* indicamos que tendremos un ritmo constante */
 - transition-timing-function: linear ease-out ; /*indicamos que tendremos un ritmo constante para el color y para el borde un final lento */
- ✓ transition-delay: indicamos el retraso con el que empezará la transición.
 - ◆ transition-delay: 2s; /* indicamos que esperaremos dos segundos antes de comenzar la animación */
 - ◆ transition-delay: 2s 4s; /* indicamos que esperaremos dos segundos para comenzar el cambio del color y 4 segundos para el cambio del borde.*/

Todas estas propiedades pueden incluirse directamente en la propiedad transition. A continuación vamos a analizar una serie de ejemplos con las diferentes posibilidades que nos ofrecen estas propiedades.

- ✓ transition: podemos indicar propiedades, tiempo en el que se produce la transición, ritmo, tiempo de retraso. A continuación indicamos un ejemplo:
 - ◆ transition: all 3s ease 1s; /* sobre todas las propiedades definidas en el cambio, la transición durará 3 segundos, con un ritmo con un inicio lento, después rápido, y al final de nuevo lento y que tardará un segundo en comenzar dicha animación */
 - ◆ transition: width 3s linear, background-color 2s ease; /* realizamos una transición sobre la propiedad width que durará 3 segundos con un ritmo linear y realizamos una transición sobre el color que durará 2 segundos con el ritmo de tipo ease.
- ✓ transition: <property> <duration> <timing-function> <delay>

En el siguiente ejemplo, definimos la transición sobre dos propiedades, indicando para las mismas el mismo tiempo de duración y el mismo ritmo.

```
transition-property: width, background-color;
transition-duration: 5s;
transition-timing-function: linear;
transition-delay: 1s
```

En el siguiente ejemplo definimos la transición sobre dos propiedades, indicando tiempos diferentes para cada propiedad:

```
transition-property: width, background-color;
transition-duration: 5s, 2s;
```



Brian Merrill. Movimiento. ([Licencia Pixabay](#))

Hasta ahora hemos hablado de que definimos un cambio, pero, ¿cómo podemos definir este cambio en nuestro desarrollo?. Para ello podremos utilizar las pseudoclases que vimos en la unidad relacionada con CSS como puede ser :hover , :active , etcétera. Así definiremos un estado para un objeto en nuestro CSS (inicio) y posteriormente cuando se aplica la pseudoclase :hover (pasar el ratón por encima) definiremos otro estado (fin). El paso del estado inicio a fin es lo que define la transición, indicando las propiedades que queremos que se vean afectadas por el cambio y como van a cambiar estas propiedades (duración, ritmo, tiempo). A continuación vamos a comentar un ejemplo:

Lo primero que haremos es seleccionar un objeto y aplicarle los estilos mediante una regla:

Sobre un div que tendrá como identificador transición (#transicion) definimos un estado inicial

```
#transicion {
    background-color:plum;
    width: 115px;
    height: 115px;
    border: solid 2px black;
    margin: 100px;
    text-align: center;
}
```

posteriormente definimos el estado final, indicamos que se va producir con la pseudoclase :active definimos los valores de la propiedad transition

```
#transicion:hover {
    background-color: yellowgreen;
    opacity: 0.5;
    width: 200px;
    height: 200px;

    transform: rotate(90deg);
    /* La transición se aplica sobre todas las propiedades que cambian, concretamente, fondo, ancho, alto y la transformación y dura 5 s
    transition: all, 5s;
}
```

En este ejemplo cambiamos el fondo, la opacidad, el alto y el ancho y además realizamos una transformación de un giro de 90 grados.

Todos estos cambios se van a producir en 5 segundos, esto lo realizamos mediante la propiedad: transition: all, 5s;

A continuación creamos otro objeto que llamaremos transicion2. Tendremos el primer estado inicial y final, pero ahora vamos a manipular las propiedades para que cambien en tiempos diferentes y ritmos diferentes.

Analizamos la siguiente transición:

```
<span></span>
<span>      </span><span>transition</span><span>: height </span><span>1s</span><span> </span><span>ease</span><span> </span>
<span>3s</span><span>,background-color </span><span>5s</span><span>      </span><span>linear</span><span>, width </span>
<span>10s</span><span>      </span><span>ease-in</span><span>, transform </span><span>15s</span><span>      </span><span>ease-out
10s</span><span>;</span>
```

- ✓ La propiedad `height` tardará 3 segundos en empezar a cambiar y el cambio lo realizará en 1 segundo con un ritmo de tipo `ease`.
- ✓ `background-color` La propiedad `background-color` cambiará en 5 segundos sin retraso y con un ritmo `linear`.
- ✓ La propiedad `width` cambiará en 10 segundos sin retraso y con un ritmo `ease-in`.
- ✓ Y por último la transformación (rotación), propiedad `transform` cambiará en 15 segundos con un retraso de 10 segundo un ritmo `ease-out`.

Como puede observarse el alto del objeto se realiza de forma rápida y la transformación comienza cuando el resto de cambios se han producido.

A continuación dejamos un video donde podemos ver estas dos transiciones.

Elaboración propia. ([CC BY-NC-SA](#))

En los ejemplos anteriores hemos definido la transición en estado final, pero podemos definir también una transición cuando estamos definiendo el estado inicial de nuestro objeto, así tendríamos una transición de entrada y otra de salida, esto lo realizaremos definiendo la propiedad transition en el estado inicial y final. De esta forma por ejemplo en un :hover, cuando el ratón deje de estar encima del objeto, el cambio no se producirá de inmediato, sino que se realizará en función de la transición que hayamos definido.

De todo esto se deduce que la transición puede definirse solo en el estado inicial y no en el estado final, de esta forma, con nuestro ejemplo el cambio con el :hover se produciría de forma inmediata y la vuelta al estado inicial se haría conforme lo hayamos definido en la transición.

Como puede verse en el video, centrándonos en la transición 2 y 3, en la transición dos al quitar el ratón el estado del objeto cambia de forma inmediata, mientras que en la transición 3 tarda 5 segundos.

Para ver como funciona mejor el ritmo dejamos el código de unas transición con diferentes valores para la propiedad transition-timing-function que la hemos incluido directamente en la propiedad transition.

Código ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Ejemplo de Transiciones</title>
    <meta charset="UTF-8">
    <style type="text/css">
        h2{text-align: center;}
        #transicion {
            background-color:plum;
            width: 115px;
            height: 115px;
            border: solid 2px black;
            margin: 100px;
            text-align: center;
        }
        /* La transición se activa con pseudo-clase :active, es decir al hacer click en el objeto */

        #transicion:hover {
            background-color: yellowgreen;
            opacity: 0.5;
            width: 200px;
            height: 200px;

            transform: rotate(90deg);
            /* La transición se aplica sobre todas las propiedades que cambian, concretamente, fondo, ancho, alto y la transformación
            transition: all, 5s;
        }

        .contenedor{
            width:90%;
            border:solid blue 2px;
            text-align: center;
        }

        .timing {
            background-color:rgb(116, 186, 168);
            width: 150px;
            height: 50px;
            border: solid 2px black;
            margin: 10px;
            left:2%;
            position:relative;
        }

        /* En las siguientes transiciones cambiaremos el ritmo o timing */

        #ease:active {
            /* La transición se aplica sobre una propiedad en concreto left, durará 5 segundos con un timing ease.*/
            /* Como se puede apreciar el color cambia de inmediato y el desplazamiento se produce en 5 segundos*/
            left: 80%;
            background-color: hotpink;
            transition: left 5s ease;
        }

        #ease-in:active {
            left: 80%;
            background-color: hotpink;
            transition: left 5s ease-in;
        }

        #ease-out:active {
            left: 80%;
            background-color: hotpink;
            transition: left 5s ease-out;
        }

        #ease-in-out:active {
            left: 80%;
            background-color: hotpink;
            transition: left 5s ease-in-out;
        }
    </style>

```

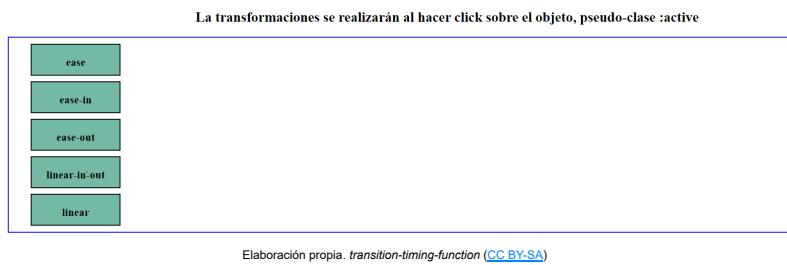
```

        #linear:active {
            left: 80%;
            background-color: hotpink;
            transition: all 5s linear;
        }
    </style>
</head>

<body>
    <h2> La transformación se realizará al hacer pasar el ratón sobre el objeto, pseudo-clase :hover </h2>
    <div class="contenedor">
        <div id="transicion">
            <h4>Transformación - Pon el ratón encima</h4>
        </div>
    </div>
    <h2> La transformaciones se realizarán al hacer click sobre el objeto, pseudo-clase :active</h2>
    <div class="contenedor">
        <div class="timing" id="ease">
            <h3>ease</h3>
        </div>
        <div class="timing" id="ease-in">
            <h3>ease-in</h3>
        </div>
        <div class="timing" id="ease-out">
            <h3>ease-out</h3>
        </div>
        <div class="timing" id="ease-in-out">
            <h3>linear-in-out</h3>
        </div>
        <div class="timing" id="linear">
            <h3>linear</h3>
        </div>
    </div>
</body>
</html>

```

Resultado.



Fuentes con ejemplos.

[Código con diferentes ejemplos sobre transiciones](#) (zip - 2,82 KB)

Para saber más

Uno de los valores que puede tomar la propiedad transition-timing-function es una función curva de Bezier, por eso se deja el siguiente enlace, para profundizar un poco más sobre este tipo de funciones.

[Curvas de Bezier.](#)

Autoevaluación

La propiedad indica el retraso con el que empezará la transición.

La propiedad indica el tiempo que durará la transición.

La propiedad define las propiedades sobre las que se aplicará la transición.

[Averiguar la puntuación](#) [Mostrar retroalimentación](#) [Mostrar/Eliminar las respuestas](#)

transition-delay: indicamos el retraso con el que empezará la transición.

transition-duration: indicamos el tiempo en segundos que aplicaremos a cada propiedad que participa de la transición

transition-property: indicamos la propiedad o propiedades (separados por comas) sobre las que queremos realizar la transición.

2.3.- Animaciones (@keyframe).

Como hemos visto las transiciones nos permiten crear animaciones simples donde definimos un estado inicial y un estado final, pero si quisieramos definir una animación con más estados (estados intermedios), las transiciones no lo permiten. Para poder realizar animaciones más complejas CSS dispone de la propiedad animation. Esta propiedad permite que un objeto realice una animación, que definiremos con la regla @keyframe. Al igual que ocurría con la propiedad transition, veremos como animation permitirá definir las características con las que queramos que se ejecute la animación, tiempo, tiempo de retraso, dirección, etcétera. Esto podemos realizarlo directamente en la propiedad animation o mediante subpropiedades de animation.

A un objeto podemos asignarle más de una animación, es decir definir varias animaciones y estas asignarlas al objeto u objetos que queramos.

[OpenClipart-Vectors. Animación ardilla \(Licencia Pixabay\)](#)

Para definir la animación, nos valdremos de lo que denominaremos como **fotogramas clave**, donde definiremos las propiedades de nuestro elemento y los valores que van a tener. Podemos definir tantos fotogramas clave como queramos. En un primer momento nos centraremos en una animación sencilla, donde solo tendremos un estado inicial y un estado final (sería algo similar a una transición). En cada uno de estos estados (fotogramas clave) indicamos las propiedades sobre las que vamos a actuar y su valor.

```
@keyframes ejemplo_simple {
  from {
    background-color:aqua;
    width: 100px;
    height: 100px;
    border: solid 2px black;
    margin: 100px;
  }
  to {
    background-color: olivedrab;
    width: 200px;
    height: 200px;
    transform: rotate(90deg);
    opacity: 0.5;
  }
}
```

Analizando el código después de @keyframe tenemos el nombre de la animación en este caso ejemplo_simple, abrimos y cerramos llaves {} y dentro definimos los diferentes estados, en este caso en from, definimos el estado inicial y en to el estado final.

Una vez que hemos definido nuestra animación veremos como asignarla a un objeto y como queremos que se ejecute sobre este objeto. Al igual que ocurre con las transiciones, tendremos diferentes propiedades como tiempo que durará la animación, ritmo, tiempo de retraso antes de que se inicie la animación, etcétera. Esto como hemos dicho se puede realizar sobre la propiedad animation o sobre las siguientes subpropiedades:

- ✓ animation-name: nombre de la animación.
- ✓ animation-duration: indicaremos en segundos (s) el tiempo que durará la animación.
- ✓ animation-timing-function: indicaremos al igual con las transiciones cómo se comportará nuestra animación, definimos el ritmo.
- ✓ animation-iteration-count: indicará el número de repeticiones que tendrá nuestra animación, le asignaremos un valor entero. Si queremos que nuestra animación esté repitiéndose de forma ininterrumpida, asignaremos el valor infinite.
- ✓ animation-direction: puede realizarse de la forma en que hemos definido la animación o al revés, es decir, empezaremos por el estado final y terminaremos con el estado inicial ([simulador de estados](#)). Podemos tener los siguientes valores:
 - ◆ normal: ejecución normal.
 - ◆ reverse: se ejecuta en sentido inverso.
 - ◆ alternate: se ejecuta en sentido normal (de inicio a fin) y luego al revés (de fin a inicio).
 - ◆ alternate-reverse: se ejecuta al revés (de final a inicio) y luego en sentido normal (estado inicial a estado final)
- ✓ animation-delay: tiempo de retraso con el que empezará la animación.
- ✓ animation-fill-mode: definiremos los valores que se tendrán las propiedades del objeto antes y después de que se produzca la animación.
 - ◆ none: no se aplicará ningún estilo al objeto ni antes ni después de la animación. Es el valor por defecto.
 - ◆ forwards: el objeto conserva los valores de las propiedades del último fotograma clave (depende de la dirección y del número de iteraciones definidas en la animación).
 - ◆ backwards: el objeto conservará los valores de la propiedad del primer fotograma clave y los conservará durante el periodo de retraso (depende de la dirección y del número de iteraciones definidas en la animación).
- ✓ animation-play-state: nos indica si la animación se está ejecutando o se encuentra en pausa, los valores que puede tener esta propiedad son ([simulador de los valores](#)):
 - ◆ paused: paramos la animación
 - ◆ running: continuamos o comenzamos la ejecución de la animación (valor por defecto)

Todos esas subpropiedades podemos plasmarlas en la propiedad animation como hemos comentado al principio.

animation : <i>name duration timing-function delay iteration-count direction fill-mode play-state</i>;

En el anterior ejemplo, teníamos un estado inicial y uno final, lo cual puede parecerse a la transición, pero con la regla @keyframe, podemos definir más estados intermedios o fotogramas clave, tantos como queramos. De esta forma vamos configurando nuestra animación, indicando el comportamiento de nuestro elemento en cada fotograma clave (o porcentaje). Así cada uno de estos estados lo definiremos indicando el momento en que el elemento u objeto obtendrá esas propiedades, lo haremos mediante un número (de 0 a 100) y las propiedades que queremos el objeto tome en ese momento. Por ejemplo:

50% {top: 250px;right: 125px;}<code>



El elemento en mitad de la animación (50%) se le asignará a las propiedad top un valor de 250px y a la propiedad right: 125px.

Creamos una animación con tres estados:

```
@keyframes simulacion_a {
    0% {
        top: 0px;
        right: 0px;
        background-color: white;
    }
    50% {
        top: 250px;
        right: 125px;
        opacity: 0.8;
        background-color: blue;
    }
    100% {
        top: 150px;
        right: 0px;
        opacity: 0.2;
        background-color: red;
    }
}
```

Una vez definida nuestra animación indicamos como se va a aplicar ésta, para eso utilizamos las subpropiedades definidas anteriormente o la propiedad animation. Particularmente el código queda más intuitivo utilizando las subpropiedades.

Así sobre el objeto animacion_a le aplicaremos la animacion simulacion_a, que durará 5 segundos y se repetirá de forma infinita.

```
#animacion_a {
    animation-name: simulacion_a;
    animation-duration: 5s;
    animation-iteration-count: infinite;
```

Código ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Ejemplo de Animaciones</title>
    <meta charset="UTF-8">
    <style type="text/css">
        @keyframes ejemplo_simple {
            from {
                background: gray;
                width: 100px;
                height: 100px;
                border: solid 2px black;
                margin: 100px;
            }
            to {
                background-color: olivedrab;
                width: 200px;
                height: 200px;
                transform: rotate(90deg);
                opacity: 0.5;
            }
        }
        /*Al objeto animacion se le aplicará la animación con nombre ejemplo simple, durará 10 segundos, con un ritmo ease-in-out, h
        /*comenzando normalmente valor normal, podemos aplicar también el valor reverse y con un retardo de 2 segundos */

        #animacion {
            animation-name: ejemplo_simple;
            animation-duration: 10s;
            animation-timing-function: ease-in-out;
            animation-iteration-count: 3;
            animation-direction: normal;
            animation-delay: 2s;
        }

        @keyframes ejemplo_2_fotogramas {
            0% {
```

```

background: gray;
width: 100px;
height: 100px;
border: solid 2px black;
margin: 100px;
}
100% {
background-color: olivedrab;
width: 200px;
height: 200px;
transform: rotate(90deg);
opacity: 0.5;
}
}
/*Igual que la anterior animación pero se ha cambiado la propiedad direction a reverse*/
#animacion2 {
animation-name: ejemplo_2_fotogramas;
animation-duration: 10s;
animation-timing-function: ease-in-out;
animation-iteration-count: 3;
animation-direction: reverse;
animation-delay: 2s;
}
/* Animación con diferentes fotogramas clave */

@keyframes tamano {
55% {
transform: scale(2, 2);
transform: rotate(90deg);
}
80% {
transform: scale(0.5, 0.5);
}
100% {
transform: scale(1, 1);
transform: rotate(-90deg);
}
}
/* Animación con diferentes fotogramas clave, para cada porcentaje de la duración de la animación definimos como se comporta */

@keyframes simulaciona {
0% {
top: 0px;
right: 0px;
background-color: white;
}
15% {
top: 100px;
right: 50px;
opacity: 0.6;
}
55% {
top: 200px;
right: 175px;
opacity: 0.1;
}
75% {
top: 250px;
right: 125px;
opacity: 0.6;
background: blue;
}
100% {
top: 350px;
right: 0px;
opacity: 0.8;
background: red;
}
}
@keyframes simulacionb {
0% {
top: 0px;
right: 0px;
opacity: 0;
}
15% {

```

```

        top: 100px;
        right: 50px;
        opacity: 0.3;
        background: aqua;
    }
    45% {
        top: 150px;
        right: 125px;
        opacity: 0.6;
        background: aquamarine;
    }
    55% {
        top: 200px;
        right: 175;
        opacity: 1;
        background: green;
    }
    75% {
        top: 250px;
        right: 125;
        opacity: 0.6;
    }
    100% {
        top: 350px;
        right: 0px;
        opacity: 0;
    }
}
/* Para este objeto definimos dos animaciones a la vez */

.animacion {
    position: absolute;
    text-align: center;
    background: pink;
    width: 50px;
    height: 50px;
    border: solid 2px black;
    border-radius: 50px;
}

#animacionA {
    animation-name: simulaciona, tamano;
    animation-duration: 5s;
    animation-iteration-count: infinite;
}

#animacionB {
    animation-name: simulacionb, tamano;
    animation-duration: 5s;
    animation-iteration-count: infinite;
}

```

</style>

</head>

<body>

 <h2> Ejemplo de animación con from - to </h2>

 <div id="animacion">

 <h3>Animación</h3>

 </div>

 <h2> Ejemplo de animación definiendo dos fotogramas clave 0% y 100%</h2>

 <div id="animacion2">

 <h3>Animación</h3>

 </div>

 <div class="animacion" id="animacionA">

 <h3>A</h3>

 </div>

 <div class="animacion" id="animacionB">

 <h3>B</h3>

 </div>

</body>

</html>

Código.

A continuación se deja un fichero con diferentes animaciones para su estudio (se incluye la animación de un péndulo que puede ser iniciada y parada con botones).

[Animaciones](#) (zip - 264,14 KB)

Se deja también un ejemplo de una animación utilizando propiedades de transformación 3D

[Animaciones 3D](#) (zip - 383,98 KB)

En los ejemplos de código anteriores, se ha incluido la animación de un péndulo que cuya animación puede ser parada y reanudada con dos botones, para ello utilizamos un poco de javascript y las propiedades animation-play-state con los valores: paused y running.

La animación se puede visualizar en el fichero y no tiene demasiada complejidad y mostramos el código asociado a los dos botones que permiten comenzar y pausar la animación

```
<script>
    // Botón que permite parar la animación<br />        var boton1 = document.getElementById("parar"); // Selecciona el botón por su ID
    boton1.addEventListener("click", function() { // Agrega un evento de clic al botón
        var pendulo1 = document.getElementById("pendulo1"); //seleccionamos el péndulo por id
        pendulo1.style.animationPlayState = "paused";           // modificamos la propiedad CSS animationPlayState

    });
    // Botón que permite iniciar la animación
    var boton2 = document.getElementById("reanudar"); // Selecciona el botón por su ID
    boton2.addEventListener("click", function() { // Agrega un evento de clic al botón
        var pendulo1 = document.getElementById("pendulo1");
        pendulo1.style.animationPlayState = "running";
    });
</script>
```

Para saber más

A continuación dejamos dos enlaces donde podemos seguir profundizando sobre las transiciones y las animaciones:

- ✓ En este enlace podemos conocer como interactuar en las animaciones con JavaScript.

[Animaciones y JavaScript](#)

- ✓ En este enlace se pueden estudiar diferentes ejemplos de animaciones y transiciones.

[Transiciones y animaciones con CSS.](#)

3.- API Canvas.

Previamente a crear animaciones en HTML5 es necesario conocer cuáles son las novedades que introdujo el lenguaje al respecto hace unos años.

HTML5 introduce varias API (interfaces de programación de aplicaciones) para proveer acceso a librerías incluidas en los navegadores. En este caso hablamos de la API Canvas, que permitirá generar e imprimir gráficos en pantalla, crear animaciones o manipular imágenes y videos. La forma en que se trata este tipo de imágenes es como si fueran imágenes mapas de bits, es decir trabajamos el contenido del elemento canvas pixel a pixel.

Canvas en español significa lienzo, y es básicamente eso, un área donde podemos dibujar, crear animaciones y desarrollar aplicaciones interactivas, como si fuera un lienzo.

Si bien el elemento canvas se crea con la etiqueta `<canvas></canvas>`, cuando hablamos de canvas nos referimos a toda la API que incluye un conjunto de funciones para dibujar líneas, rectángulos, círculos, etcétera, añadir texto y realizar operaciones de transformación con estos objetos como rotaciones, escalados, desplazamientos. Esto es a grandes rasgos lo que podremos realizar.

El elemento canvas permitirá especificar un área de la página donde se puede, a través de scripts, dibujar y renderizar imágenes, lo que amplía notablemente las posibilidades de las páginas dinámicas y permite hacer cosas que hasta su día estaban reservadas a los desarrolladores en Flash. La ventaja que nos ofrece canvas frente a Flash es que no necesita de ningún plugin en el navegador, lo que mejorará la disponibilidad de esta nueva aplicación.

La utilización del elemento `<canvas></canvas>` va ligada directamente a la utilización de JavaScript, este módulo está orientado al diseño web y no a la programación en JavaScript, aunque para el desarrollo de interfaces como vimos en la unidad anterior, son necesarios conocimientos básicos de este lenguaje de programación. En esta unidad veremos cuestiones básicas como crear objetos, transformarlos y realizar pequeñas animaciones. También podremos mostrar imágenes, texto y video dentro de una etiqueta `<canvas></canvas>`.

Como se ha comentado canvas es un elemento bitmap, es decir, es una imagen donde definimos los valores que tendrá cada uno de los píxeles que componen dicho elemento. De esto podemos deducir que canvas puede asemejarse a una etiqueta ``, con la diferencia que en lugar de mostrar una imagen, nosotros podemos crearla mediante código.

Canvas es un objeto más del DOM, igual que una etiqueta p o un título, todos los elementos que definamos dentro de un canvas no son accesibles desde el DOM, es decir, los elementos que hay dentro se tratan como un todo, la etiqueta canvas. Como veremos más adelante una de las grandes diferencias con SVG es que los elementos definidos dentro de este elemento si son accesibles y pueden aplicársele reglas de estilo.

Para finalizar este apartado vemos las ventajas que presenta canvas frente a flash

Canvas vs Flash.

- ✓ La mayoría de los navegadores han dejado de dar soporte a la tecnología Flash.
- ✓ Canvas es un elemento estándar de HTML5 y cualquier navegador (en su versión actual) debe ser capaz de implementarlo, mientras que Flash es una tecnología que requiere de la instalación de plugins en el navegador.
- ✓ Últimamente son varios los navegadores que no incorporan Flash, pero sí soportan Canvas.
- ✓ Además Canvas es libre y abierto, mientras que Flash es propiedad de la empresa Adobe.
- ✓ Si tenemos que utilizar un gráfico a varias resoluciones, la tecnología vectorial de Flash es más eficiente que los mapas de bits de Canvas. En cambio, para gran parte de gráficos para la web, los gráficos de Canvas se cargan más rápidamente que los de Flash.



Jurai Varma. Pinceles. ([Licencia Pixabay](#))

Para saber más

A estas alturas, sobra decir que harán falta algunos conocimientos básicos de Javascript para poder trabajar con canvas. Por ello, se recomiendan algunos tutoriales que pueden servir de apoyo en el desarrollo de esta unidad:

[Uniwebsidad.com](#)

[w3scholls.com](#)

[JavaScript.Info](#)

En el siguiente enlace se dejan las especificaciones sobre el elemento canvas de HTML.

[Canvas](#)

3.1.- Primeros pasos con Canvas.

1. Definir el elemento canvas.

El elemento `<canvas></canvas>` tiene solo dos atributos `width` y `height`. No es obligatorio definirlos junto a la etiqueta en el HTML (aunque en caso de no hacerlo tomarán valores por defecto), aunque sí recomendable. Cuando los atributos ancho y alto no están especificados, el lienzo se inicializará con 300 píxeles de ancho y 150 píxeles de alto. El elemento puede ser arbitrariamente redimensionado por CSS, pero durante el renderizado la imagen es escalada para ajustarse al tamaño de nuestra maqueta. Si el tamaño del CSS no respeta el ratio del canvas inicial, este aparecerá distorsionado.

Procedemos a definir nuestro primer elemento canvas, para ello definimos dentro del cuerpo de la página la etiqueta `<canvas></canvas>`, este espacio puede definirse como un espacio vacío en la página web (lienzo) en el cual serán mostrados los resultados de ejecutar los métodos provistos por la API.

```
<canvas id="ele_canvas" width="200" height="150">
    Su navegador no soporta el elemento canvas.
</canvas>
```

Con los atributos `width` (ancho) y `height` (alto) definimos el tamaño del lienzo en píxeles. Estos atributos son necesarios (no obligatorios) debido a que todo lo que sea dibujado sobre el elemento tendrá esos valores como referencia. El atributo `id`, como en otros casos, nos facilita el acceso al elemento desde el código Javascript. Además, se podrán colocar otros identificadores o clases, de forma opcional y mediante hojas de estilo (reglas) podremos definir el estilo del lienzo.

En nuestro ejemplo, la zona definida no tendrá contorno, así que será invisible. Utilizando hojas de estilo, se pueden establecer valores que permitan cambiar el aspecto de visualización del elemento. Mediante la siguiente regla CSS, damos borde y color de fondo al lienzo.

```
#ele_canvas
{
    border: solid 3px #490;
    background-color: #509;
}
```

Así se visualizará el elemento canvas por ahora:



Elaboración Propia.. Canvas ([CC BY-SA](#))

Eso es básicamente todo lo que el elemento canvas hace. Simplemente crea una caja vacía en la pantalla.

2. Dibujar en canvas mediante JavaScript

La API Canvas proporciona varias funciones ya listas para dibujar formas y trazados en un elemento `<canvas></canvas>`. En este caso, como primer ejemplo se va a comenzar dibujando un rectángulo sencillo. El código se puede insertar entre las etiquetas `<script></script>` en la cabecera de la página o en un archivo `.js` externo y, este sería el código completo de la página:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Documento HTML5</title>
    <style>
        #ele_canvas{
            border: solid 3px #4682B4;
            background-color: #FFFFFF0;
        }
    </style>

    <script>
        /* Función definida para crear un rectángulo en el canvas con id="ele_canvas"*/
        function dibujarRectangulo(){
```

```

var miCanvas=document.getElementById("ele_canvas");
var dibujo=miCanvas.getContext("2d");
dibujo.rect(10,10,150,100);
dibujo.fill();
}

/* Escucha que llama a la función dibujarRectangulo cuando se cargue la página*/
window.addEventListener('load',dibujarRectangulo);

</script>

</head>

<body>
<canvas id="ele_canvas" width="200" height="150">
    Su navegador no soporta el elemento canvas.
</canvas>

</body>
</html>

```

Analizando la función dibujarRectangulo() se tiene que:

- ✓ La primera línea declara la variable llamada miCanvas, que define un nuevo objeto de diseño en el elemento Canvas del documento en curso, que ha sido identificado a su vez mediante su id ele_canvas.
- ✓ La segunda línea declara un objeto llamado dibujo que, está definido en el elemento miCanvas que acabamos de crear y se usará para crear un gráfico en dos dimensiones mediante la sentencia: miCanvas.getContext("2d").
 - ◆ El método getContext() es el primer método que tenemos que llamar para dejar al elemento <canvas> listo para trabajar. Genera un contexto de dibujo que será asignado al lienzo. A través de la referencia que retorna podremos aplicar el resto de la API. El método puede tomar dos valores: 2d y 3d. Esto es, por supuesto, para ambientes de 2 dimensiones y 3 dimensiones.
- ✓ La tercera línea permite utilizar el método rect para definir un rectángulo en el objeto dibujo. Para definir el rectángulo se utilizan las coordenadas x (horizontal) e y (vertical), 10 píxeles y 10 píxeles, contados a partir del ángulo superior izquierdo del elemento canvas, con un ancho de 200 píxeles y una altura de 100 píxeles (dibujo.rect(10,10,200,100)).
- ✓ La cuarta línea solicita que se cree o se muestre el rectángulo definido mediante: dibujo.fill().
- ✓ Aunque, también se podría haber usado el método fillRect() para definir y dibujar el rectángulo en una sola sentencia.

Y, el resultado del código anterior se visualizaría de la siguiente forma:



Elaboración propia.. Rectángulo simple (CC BY-SA)

[Código ejemplo.](#) (zip - 0,57 KB)

Recuerda que para evitar que el elemento canvas aparezca distorsionado debemos definir el ancho y el alto en la etiqueta canvas del HTML y no usando CSS.

Autoevaluación

Indica V (verdadero) o F (falso), según corresponda en la siguiente afirmación.

Canvas permite dibujar en un documento html y actualizar dinámicamente dichos dibujos, por medio de scripts.

Verdadero Falso

Verdadero

Recomendación

Como material de apoyo, para dar los primeros pasos con canvas se recomiendan los siguientes tutoriales:

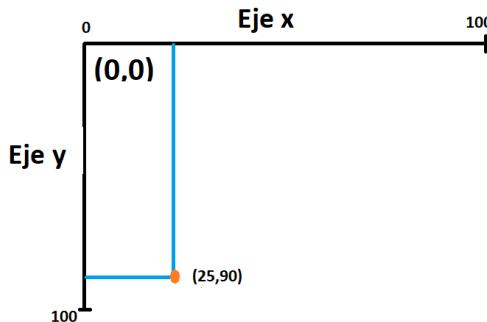
[w3schools.com](#)

[Aprendiendo canvas.](#)

[Guía sobre canvas.](#)

3.2.- Dibujar figuras.

Como ya se ha explicado anteriormente, canvas es un elemento sobre el que dibujamos por medio de sentencias JavaScript. En este apartado se verán qué funciones se pueden utilizar para dibujar ciertas figuras ya definidas como pueden ser rectángulos y circunferencias. Pero antes de continuar es conveniente saber cual es el origen de coordenadas del lienzo con el que vamos a trabajar.



Elaboración propia.. Origen de coordenadas. ([CC BY-NC-SA](#))

Como puede observarse el punto 0,0 estará situado en la parte superior izquierda, conforme nos desplazamos hacia la derecha avanzamos en el eje X y conforme descendemos avanzamos en el eje Y.

Ahora comenzaremos a dibujar figuras básicas.

✓ Rectángulos:

- ◆ rect(x, y, anchura, altura): esta función genera un rectángulo cuya esquina superior izquierda estará en el punto (x,y) con las dimensiones determinadas por anchura y altura. A diferencia de otras funciones, esta función no dibuja directamente el rectángulo sólo lo define.
- ◆ fillRect(x,y,anchura,altura): esta función sirve para dibujar rectángulos llenos de color. Sus parámetros tienen el mismo objetivo que en la función anterior; posición y dimensiones (ancho y alto).
- ◆ strokeRect(x,y,anchura,altura): esta función sirve para dibujar simplemente la silueta de un rectángulo, es decir, sólo su borde.
- ◆ clearRect(x,y,anchura,altura): esta función sirve para borrar áreas rectangulares de un canvas y hacerlas totalmente transparentes o sin contenido gráfico.

✓ Círculos u arcos:

para dibujar un arco o un círculo en la posición (x,y), se utiliza la función arc(x, y, radio, ángulo inicio, ángulo final, dirección) donde sus parámetros son:

- ◆ x: coordenada horizontal calculada a partir del centro del óvalo.
- ◆ y: coordenada vertical calculada a partir del centro del óvalo.
- ◆ radio: radio del óvalo.
- ◆ ángulo inicio: punto de partida del arco en radianes.
- ◆ ángulo final: punto de llegada del arco en radianes.
- ◆ dirección: valor booleano que indica que el arco será dibujado en el sentido de las agujas del reloj (valor true) o al contrario (valor false).

Un ejemplo sencillo de una circunferencia sería el resultado obtenido al ejecutar el siguiente código:

Código.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Documento HTML5</title>
    <style>
        #ele_canvas{
            border: solid 3px #4682B4;
            background: #FFFFFF;
        }
    </style>
<script>
    function dibujarCirculo()
    {
        var miCanvas=document.getElementById("ele_canvas");
        var dibujo=miCanvas.getContext("2d");
        dibujo.fillStyle="rgb(0,255,127)";
        dibujo.arc(50,50,40,0,2*Math.PI,true);
    }
</script>
```

```
dibujo.fill();
}
window.addEventListener('load',dibujarCirculo);

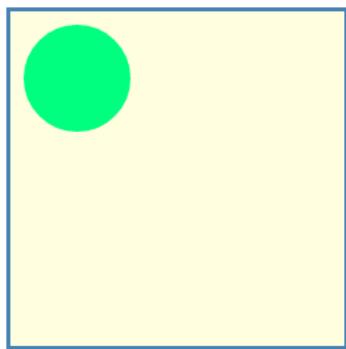
</script>

</head>

<body>
<canvas id="ele_canvas" width="250" height="250">
    Su navegador no soporta el elemento canvas.
</canvas>

</body>
</html>
```

Resultado del código.



Elaboración propia.. *Círculo vacío* (CC BY-SA)

[Código ejemplo.](#) (zip - 0,58 KB)

Autoevaluación

La función que pertenece al objeto contexto de un elemento canvas que sirve para dibujar rectángulos rellenos de color es:

strokeRect()

rect()

fillRect()

fill()

[Mostrar retroalimentación](#)

Solución

1. Incorrecto
2. Incorrecto

- 3. Correcto
- 4. Incorrecto

Recomendación

En los siguientes enlaces se proporciona más información sobre cómo dibujar distintos tipos de curvas en el lienzo <canvas>, como son las Curvas de Bézier y las Curvas Cuadráticas.

[Curvas de Bézier.](#)

[Curvas Cuadráticas.](#)

3.3.- Dibujar trazados.

Hasta ahora hemos visto ejemplos de cómo dibujar figuras como rectángulos o círculos y, es preciso destacar que, en canvas también podemos dibujar otras figuras no definidas explícitamente por funciones de la API. Para ello existen otras funciones que permiten definir cada uno de los puntos por los que pasa el trazado y luego pintar el color del mismo o su relleno. En este apartado se verán algunas de las funciones disponibles para hacer caminos o trazados:



Elaboración propia.. Triángulos (CC BY-SA)

- ✓ beginPath(): esta función le indica al contexto del canvas que se va a comenzar a dibujar un camino. No tiene ningún parámetro y por sí sola no hace ninguna acción visible. Se invoca en primer lugar, antes de comenzar a dibujar el camino o trazado incorporando segmentos.
- ✓ moveTo(x,y): esta función mueve el puntero imaginario a una posición específica donde comenzar o continuar el trazado. En realidad, no dibuja nada en sí, pero permite definir el primer punto del camino.
- ✓ closePath(): esta función sirve para cerrar un trazado o camino a su punto inicial, dibujando una línea recta desde el último punto hasta el punto de partida (definido con moveTo()). Tampoco recibe ningún parámetro.
- ✓ stroke(): esta función dibuja el contorno de un trazado a diferencia de fill() que rellena de color el trazado.
- ✓ fill(): este método dibuja el trazado como una figura sólida. Cuando se usa este método no es necesario cerrar el trazado con closePath(), el trazado es automáticamente cerrado con una línea recta trazada desde el punto final hasta el origen.
- ✓ lineTo(x,y): esta función dibuja una línea recta, desde la posición actual del puntero de dibujo, hasta el punto (x,y) que se indique como parámetro.

Un ejemplo sencillo de un trazado simple sería el siguiente (JavaScript):

Código

```
//Define el objeto que referencia al canvas.
var elemento=document.getElementById('ele_canvas');

// Crea el contexto del canvas.
dibujo=elemento.getContext('2d');

// Indica al contexto que se va a comenzar un trazado
dibujo.beginPath();

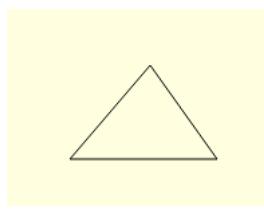
//Define las coordenadas de inicio.
dibujo.moveTo(150,50);

//Crea una línea desde el punto anterior hasta las coordenadas 200, 120
dibujo.lineTo(200,120);

//Crea una línea hasta las nuevas coordenadas
dibujo.lineTo(90,120);

//Cierra el camino creado anteriormente.
dibujo.closePath();<br /> // Dibujamos el contexto<br />dibujo.stroke();
```

Resultado del código.



Elaboración propia.. Trazado simple (CC BY-SA)

Si en vez de terminar el trazado mediante `closePath()` se hubiese utilizado `fill()`, mediante el siguiente código (JavaScript):

Código.

```
// creamos el objeto que referencia al canvas.  
var elemento=document.getElementById('ele_canvas');  
  
// creamos el contexto  
dibujo=elemento.getContext('2d');  
  
// inicializamos el dibujo  
dibujo.beginPath();  
  
//definimos punto de inicio.  
dibujo.moveTo(150,50);  
  
//línea hasta las coordenadas 200, 120  
dibujo.lineTo(200,120);  
  
//línea hasta las coordenadas 90,120  
dibujo.lineTo(90,120);<br /><br />  
// Cierra el trazado definido y rellena la figura.  
dibujo.fill();
```

Resultado del código



Elaboración propia. *Trazado simple relleno*
(CC BY-SA)

Recomendación

En los siguientes enlaces se proporcionan distintos ejemplos de cómo dibujar distintos trayectos y/o caminos en el lienzo `<canvas>`.

[Dibujar rectángulos y trayectos I.](#)

[Dibujar caminos.](#)

Dos métodos importantes que debemos conocer son los métodos:

- ✓ `save()`: permite guardar el contexto del canvas actual, es decir, posición en la que nos encontramos y características con las que estamos dibujando.
- ✓ `restore()`: si `save()` nos permite guardarlos `restore()` nos permite recuperar el contexto que hemos almacenado.

Dichos estados se guardan en forma de pila de tal forma que el último estado que se guarda es el primero que se recupera.

En el siguiente ejemplo se muestra como dibujar tres rombos uno dentro de otro, dibujado el primero se guarda el contexto, se dibuja un segundo y para el tercero se recupera el contexto guardado.

Código

```

<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <title>Documento HTML5</title>
        <style>
            #ele_canvas{
                border: solid 3px #4682B4;
                background-color: #FFFFE0;
            }
            #ele_canvas_2{
                border: solid 3px black;
                background-color: #FF0000;
            }
        </style>
        <script>
            /* Función definida para crear un rectángulo en el canvas con id="ele_canvas"*/
            function dibujar(){

                var elemento=document.getElementById('ele_canvas');
                var dibujo= elemento.getContext("2d");
                dibujo.beginPath(); //inicio de ruta
                dibujo.fillStyle="red"; //color de relleno rojo;
                dibujo.strokeStyle="black"; //color de contorno negro;
                dibujo.moveTo(100,50); //dibujar rombo
                dibujo.lineTo(150,150);
                dibujo.lineTo(100,250);
                dibujo.lineTo(50,150);
                dibujo.lineWidth="4"; // definimos el grosor del contorno a 2
                dibujo.closePath();
                dibujo.fill(); //rellenar de color
                dibujo.stroke(); //rellenar de color

                dibujo.save(); //guardamos contexto

                dibujo.beginPath(); //inicio de ruta
                dibujo.fillStyle="blue"; //color de relleno azul;
                dibujo.strokeStyle="green"; //color de contorno verde;
                dibujo.moveTo(100,60); //dibujar rombo
                dibujo.lineTo(140,150);
                dibujo.lineTo(100,240);
                dibujo.lineTo(60,150);
                dibujo.lineWidth="2";
                dibujo.closePath();
                dibujo.fill(); //rellenar de color
                dibujo.stroke(); //rellenar de color

                dibujo.restore(); //recuperamos contexto.
                // Nos cogerá los mismos estilos que teníamos.
                dibujo.beginPath(); //inicio de ruta
                dibujo.lineWidth="6"; // Se modifica la línea para que sea visible, aunque con el contexto se recupe
                dibujo.moveTo(100,80); //dibujar rombo
                dibujo.lineTo(120,150);
                dibujo.lineTo(100,220);
                dibujo.lineTo(80,150);
                dibujo.closePath();
                dibujo.stroke(); //borde de color seleccionado.
                dibujo.fill(); //rellenar de color

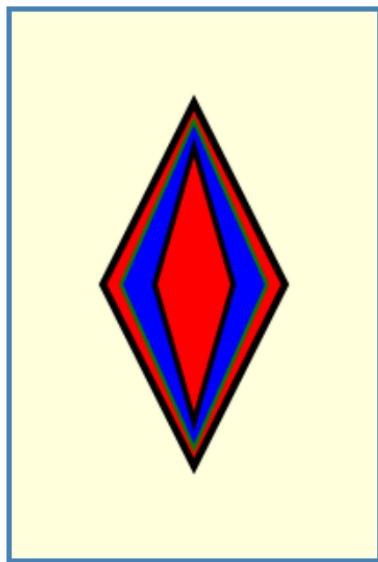
            }

            /* Escucha que llama a la función dibujarRectangulo cuando se cargue la página*/
            window.addEventListener('load',dibujar);
        </script>
    </head>
    <body>
        <canvas id="ele_canvas" width="200" height="300">
            Su navegador no soporta el elemento canvas.
        </canvas>
    </body>

```

```
</canvas>  
</body>  
</html>
```

Resultado del código.



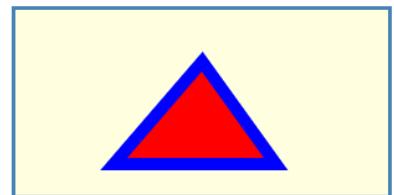
Elaboración propia.. *Rombos* ([CC BY-NC-SA](#))

[Código ejemplo.](#) (zip - 878 B)

3.4.- Colores y sombras.

Para trabajar con color y grosor de los diferentes trazados que vamos a realizar existen un conjunto de propiedades en <canvas></canvas> que nos ofrecen diferentes posibilidades, como son:

- ✓ `lineWidth`: esta propiedad nos permite definir el ancho de una línea en pixeles.
◆ `dibujo.lineWidth=19;`
- ✓ `fillStyle`: esta propiedad nos permite cambiar el color de relleno.
◆ `dibujo.fillStyle="#000099";`
- ✓ `strokeStyle`: esta propiedad nos permite cambiar el color del trazado que vamos a realizar.
◆ `dibujo.strokeStyle="#000099";`



Elaboración propia.. Colores ([CC BY-SA](#))

Además de colores, mediante la API Canvas también se pueden generar sombras mediante las siguientes cuatro propiedades:

- ✓ `shadowBlur`: para definir el difuminado de la sombra.
- ✓ `shadowColor`: para definir el color.
- ✓ `shadowOffsetX`: para definir la distancia horizontal (si no se indica nada, la sombra aparecerá alrededor del dibujo).
- ✓ `shadowOffsetY`: para definir la distancia vertical (si no se indica nada, la sombra aparecerá alrededor del dibujo).

Así por ejemplo, si aplicásemos sombras al círculo dibujado en el apartado 1.3.2. podríamos obtener el siguiente resultado, mediante el siguiente código:

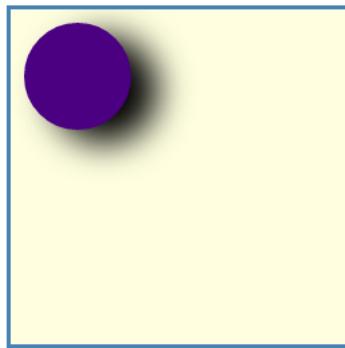
Código.

```
var miCanvas=document.getElementById("ele_canvas");
var dibujo=miCanvas.getContext("2d");

dibujo.fillStyle="#4B0082";
dibujo.shadowBlur=40;
dibujo.shadowColor="black";
dibujo.shadowOffsetX=20;
dibujo.shadowOffsetY=10;

dibujo.arc(50,50,40,0,2*Math.PI,true);
dibujo.fill();
```

Resultado del código.



Elaboración propia.. Círculo con sombras. ([CC BY-NC-SA](#))

[Código ejemplo.](#) (zip - 2,15 KB)

Autoevaluación

Indica V (Verdadero) o F (falso), según corresponda en la siguiente afirmación.

La línea de código context.strokeStyle = '#ff0000'; establece el color de relleno que tendrá una figura que se dibuje en el lienzo canvas.

- Verdadero Falso

Falso

3.5.- Añadir texto.

También es posible insertar texto en el canvas con el método `fillText()`, para crear un texto de relleno, y `strokeText()`, para crear el borde del texto.

- ✓ La propiedad `textBaseline` permite definir la línea de base que prefiramos. Los posibles valores son: `top`, `hanging`, `middle`, `alphabetic`, `ideographic` y `bottom`.
- ✓ La propiedad `textAlign` permite definir la alineación del texto. Existen varios valores posibles: `start`(comienzo), `end` (final), `left` (izquierda), `right` (derecha) y `center` (centro).
- ✓ La propiedad `font` permite definir la tipografía de caracteres que queramos utilizar y tiene una sintaxis similar a la propiedad `font` de CSS, y acepta los mismos valores.

A continuación dejamos un ejemplo y el resultado del mismo.



Gerd Altman. Texto ([Licencia pixabay](#))

Código.

```
<!DOCTYPE HTML>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <style>
        #lienzo_texto{
            border: dashed 2px #006400;
        }
    </style>
    <script>
        function dibujarTexto(){
            var miCanvas=document.getElementById("lienzo_texto");
            var dibTexto=miCanvas.getContext("2d");
            dibTexto.textBaseline="ideographic";
            dibTexto.textAlign="center";
            dibTexto.font="2em Nueva";
            dibTexto.fillStyle="purple";
            dibTexto.fillText("TEXTO DE PRUEBA",150,75);
        }
        window.addEventListener('load',dibujarTexto);
    </script>
</head>
<body>
    <canvas id="lienzo_texto" width="300" height="100"></canvas>
</body>
</html>
```

Resultado del código.



Elaboración propia. [Texto en Canvas \(CC BY-SA\)](#)

[Código ejemplo.](#) (zip - 0,49 KB)

Los métodos que están disponibles para dibujar texto en el lienzo son los siguientes:

- ✓ `strokeText(texto, x, y)`: este método dibujará el texto especificado en la posición x,y como una figura vacía (solo los contornos). Puede también incluir un cuarto valor para declarar el tamaño máximo. Si el texto es más extenso que este último valor, será encogido para caber dentro del espacio establecido.
- ✓ `fillText(texto, x, y)`: este método es similar al método anterior excepto que esta vez el texto dibujado será sólido.

3.6.- Aplicar un degradado.

De la misma forma que en CSS3, los gradientes en la CSS Canvas pueden ser lineales o radiales, y pueden incluir puntos de terminación para combinar colores:

- ✓ `createLinearGradient(x0, y0, x1, y1)` Este método crea un objeto que luego será usado para aplicar un gradiente lineal al lienzo. Donde sus parámetros son:

- ◆ `x0`: coordenada x del punto inicial del degradado.
- ◆ `y0`: coordenada y del punto inicial del degradado.
- ◆ `x1`: coordenada x del punto final del degradado.
- ◆ `y1`: coordenada y del punto final del degradado.



[donations welcome.](#) Degrado. ([Licencia pixabay](#))

- ✓ `createRadialGradient(x1, y1, r1, x2, y2, r2)` Este método crea un objeto que luego será usado para aplicar un gradiente circular o radial al lienzo usando dos círculos. Los valores representan la posición del centro de cada círculo y sus radios.

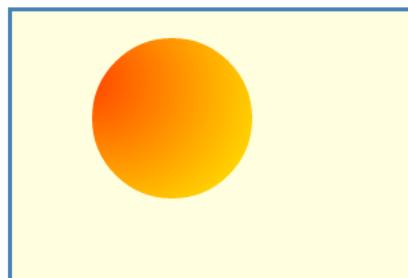
- ◆ `x1`: coordenada x del círculo inicial del degradado.
- ◆ `y1`: coordenada y del círculo inicial del degradado.
- ◆ `r1`: radio del círculo desde donde comenzar.
- ◆ `x2`: coordenada x del círculo final del degradado.
- ◆ `y2`: coordenada y del círculo final del degradado.
- ◆ `r2`: radio del círculo desde donde comenzar.

- ✓ `addColorStop(posición, color)` Este método especifica los colores a ser usados por el gradiente. El atributo posición es un valor entre 0.0 y 1.0 que determina dónde la degradación comenzará para ese color en particular.

Mostramos como aplicar un degradado al círculo anterior mediante el siguiente código:

```
function dibujarCirculo()
{
    var miCanvas=document.getElementById("ele_canvas");
    var dibujo=miCanvas.getContext("2d");
    var grd=dibujo.createRadialGradient(50,50,5,90,60,100);
        grd.addColorStop(0,"OrangeRed");
        grd.addColorStop(1,"Gold");
    dibujo.fillStyle=grd;
    dibujo.arc(120,80,60,0,2*Math.PI,true);
    dibujo.fill();
}
window.addEventListener('load',dibujarCirculo);
```

Y el resultado obtenido sería el siguiente:



Elaboración propia. Circulo con degradado. ([CC BY-NC-SA](#))

[Códigos ejemplo.](#) (zip - 1,35 KB)

3.7.- Las transformaciones y manipulaciones de objetos.

Al igual que vimos con las transformaciones de elementos en CSS, la API canvas también permite "transformar" los objetos dibujados con los métodos `translate(x,y)` (para desplazar un objeto), `rotate(angulo)` (para aplicar una rotación a un objeto), y `scale(x,y)` (para redimensionar un objeto).

El elemento canvas permite "transformar" los objetos dibujados en el lienzo con los métodos:

- ✓ `translate(x,y)`: Este método de transformación es usado para mover el origen del lienzo. Es decir, permite mover el punto 0,0 a una posición específica para usar el origen como referencia para nuestros dibujos o para aplicar otras transformaciones.

Siguiendo con los ejemplos de dibujar círculos, podríamos utilizar este método para dibujar dos círculos en el lienzo mediante el siguiente código:

```
function dibujarCirculo()
{
    var miCanvas=document.getElementById("ele_canvas");
    var dibujo=miCanvas.getContext("2d");

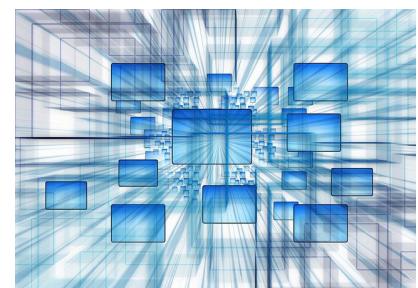
    dibujo.fillStyle="gold";

    dibujo.arc(120,80,30,0,2*Math.PI,true);
    dibujo.fill();

    dibujo.translate(90,70);
    dibujo.arc(120,80,40,0,2*Math.PI,true);
    dibujo.fill();

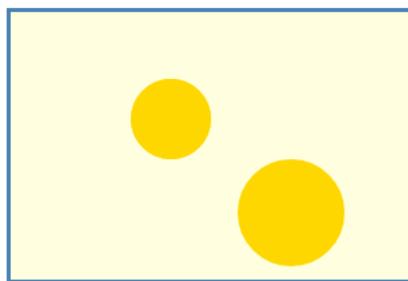
}

window.addEventListener('load',dibujarCirculo);
```



Gerd Altmann, Transformación (Licencia Pixabay)

Y, el resultado visual sería el siguiente:



Elaboración propia. Ejemplo translate (CC BY-SA)

- ✓ `rotate(angulo)`: Este método de transformación rotará el lienzo alrededor del origen tantos ángulos como sean especificados. El ángulo de rotación se produce en radianes y en el sentido de las agujas del reloj. Para poder indicarlo en grados, podemos utilizar la siguiente fórmula: $\text{grado} * \text{Math.PI} / 180$. El centro de rotación es el origen del elemento canvas, posición 0,0
- ✓ `scale(x, y)`: Este método de transformación incrementa o disminuye las unidades de la grilla para reducir o ampliar todo lo que esté dibujado en el lienzo.

A continuación dejamos otro ejemplo con traslación y rotación

Código.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <title>Documento HTML5</title>
        <style>
```

```

#ele_canvas{
    border: solid 3px #4682B4;
    background-color: aqua;
}

</style>
<script>
/* Función definida para crear un rectángulo en el canvas con id="ele_canvas"*/
function dibujar(){

    var miCanvas=document.getElementById("ele_canvas");
    var dibujo=miCanvas.getContext("2d");

    dibujo.fillStyle="blue";

        dibujo.fillRect(10, 10, 100, 50);
        dibujo.translate(70, 70);
        dibujo.fillRect(10, 10, 100, 50);

    dibujo.fillStyle="red";

        dibujo.translate(70, 70); //realizamos una translación
        dibujo.rotate(15 * Math.PI / 180); //realizamos una rotación de 15º
        dibujo.fillRect(10, 10, 100, 50);

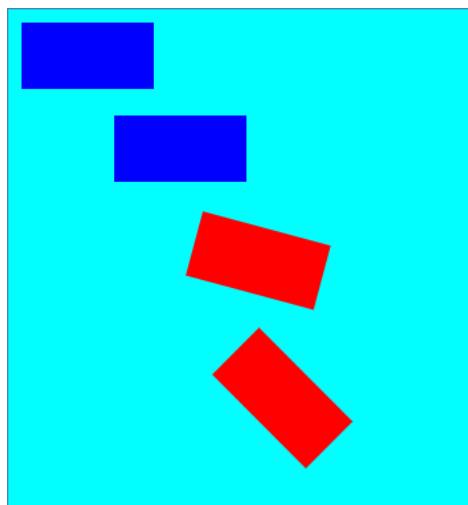
        dibujo.translate(70, 70); //realizamos una translación
        dibujo.rotate(30 * Math.PI / 180); //realizamos una rotación de 30º más los 15º de antes ser
        dibujo.fillRect(10, 10, 100, 50);

}

/* Escucha que llama a la función dibujarRectangulo cuando se cargue la página*/
window.addEventListener('load',dibujar);
</script>
</head>
<body>
    <canvas id="ele_canvas" width="500" height="500">
        Su navegador no soporta el elemento canvas.
    </canvas>
</body>
</html>

```

Resultado del código.



Elaboración propia.. *Transformaciones* ([CC BY-NC-SA](#))

Se puede proponer interactividad a los usuarios con los objetos dibujados, se podrían llegar incluso a [crear juegos](#). Pero ahí nos estaríamos alejando de nuestro ámbito de aplicación, es decir el diseño web, y nos estaríamos introduciendo en el mundo de la programación en JavaScript.

Autoevaluación

Tras ejecutar el siguiente código JavaScript, se dibujará:

```
var miCanvas=document.getElementById("ele_canvas");
var dibujo=miCanvas.getContext("2d");
    dibujo.fillRect(10,10,100,50);
    dibujo.translate(70,70);
    dibujo.fillRect(10,10,100,50);
```

- Se dibujará un único rectángulo situado en las coordenadas (10,10).

- Se dibujarán dos rectángulos: uno en las coordenadas (10,10) y otro en las coordenadas (70,70).

- Se dibujarán dos rectángulos: uno en las coordenadas (10,10), y otro que se dibujará tras redefinir el origen de coordenadas en los puntos (70,70)

- Se dibujarán dos rectángulos, ambos rellenos del color negro.

[Mostrar retroalimentación](#)

Solución

1. Incorrecto
2. Incorrecto
3. Correcto
4. Correcto

Para saber más

A continuación se deja un par de hojas resumen sobre canvas.

[Canvas Cheat Sheet](#)

[Hoja Resumen sobre métodos y propiedades de canvas](#)

3.8.- Procesando imágenes.

Cuando se dibuja en el lienzo del elemento canvas se puede importar y mostrar directamente el contenido de archivos gráficos externos, es decir, usar imágenes GIF, JPG o PNG dentro de los dibujos realizados con canvas.

Para dibujar una imagen en el lienzo se utiliza el método drawImage(), que pertenece al objeto contexto de la API de canvas, con la siguiente sintaxis:

```
drawImage(objeto_imagen, x,y)
```

Este método puede recibir un número de valores que producen diferentes resultados y, las posibilidades son:

- ✓ Dibujar la imagen: drawImage(imagen, x, y);
 - ◆ Primer parámetro: contiene la referencia al objeto imagen.
 - ◆ Segundo parámetro: posición en el eje X en el que queremos dibujar la imagen.
 - ◆ Tercer parámetro: posición en el eje Y en el que queremos dibujar la imagen.
- ✓ Dibujar la imagen con las propiedades re-escaladas: drawImage(imagen, x, y, ancho, alto);
 - ◆ Primer, segundo y tercer parámetro: igual que el método anterior.
 - ◆ Cuarto parámetro "ancho": el ancho que queremos que tenga la imagen.
 - ◆ Quinto parámetro "alto": el alto que queremos que tenga la imagen.
- ✓ Dibujar partes de una imagen: drawImage(imagen, x, y, width, height, dx, dy, dWidth, dHeight); este es el método más completo, ya que nos permite crear animaciones.
 - ◆ Primer parámetro: igual que los anteriores.
 - ◆ Segundo parámetro: posición X dentro de nuestra imagen en la que queremos empezar la selección a recortar.
 - ◆ Tercer parámetro: posición Y dentro de nuestra imagen en la que queremos empezar la selección a recortar.
 - ◆ Cuarto parámetro: ancho de la selección a recortar.
 - ◆ Quinto parámetro: alto de la selección a recortar.
 - ◆ Sexto parámetro: posición en el eje X que queremos dibujar nuestra imagen dentro del elemento canvas.
 - ◆ Séptimo parámetro: posición en el eje Y que queremos dibujar nuestra imagen dentro del elemento canvas.
 - ◆ Octavo parámetro: ancho que queremos que tenga la imagen recortada en el canvas.
 - ◆ Noveno parámetro: alto que queremos que tenga la imagen recortada en el canvas.



Darkmoon_Art. Imagen (Licencia Pixabay)

En cada caso, el primer atributo puede ser una referencia a una imagen en el mismo documento generada por métodos como getElementById(), o creando un nuevo objeto imagen usando métodos regulares de JavaScript. No es posible usar una URL o cargar un archivo desde una fuente externa directamente con este método.

En el siguiente código podemos ver como se utiliza el método drawImage(). Se muestran 4 imágenes:

- ✓ La primera imagen solo pinta una zona de la imagen original en el canvas, ya que las dimensiones de la imagen (4624x2080) son muy superiores al canvas definido (600x600).
- ✓ La segunda imagen se muestra redimensionada para mostrarse en una zona del canvas de 100x100 píxeles.
- ✓ La tercera se coge una zona de la imagen inicial de 1000x2000 píxeles comenzando en la posición 300x500 y redimensionándola a 200x200 píxeles.
- ✓ La cuarta es similar a la segunda pero desplazada e invertida.

Código.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo canvas con imágenes</title>
    <style>
        #ele_canvas{
            border: solid 3px #4682B4;
            background-color: #FFFFFF;
        }
    </style>
    <script>

        function dibujarRectangulo(){
            var miCanvas=document.getElementById("ele_canvas");
            var dibujo=miCanvas.getContext("2d");
            var imagen=new Image();
                imagen.src = "foto.jpg";
                imagen.onload = function() {
                    /* Cargaríamos la imagen en el canvas*/
                    dibujo.drawImage(this, 0, 0);
                /* Cargaríamos la imagen rescalándola a unas dimensiones de 100 x 100 en la posición 0,400*/
                    dibujo.drawImage(this, 0, 400,100,100);
                }
        }

    </script>
</head>
<body>
    <div id="ele_canvas"></div>
</body>
```

```
// Seleccionamos una parte de la imagen desde la posición 0,0 a la 2000,2000 y la m
// Posición 300,500 de nuestra imagen original,
    // Cogemos 1000 de ancho y 2000 de alto
    // Mostramos en la posición del canvas 100,100
    // El trozo de imagen redimensionado a 200 x 200 pixeles
    dibujo.drawImage(this, 300,500,1000,2000,100,100,200,200);

    /* posicionamos la imagen original invertida y desplazada */

    dibujo.translate(300,500);
    dibujo.scale(1,-1);
    dibujo.drawImage(this, 0, 0,100,100);
}
}

/* Escucha que llama a la función dibujarRectangulo cuando se cargue la página*/
window.addEventListener('load',dibujarRectangulo);

</script>

</head>
<body>
<canvas id="ele_canvas" width="600" height="600">
    Su navegador no soporta el elemento canvas.
</canvas>
</body>
</html>
```

Resultado del código.



Elaboración propia. *Imágenes en canvas*. ([CC BY-SA](#))

[Códigos ejemplo.](#) (zip - 4,29 MB)

Existen otros métodos para procesar imágenes en esta [API](#), aunque en este caso no son objeto de nuestro estudio.

3.9.- Animaciones en el lienzo.

Las animaciones son creadas por código JavaScript convencional. Es decir, no se dispone de métodos concretos para animar figuras en el lienzo, y tampoco existe un procedimiento predeterminado para hacerlo. Básicamente, la técnica consiste en borrar el área del lienzo que se pretenda animar, dibujar las figuras y repetir el proceso una y otra vez. Es por esto que, en juegos o aplicaciones que requieren grandes cantidades de objetos a ser animados, es mejor usar imágenes en lugar de figuras construidas con trazados complejos.

Aunque realizar un juego sería algo más complejo, con los conocimientos actuales si que podríamos realizar pequeñas animaciones pintando y borrando objetos en nuestro canvas, para ello solo tendremos que añadir a lo que ya hemos visto las funciones de JavaScript setInterval() y clearInterval().

- ✓ setInterval(nombre_funcion, intervalo_en_milisegundos): esta función ejecutará una función o un código de forma indefinida con un intervalo de tiempo que nosotros indicamos (segundo parámetro) expresado en milisegundos. Cuando comienza a ejecutarse setInterval se devuelve un identificador, que tendremos que tener en cuenta para posteriormente parar esta ejecución indefinida.
- ✓ clearInterval: nos permitirá detener la función que se ejecuta de forma indefinida lanzada con setInterval.

Podemos llamar a la función de diferentes formas:

- ✓ setInterval(pintar,300); llamará a la función pintar cada 300 milisegundos. Si la función pintar tuviera parámetros, tendríamos que ponerlos separados por comas después del intervalo. Ej.: setInterval(pintar,300,x,y); En este caso llamará a la función pintar la cual debe tener dos parámetros.
- ✓ SetInterval(funcion(){codigo_a_ejecutar},300); en esta segunda forma de llamada, el código que hay entre los corchetes {} se ejecutará cada 300 milisegundos. Esta forma es la que hemos utilizado para implementar el ejemplo que a continuación se muestra.

Para parar esta función que se ejecuta de forma indefinida, tendremos que identificar, para eso la igualaremos a una variable de la siguiente forma, por ejemplo:

```
var identificador_interval=setInterval(pintar,300);
```

El valor que devuelve la ejecución indefinida se almacena en la variable identificador_interval, el valor que está aquí almacenado nos permitirá detener la ejecución indefinida de la siguiente forma:

```
clearInterval(identificador_interval);
```

En los siguientes ejemplos podemos ver como se crean tres objetos canvas, en los dos objetos canvas más grandes se realizarán animaciones y en el canvas pequeño con fondo rojo, nos permitirá comenzar la animación que se ejecuta en el canvas primero. Así vemos como se puede comenzar una animación de diferentes formas.

En los siguientes ejemplos:

- ✓ El canvas primero se comenzará a ejecutar cuando hagamos clic en el canvas pequeño coloreado de color rojo. En el se muestra el movimiento de un círculo que cambia de color y tamaño.
- ✓ En el segundo canvas, comienza a ejecutarse al cargarse la página, en el se muestra una barra de progreso con un porcentaje.

Código.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Animaciones con canvas</title>
    <style>
        /* Definimos el fondo y el borde de nuestros canvas*/
        #canvas_animacion,#canvas_grafica{
            border: solid 3px #4682B4;
            background-color:darkseagreen;
        }
        #canvas_inicio{
            border: solid 3px #4682B4;
            background-color:red;
        }
    </style>

    <script>

        window.onload = function() {
            // Al cargar la página dibujamos el botón play

            // Dibujamos el botón de play
            /* Inicializamos nuestro canvas */

    
```



[Darkmoon_Art](#). Animación en el lienzo ([Licencia Pixabay](#))

```

var micanvas_02=document.getElementById("canvas_inicio");
var boton=micanvas_02.getContext("2d");
boton.beginPath();
boton.fillStyle="green";
boton.moveTo(5,5);
boton.lineTo(45,25);
boton.lineTo(5,45);
boton.closePath();
boton.fill();
}

/*Definimos las funciones que nos permitiran dibujar en nuestros canvas */

/* Con esta función vamos a dotar de movimiento a un círculo*/
function movimiento(){

/* Inicializamso nuestro canvas */
var micanvas_01=document.getElementById("canvas_animacion");
var dibujo=micanvas_01.getContext("2d");
var i;

var x=75;
var y=75;
var r=50;
/* Obtenemos las dimensiones de nuestro canvas ancho y alto.*/
var ancho=micanvas_01.width;
var alto=micanvas_01.height;

/* Definimos una función que nos dibujará en el canvas dibujo, en la posicion_x,posicion_y una cincunferencia con un rad
function dibuja(dibujo,posicion_x,posicion_y,radio){
    /* Iniciamos la ruta */
    dibujo.beginPath();
    /*Dibujamos la circunferencia*/
    dibujo.arc(posicion_x,posicion_y,radio,0,2*Math.PI,true);
    /* Cerramos la ruta, aunque con fill no sería necesario ya que lo rellenaría.*/
    dibujo.closePath();
    /* Rellenamos*/
    dibujo.fill();
}

/* Utilizamos la función setInterval para dibujar una circuferencia cada x milisegundos, en nuestro caso 40.*/
var mip=setInterval(function(){
    /* Estas variables nos van a permitir desplazar la figuar por el eje x e y*/
    x=x+1;
    y=y+1;

    /* Borraremos el canvas para posteriormente dibujar una nueva fijura en la posición x e y*/
    /* como estas van cambiando, se producirá así el movimiento. */

    dibujo.clearRect(0,0, ancho, alto);

    /* Llamamos a la función que nos dibujará la circunferencia*/
    /* x será la posición en el eje x del canvas */
    /* y será la posición en el eje y del canvas */
    /* r será el radio de la circunferencia.*/

    dibuja(dibujo,x,y,r);

    /* Las líneas que vienen a continuación nos permiten cambiar el color y el radio de la circunferencia*/
    if(x>=0 && x<=125){
        dibujo.fillStyle="rgb(255,15,127)";
        r=r-0.5;
    }
    if(x>=126 && x<=250){
        dibujo.fillStyle="rgb(0,215,227)";
        r=r+0.5;
    }
    /* Paramos la animación cuando x llega a 200*/
    if (x>=200){
        clearInterval(mip);
    }
}
,40);
}

function desplazamiento(){
    var micanvas_03=document.getElementById("canvas_grafica");
}

```

```

var barra=micanvas_03.getContext("2d");
var i;
var x=10;
var ancho=micanvas_03.width;
var alto=micanvas_03.height;
var miint;

/* Esta función nos dibuja una rectángulo y nos muestra un número en pantalla*/
function dibuja_barra(barra,a){
    barra.beginPath();
    barra.fillStyle = "turquoise";
    barra.fillRect(10,10,a,100);
    barra.closePath();
    barra.textAlign="center";
    barra.font="2em Arial";
    barra.fillStyle="red";
    barra.fillText(parseInt(a/2.8)+"%",150,75);
}
/* Iniciamos la función setInterva llamará a la función que pinta el rectángulo hasta que la variable x llegue a 280*/
/* En ese momento paramos la repetición de la llamada a la función con clearInterval*/

/* Introducimos el identificador en la variable miint, que nos permitirá posteriormente detener la llamada a la función*/
miint=setInterval(function(){
    x=x+10;
    /* Borramos solo la zona donde se escribe el texto*/
    /* Aunque podríamos borrar todo el canvas*/
    barra.clearRect(120,50, 200, 100);
    dibuja_barra(barra,x);

    if (x>=280){
        clearInterval(miint);
    }
},40);

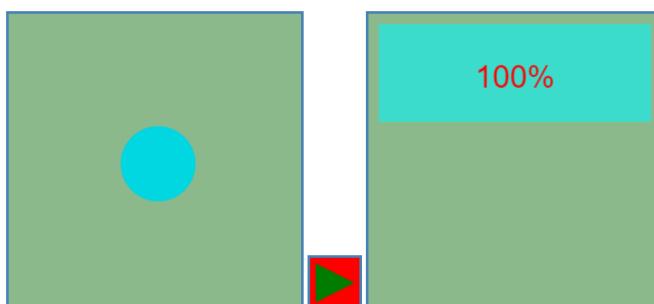
}

/* Se ejecuta el canvas al cargar la página*/
window.addEventListener('load',desplazamiento);
</script>
</head>
<body>
    <canvas id="canvas_animacion" width="300" height="300">
        Su navegador no soporta el elemento canvas.
    </canvas>

    <!-- Al hacer click en el canvas de rojo se activa la animación del canvas primero, la circunferencia moviéndose -->
    <canvas id="canvas_inicio" width="50" height="50" onclick="movimiento()">
        Su navegador no soporta el elemento canvas.
    </canvas>
    <canvas id="canvas_grafica" width="300" height="300">
        Su navegador no soporta el elemento canvas.
    </canvas>
</body>
</html>

```

Resultado del código.



Elaboración propia.. Animaciones ([CC BY-SA](#))

[Código ejemplo.](#) (zip - 1,98 KB)

Para saber más

En el siguiente enlace podemos ver más ejemplos de como animar un objeto.

[Animación de un objeto.](#)[Animación con canvas](#)

Autoevaluación

La regla @keyframe junto con la propiedad animation , permite definir las animaciones en la API canvas.

- Verdadero Falso

Falso

La regla @keyframe y la propiedad animation se utiliza para definir animaciones en CSS

3.10.- Vídeos en el lienzo.

Al igual que ocurre con las animaciones, no hay ningún método especial para mostrar vídeo en el elemento <canvas></canvas>. La única opción de hacerlo es tomando cada cuadro del vídeo desde el elemento <video> y dibujarlo como una imagen en el lienzo usando drawImage(). Así que básicamente, el procesamiento de vídeo en el lienzo es hecho con la combinación de técnicas ya estudiadas.



[200 Degrees](#), Video ([Licencia Pixabay](#))

Para saber más

En el siguiente enlace podemos ver como tratar un vídeo con canvas.

[Vídeos en Canvas](#)

4.- SVG (Gráficos Vectoriales Escalables).

SVG (Gráficos Vectoriales Escalables) es un tipo de imágenes que como ya sabemos nos permiten hacer dibujos, banners, gráficos, etcétera, tanto estáticos como animados, mediante gráficos vectoriales. Es decir, permite hacer las mismas cosas que canvas, aunque de forma diferente, con canvas los dibujos se crean mediante píxeles (mapa de bits) y con SVG son creados mediante gráficos vectoriales.

SVG es un lenguaje basado en XML, que permite definir imágenes vectoriales. Es un lenguaje de marcas similar a HTML, excepto que posee muchos elementos diferentes para definir las formas que deseas que aparezcan en nuestra imagen y los efectos que se desean aplicar a estas formas.

Los gráficos SVG (en HTML) acaban siendo representados en el navegador como elementos que forman parte de DOM, por lo que podemos animarlos también con CSS.

Características de SVG.

- ✓ Se utiliza para definir los gráficos basados en vectores para la web.
- ✓ Es fácil de usar, pues define los gráficos en formato XML, pudiendo ser indexado por buscadores.
- ✓ Los gráficos SVG no pierden calidad si se cambia el tamaño del elemento.
- ✓ Cada elemento y cada atributo en archivos SVG pueden ser animados.
- ✓ SVG es una recomendación del W3C.
- ✓ Se muestra de forma progresiva, no tenemos que esperar a que se descargue todo el archivo.
- ✓ Se le puede aplicar estilos CSS y pueden ser modificados mediante JavaScript.

Ventajas de SVG.

Las imágenes SVG se pueden:

- ✓ crear y editar con cualquier editor de texto.
- ✓ buscar, indexar en una web.
- ✓ optimizar y comprimir.
- ✓ imprimir con alta calidad en cualquier resolución.

Compatibilidad con los navegadores.

SVG es compatible con todos los navegadores en sus versiones modernas



Gerd Altmann, Navegadores ([Licencia Pixabay](#))

Para saber más

A continuación se muestra un enlace donde se puede ver una línea de tiempo del grupo de trabajo de SVG

[Línea del tiempo del grupo de trabajo SVG dentro de W3C](#)

4.1.- Primeros pasos con SVG.

HTML5, permite incrustar directamente imágenes en SVG, así dispone de una serie de etiquetas y atributos para introducir diferentes tipos de elementos, como figuras geométricas, textos, trazados e incluso animarlos si así lo deseamos.

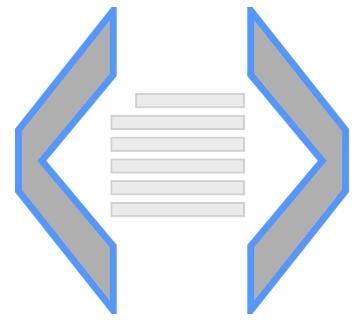
Para crear un elemento SVG en una página web, utilizaremos la etiqueta <svg></svg>. En el siguiente ejemplo se crea un objeto SVG con unas dimensiones 150px de ancho por 150px de alto, que además puede ser referenciado con el identificador (id) dibujo (id="dibujo")

```
<svg id="dibujo" width=150 height=150>
</svg>
```

En la etiqueta <svg></svg> de nuestro HTML es necesario definir los atributos `width` y `height` (ancho y alto), aunque también es posible especificarlo mediante CSS. A la etiqueta <svg></svg> y los elementos definidos dentro de ella se le puede aplicar reglas de estilo CSS y pueden ser modificados (añadir, eliminar y modificar) mediante JavaScript.

Una forma efectiva y de más posibilidades que tenemos con SVG es definir los gráficos o contenido gráfico que queramos por medio de código con formato XML, para ello la declaración es la siguiente:

```
<svg id="dibujo" width=150 height=150 xmlns="http://www.w3.org/2000/svg" version="1.1">
</svg>
```



khaase, Código XML (Licencia Pixabay)

También es posible incluir todo este código en un archivo aparte con la extensión .svg, y para incluir después este archivo en la página podemos hacerlo de varias maneras:

- ✓ Con la etiqueta `img`, referenciando el archivo desde el atributo `src`.
 - ✓ Como la etiqueta `<iframe>`, referenciando el archivo mediante el atributo `src`.
 - ✓ Con la etiqueta `<object>`, referenciando el archivo mediante el atributo `src`.
 - ✓ Con la etiqueta `<embed>`, referenciando el archivo mediante el atributo `src`.
- A continuación se muestra el código para cada una de estas opciones:

```
<br /><br /><iframe src="dibujo.svg" width="200" height="200"></iframe><br />
```

Como puede apreciarse, en cualquiera de los casos es necesario incluir la ruta del archivo, y los atributos `width` y `height` de la ventana donde se verá el elemento SVG.

Para saber más

SVG 2.0 se encuentra actualmente en desarrollo, en el siguiente enlace tenemos la [recomendación 1.1 \(segunda edición\)](#).
[Grupo de trabajo sobre SVG del W3C](#).

4.2.- Dibujar figuras básicas.

Para crear distintas formas con los gráficos SVG, existen una serie de etiquetas XML. En la siguiente tabla se muestran algunas de estas etiquetas:

Etiquetas XML para crear figuras en SVG.

Elemento	Descripción
line	Crea una línea simple.
polyline	Define formas construidas a partir de múltiples definiciones de línea.
rect	Crea un rectángulo.
circle	Crea un círculo.
ellipse	Crea una elipse.
polygon	Crea un polígono.
path	Permite definir rutas arbitrarias.



Alicja. Figuras geométricas. ([Licencia Pixabay](#).)

Dibujando círculos:

Para dibujar un círculo existen una serie de elementos que son imprescindibles:

- ◆ circle: mediante este atributo se indica que se dibujará un círculo.
- ◆ cx=n cy=n: los atributos cx y cy indican las coordenadas del centro del círculo.
- ◆ r=n: este atributo indica la longitud del radio de la circunferencia medida en píxeles.

El resto de atributos definen el estilo del círculo, y se pueden utilizar con otras figuras.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Gráficos con SVG HTML5</title>
  </head>
  <body>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="400" height="400">
      <circle cx=90 cy=90 r=50 stroke="OrangeRed" stroke-width=2 fill="Coral" />
    </svg>
  </body>
</html><br />
```



Elaboración propia. Círculo SVG. ([CC BY-SA](#))

Es imprescindible resaltar que en todo contenedor SVG el origen de coordenadas, en principio, estará en la esquina superior izquierda con el valor 0,0. desde ahí se miden los píxeles para trazar los elementos del dibujo.

Dibujando rectángulos:

Para dibujar rectángulos, se debe especificar tanto el ancho como el alto. El siguiente código sería un ejemplo sencillo:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="300" height="300">
  <rect width="200" height="100" fill="YellowGreen" stroke="Green" stroke-width=2 />
</svg>
```

El resultado visual sería el siguiente:

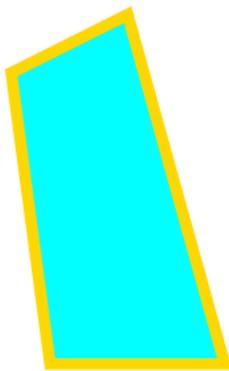


Elaboración propia. Rectángulo con SVG
(CC BY-SA)

Dibujando polígonos:

A continuación mostramos un ejemplo con `polygon`, en esta etiqueta tenemos que poner un conjunto de coordenadas x,y (como mínimo tres) tras poner el último punto el polígono se cierra con el primer punto que hemos definido. A continuación mostramos un código de un polígono irregular.

```
<svg viewBox="-10 -10 310 310" xmlns="http://www.w3.org/2000/svg" version="1.1" width="300" height="300">
  <polygon points="200,10 250,190 160,190 140,40" style="fill:aqua;stroke:gold;stroke-width:6" />
</svg>
```



Elaboración propia. Polígono irregular (CC BY-SA)

Como se puede apreciar definimos también el relleno, el borde y el ancho de este.

Los objetos que hemos definido se empiezan a posicionar en la posición 0,0, si queremos posicionar en otra posición nuestras figuras tendremos que utilizar los atributos x e y, así por ejemplo este rectángulo se posicionará en la posición del eje x 200 y en el eje y en la posición 300.

```
<rect x="200" y="300" width="200" height="100" fill="YellowGreen" stroke="Green" stroke-width=2 />
```

A continuación se deja un ejemplo donde se muestran los diferentes elementos vistos en diferentes etiquetas `<svg></svg>` y todos ellos en una misma etiqueta `<svg></svg>`.

viewBox:

En el ejemplo anterior hemos utilizado un nuevo parámetro `viewBox`, mientras que `width` y `height` definen las dimensiones del elemento SVG que se verá en nuestra página, es decir, será el espacio visible, `viewBox`, nos va a delimitar la dimensiones de nuestro espacio de trabajo. Así podemos definir un espacio de trabajo mayor (o menor) que la zona que se va a ver en nuestra página. En caso de no definir `viewBox` (no es obligatorio) nuestras dimensiones de referencia serán las indicadas por los parámetros `width` y `height`.

En nuestro ejemplo como se puede apreciar hemos definido un área de trabajo que va desde -10, -10 a 310,310, en total tendremos una zona de trabajo de 320 px por 320 px mientras que solo se visualizará en nuestra página 300 px por 300 px. A continuación dejamos un enlace donde podemos ver de forma más detallada diferentes [ejemplos del uso del atributo viewBox](#).

[Código ejemplo](#) (zip - 1,11 KB)

Para saber más

En el siguiente enlace se completa este apartado con código y ejemplo sobre cómo dibujar las figuras básicas en SVG .

[Figuras básicas con SVG](#) .

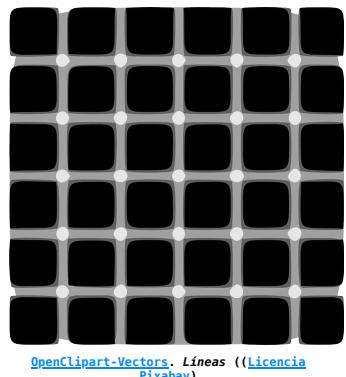
4.3.- El elemento path.

Una ruta o path es un trazado de una o varias líneas que se dibujan una tras otra, tal y como si dibujásemos con un lápiz. El elemento path, es el más complejo de todos los elementos de dibujo, y permite crear dibujos arbitrarios utilizando un conjunto de comandos especializados. En la siguiente tabla se muestra una lista de estos.

Para definir una ruta en SVG se utiliza la etiqueta path y el atributo d:

```
<path d=".comandos de la ruta .." />
```

Los comandos que ponemos como valor del atributo d consisten en una serie de letras, que indican el tipo de trazado y a continuación una serie de números que indican la posición y características del trazado. Dichos valores pueden ser:



Comando	Descripción
M	Moverse a la posición indicada.
L	Traza una línea recta desde la posición actual a la indicada.
H	Traza una línea recta horizontal desde la posición actual a la indicada.
V	Traza una línea recta vertical desde la posición actual a la indicada.
C	Traza una línea curva de Bézier desde la posición actual a la indicada.
S	Traza una línea curva leve desde la posición actual a la indicada.
Q	Traza una línea curva cuadrática de Bézier desde la posición actual a la indicada.
T	Traza una línea curva cuadrática de Bézier leve desde la posición actual a la indicada.
A	Traza un arco elíptico desde la posición actual a la indicada.
Z	Cierra la ruta actual.

Estos comandos pueden escribirse en mayúsculas o en minúsculas atendiendo a las diferencias. Ya que al escribir la letra en mayúsculas, se toma una posición absoluta y las coordenadas se miden desde su origen. Y, si se escribe la letra en minúsculas se toma una posición relativa y las coordenadas se miden desde el último punto que se ha puesto en la ruta.

Dibujando un triángulo con path.

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <path d="M10 0 L75 100 L225 100 Z" fill="OrangeRed" />
</svg>
```

El resultado visual sería el siguiente:



Elaboración propia. Triángulo con SVG ([CC BY-SA](#))

Dibujando figuras con path.

En el ejemplo anterior hemos definido un figura que hemos cerrado, pero podemos definir un trazado y el estilo de éste, para ello tenemos atributos, como son stroke, stroke-width, stroke-linecap, stroke-dasharray y stroke-dashoffset, los cuales pasamos a explicar. También se pueden utilizar atributos ya conocidos y vistos en CSS como son opacity. Estos atributos se aplican con el atributo style:

- ◆ stroke: permite definir el color de nuestro trazado.
- ◆ stroke-width: define el grosor del trazado.
- ◆ stroke-linecap: indica la forma en que tendrá los extremos de un segmento. Sus posibles valores son: butt (recto), round(redondeado) y square (redondeado con la línea prolongada).
- ◆ stroke-dasharray: esta propiedad permitirá definir valores discontinuos, así se dibujará parte de linea, posteriormente se produce una discontinuidad en el trazado y se volverá a dibujar la línea continua.
 - ◆ stroke-dasharray="20 5". Este atributo con este valor dibujará una línea de 20 píxeles, después se dejarán 5 píxeles en blanco y se volverá a repetir el patrón. Podemos poner tantos valores como queramos
 - ◆ stroke-dasharray="20 10 10 5". Dibujaremos 20 píxeles dejaremos un espacio de 10 píxeles se dibujará 10 píxeles, se dejará un espacio de 5 y volverá a repetirse el patrón.
- ◆ stroke-dashoffset: especifica un desplazamiento del extremo desde el que empezará a dibujarse nuestro trazado.

Ejemplo:

```
style="stroke:#CC3366; stroke-width:3; fill:none;" stroke-dasharray="6,3,3,3"
```

Estas propiedades, son propiedades CSS que pueden utilizarse con @keyframe para realizar también animaciones.

A continuación mostramos unos ejemplos de trazados diferentes, intenta averiguar cual será la figura que se forma.

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SVG etiqueta path</title>
</head>
<body>
    <svg width="600" height="400">
        <path d="M25 25 v225 " style="stroke:#CC3366; stroke-width:3; fill:none;" stroke-dasharray="6,3,3,3"></path>
        <path d="M25 250 h100 v50 l100 100 L50 50 h200 C100 100 150 200 300" style="stroke:blue; stroke-width:10; fill:none;" stroke-dasharray="10,2"></path>
        <path d="M25 250 h100 v50 l100 100 L50 50 h200 " style="stroke:blue; stroke-width:10; fill:none;" stroke-dasharray="10,2"></path>
        <path d="M225 250 C350 100 350 150 500 300 " style="stroke:red; stroke-width:10; fill:none;" stroke-dasharray="10,2"></path>
        <path d="M225 250 C350 400 350 350 500 300 " style="stroke:red; stroke-width:10; fill:none;" stroke-dasharray="10,2"></path>
    </svg>

    <svg viewBox="0 0 600 400" width="600" height="400">
        <path d="M25 25 v50 h50 v50 h50 v50 h50 v-50 h50 v-50 h50 v100 h60" style="stroke:green; stroke-width:3; fill:none;" x0></path>
    </svg>

    <svg viewBox="0 0 600 400" width="600" height="400">
        <!-- Vamos a dibujar una curva, nos posicionamos con M, definimos el primer punto de la curva con la letra C los dos primeros números el primer número 100 100 el segundo punto representa el segundo par 200 0 y el tercer punto viene representado por el tercer par 300 300 -->
        <path fill="none" stroke="blue" d="M100 200 C100 200 200 0 300 200" style="stroke-width:3;"/>
    </svg>
</body>
</html>

```

[Código ejemplo](#) (zip - 703 B)

Autoevaluación

Indica V (Verdadero) o F (falso), según corresponda en la siguiente afirmación.

El elemento path es usado para definir caminos en SVG .

Verdadero Falso

Verdadero -----

Recomendación

Como recursos complementarios a este apartado se recomiendan los siguientes artículos que complementan el apartado con algunos ejemplos:

[SVG_path](#)

4.4.- Texto.

Además de las formas básicas, también se puede utilizar SVG para generar texto, mediante la etiqueta `text`. Esta etiqueta tiene su correspondiente etiqueta de cierre, y entre la etiqueta de apertura y la de cierre se debe incluir el texto a dibujar. La sintaxis básica sería:

<text> I love SVG </text>

Además, será necesario incluir algunos atributos que indiquen en qué lugar poner el texto. Estos son los atributos `x` e `y` que indican las coordenadas respectivamente del lugar en donde poner el texto.

```
<text x=10 y=50 >I love SVG</text>
```

Dentro de esta etiqueta podemos definir también propiedades como fuente, tamaño y alineado:

- font-size: definimos el tamaño de la fuente.
 - font-family: podemos importar fuentes con la regla @font-face
 - text-anchor: permite alinear el texto; podemos aplicar los valores: start, middle, end

→ **text-anchor:** permite alinear el texto, podemos aplicar los valores: start, middle, end. Estos atributos y otros ya conocidos como fill, stroke, stroke-width, etcétera, pueden utilizarse, para definir el estilo de nuestro texto. Así pueden utilizarse como muestra en el siguiente ejemplo como parte de la etiqueta text o bien dentro del atributo style. También podemos utilizar otras propiedades vistas en la unidad de CSS como son letter-spacing, word-spacing, etcétera.



Gerd Altman. Texto ([Licencia pixabay](#))

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SVG etiqueta text</title>
</head>
<body>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="600" height="400">
        <text x=10 y=10 >Hola mundo</text>
        <text x=10 y=50 font-size="20">Hola mundo</text>
        <text x=10 y=100 font-size="20" font-family="Monospace">Hola mundo</text>
        <text x=10 y=150 font-size="20" font-family="Monospace" fill="blue" style="letter-spacing:4; word-spacing:20;" >Hola mundo</text>
        <text x=10 y=200 font-size="50" font-family="Verdana" fill="black" stroke="red" stroke-width=2 style="letter-spacing:4; word-spacing:10;" >Hola mundo</text>
        <text x=10 y=250 font-size="50" style="font-family:Times New Roman; fill:black; stroke:red; stroke-width:2; letter-spacing:10;" >Hola mundo</text>
    </svg>
</body>
</html>
```

Hola mundo

Hola mundo

Hola mundo

Hola mundo

Hola mundo
Hola mundo

Elaboración propia. Texto ([CC BY-SA](#))

Recomendación

En el siguiente artículo se proporcionan más ejemplos para trabajar con la etiqueta <text>.

SVG

Dentro del elemento text, si quisieramos seleccionar un conjunto de caracteres podemos utilizar el elemento tspan, es algo similar a la etiqueta span de HTML.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SVC significa todo!</title>
```

```
</head>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="600" height="400">
  <text x=10 y=10 >Hola mundo</text>
  <text x=10 y=50 font-size="20">Hola <tspan font-size="40">mundo</tspan></text>
  <text x=10 y=100 font-size="20" font-family="Monospace">Hola <tspan font-size="40" font-family="Verdana">mundo</tspan></text>
  <text x=10 y=150 font-size="20" font-family="Monospace" fill="blue" style="letter-spacing:4; word-spacing:20;" >Hola <tspan for
</svg>
</body>
</html>
```

[Código ejemplo](#) (zip - 0,97 KB)

4.5.- Gradientes.

Como ya sabemos, cuando hablamos de gradientes o degradados tenemos que especificar entre: degradados lineales y degradados radiales.

Los degradados pueden aplicarse a cualquier elemento al que se le haya aplicado color, tanto de relleno (fill) como en el trazado (stroke).

- Degradado Lineal. La etiqueta que se utiliza para crear un degradado lineal es: `linearGradient`.

Esta etiqueta necesita estar dentro de la etiqueta `defs`:

```
<defs>
  <linearGradient ..... >
  .....
</linearGradient>
</defs>
```



[donations welcome. Degradado.](#) ([Licencia pixabay](#))

Los gradientes lineales se pueden definir como horizontales, verticales o angulares:

- Los gradientes horizontales se crean cuando Y1 e Y2 son iguales y x1 y x2 diferentes.
 - Los gradientes verticales se crean cuando X1 y X2 son iguales y y1 e y2 diferentes.
 - Los gradientes angulares se crean cuando x1 y x2 son diferentes y Y1 e Y2 diferentes.
- Lo siguiente es definir los colores del degradado. Para ello, dentro de la etiqueta `linearGradient` se especifican los colores mediante la etiqueta `stop`. Esta etiqueta se repite para cada cambio de color en el degradado y tiene la siguiente sintaxis:

```
<stop offset=<num> stop-color=<color>"/>
```

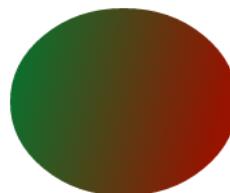
Donde:

- El atributo `offset` tiene como valor un número decimal del 0 al 1. Coloca un marcador en la línea que va desde el punto de inicio (en el 0) al punto final (en el 1) del degradado.
 - El atributo `color-stop` indica el color que habrá en ese punto. Entre un punto y otro se produce el cambio progresivo de color.
- Lo último que queda es incluir el degradado en un elemento. Para ello se utiliza el atributo `fill`, con el que haremos referencia al degradado.

Veamos un ejemplo de una elipse con degradado lineal:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
    <linearGradient id="grad" x1="0%" y1="0%" x2="100%" y2="20%">
      <stop offset="0%" style="stop-color:rgb(0,120,50);stop-opacity:1"/>
      <stop offset="100%" style="stop-color:rgb(150,20,0);stop-opacity:1"/>
    </linearGradient>
  </defs>
  <ellipse cx="200" cy="90" rx="85" ry="70" fill="#grad"/>
</svg>
```

El resultado visual sería el siguiente:



Elaboración propia. Elipse rellena con gradiente. ([CC BY-SA](#))

- Degradado Radial. La etiqueta que se utiliza para crear un degradado lineal es: `radialGradient` y su utilización es similar a la de `linearGradient`. En este degradado el cambio de color va desde un punto central hacia el exterior formando círculos. Los atributos que acompañan a esta etiqueta son:

- `cx="valor" cy="valor"`: coordenadas "x" e "y" del centro del círculo de degradado. En estos atributos y los siguientes podemos indicar una medida, un número (pixeles) o un porcentaje.
- `r="valor"`: indica el radio dentro del cual tendrá alcance el degradado.
- `fx="valor" fy="valor"`: indica el foco del degradado. El foco es el punto desde el cual se inicia el degradado. Éste se extiende desde el foco hacia el resto del círculo marcado. El foco no tiene porqué ser el mismo punto que el centro del círculo. En caso de serlo podemos omitir estos atributos.

[Código ejemplo](#) (zip - 0,47 KB)

Recomendación

En los siguientes enlaces se proporcionan, como recurso complementario a este apartado, códigos de ejemplos de gradientes radiales y lineales sobre diferentes dibujos.

[Gradientes radiales svg.](#)

[Gradientes de color canvas.](#)

4.6.- Transformaciones.

Ya vimos en canvas que una transformación se puede definir como un efecto que permite a un elemento o conjunto de elementos cambiar de tamaño, forma o posición.

En SVG los métodos utilizados para las transformaciones son los mismos que en CSS3. Para ello se utiliza el atributo **transform**, al cual se la aplican como valor diferentes métodos.

- ◆ **translate**: permite realizar un desplazamiento del elemento respecto a su posición original. La sintaxis sería la siguiente:

```
transform="translate(x,y)"
```

donde "x" e "y" indicará el desplazamiento horizontal y vertical respectivamente que tendrá el elemento.

En el siguiente código mostramos un círculo blanco y como el mismo círculo (con fondo de color verde) se desplaza 300 píxeles en el eje x y 50 en el eje y.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SVG transformaciones</title>
</head>
<body>

<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="500" height="400">

    <circle cx=90 cy=90 r=50 stroke="white" stroke-width=2 fill="green" />
    <circle cx=90 cy=90 r=50 stroke="OrangeRed" stroke-width=2 fill="green" transform="translate(300,50)" />

</svg>

</body>
</html>
```

El resultado del código se puede apreciar en la siguiente imagen.



Elaboración propia. Translación ([CC BY-SA](#))

Si además, lo que se necesita trasladar es un conjunto de elementos, habrá que englobarlos dentro de una etiqueta **<g>** ... **</g>** a la cual se le aplicará la transformación.

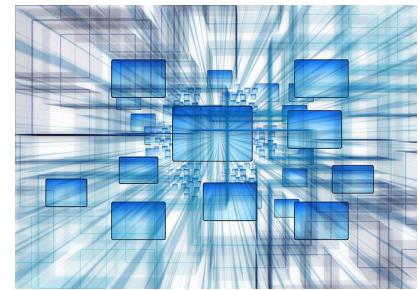
```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="400" height="400">
    <circle cx=90 cy=90 r=50 stroke="OrangeRed" stroke-width=2 fill="Coral"/>

    <circle cx=90 cy=90 r=50 stroke="black" stroke-width=2 fill="gold"/>
    tr
        <g transform="translate(200,100)">
            <circle cx=90 cy=90 r=50 stroke="red" stroke-width=2 fill="blue">
        </g>
    </tr>
</svg>
```

El resultado visual sería el siguiente:



Elaboración propia.. Translate SVG ([CC BY-SA](#))



Gerd Altmann. Transformación ([Licencia Pixabay](#))

- ◆ **scale:** permite reducir o ampliar el elemento al que se le aplica. Su sintaxis es la siguiente:

transform="scale(sx,xy)"

donde sx y sy son valores que indican las veces que se amplia o reduce el elemento en la coordenada "x" e "y" respectivamente. En relación al tamaño, el valor 1 indica el tamaño real del elemento, por debajo de 1 se reduce, y por encima se agranda.

- ◆ **rotate:** permite rotar un elemento respecto a su posición original, indicando el número de grados que girará el elemento. Su sintaxis es la siguiente:

transform="rotate(num)"

Así por ejemplo en el siguiente código se realiza la rotación del texto dibujado:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="400" height="400">

    <text x="100" y="90" style="font: italic bold 1em tahoma; fill: darkblue;"> ROTAR </text>

    <g transform="rotate(40)">
        <text x="120" y="90" style="font: italic bold 1em tahoma; fill: darkblue;"> ROTAR </text>
    </g>

</svg>
```

- ◆ **skew:** permite deformar el elemento variando el ángulo de sus coordenadas. Su sintaxis es la siguiente:

transform=="skewX(N),skewY(N)"

donde N corresponde a la inclinación de la figura respecto del eje "x" e "y" respectivamente.

- ◆ **matrix:** este método es una combinación de los métodos de traslación, escalado y sesgado. Su sintaxis es la siguiente:

transform="matrix(a,b,c,d,e,f)"

donde:

- ◆ los dos últimos parámetros actúan igual que el método translate, tal que indican el desplazamiento en los ejes "x" e "y" respectivamente.
- ◆ los parámetros primero y cuarto actúan igual que el método scale, tal que el parámetro "a" hace un escalado en el eje "x", y el parámetro "d" hace un escalado en el eje "y".
- ◆ los parámetros segundo y tercero actúan como los métodos skew. Aquí el número que debemos indicar no es el número de grados del ángulo sino su tangente. El parámetro "b" inclina el eje "x" un ángulo cuya tangente es la indicada. El parámetro "c" hace lo mismo con el eje "y".

[Código ejemplo](#) (zip - 2,95 KB)

Autoevaluación

El método que utiliza el atributo transform para cambiar el tamaño de una figura dibujada es:

matrix

scale()

rotate()

translate()

[Mostrar retroalimentación](#)

Solución

- Incorrecto
- Correcto
- Incorrecto
- Incorrecto

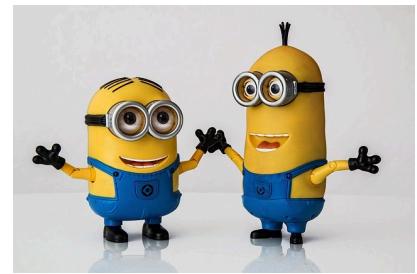
4.7.- Animaciones.

Para crear animaciones se puede hacer de varias formas: se podría aplicar código JavaScript sobre un elemento SVG, también se puede usar el método de animación de CSS3. Sin embargo SVG tiene cuatro etiquetas de animación:

- ◆ **animate**

mediante esta etiqueta se pueden hacer animaciones sobre atributos CSS de los elementos. Así por ejemplo, si se tiene un rectángulo de color azul, se puede animar para que cambie a color rojo mediante el siguiente código:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width=300 height=300>
  <rect width="200" height="100" style="fill:rgb(0,0,255);">
    <animate attributeType="CSS" attributeName="fill" to="rgb(255,0,0)" dur="3s" fill="freeze" />
  </rect>
</svg>
```



[Steve Buissinne](#). Animación ([Licencia Pixabay](#))

donde tenemos algunos de los siguientes atributos:

- ◆ **attributeType**: indica el tipo de atributo al que se le aplicará la animación, como valores puedes poner XML (valor por defecto) o CSS.
- ◆ **attributeName**: tendrá como valor el nombre del atributo al que se aplica la animación.
- ◆ **from**: indica el valor de la propiedad a cambiar al principio de la animación.
- ◆ **to**: indica el valor de la propiedad a cambiar al final de la animación.
- ◆ **dur**: indica el tiempo de duración de la animación.
- ◆ **begin**: se indica aquí el tiempo que transcurre desde que se da la orden de ejecutar la animación, hasta que la animación se pone en marcha.
- ◆ **end**: indicamos cuando queremos que finalice la animación.
- ◆ **repeatCount**: indicamos el número de repeticiones, podemos asignar el valor infinite.
- ◆ **repeatDur**: indicamos cuánto tiempo hay que repetir la animación se expresa en tiempo (s, m, h).

Si quisieramos que además de cambiar de color cambiar el ancho del rectángulo lo haríamos con el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SVG animaciones</title>
</head>
<body>

  <svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="400" height="400">
    <rect width="200" height="100" style="fill:rgb(0,0,255);">
      <animate attributeType="CSS" attributeName="fill" to="rgb(255,0,0)" dur="3s" fill="freeze" />
      <animate attributeType="CSS" attributeName="width" to="300" dur="5s" fill="freeze" />
    </rect>
  </svg>
</body>
</html>
```

- ◆ **set**

Esta etiqueta no produce una animación sino un cambio de estado instantáneo de un elemento. Utiliza los mismos atributos que **animate**, sin embargo el cambio no es gradual.

A continuación mostramos una animación con esta etiqueta que comentamos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SVG animaciones</title>
</head>
<body>

  <svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="400" height="400">
    <rect width="200" height="100" style="fill:rgb(0,0,255);">
      <set attributeType="XML" attributeName="fill" begin="3s" dur="6s" to="black" fill="remove" />
    </rect>
  </svg>
</body>
```

```
</body>  
</html>
```

Dibujamos un rectángulo que aparece con relleno azul, pasados 3 segundos se cambia a color negro, con este color estará 6 segundos y pasado esto volverá al valor inicial. Si quisieramos que se quedara con el último estado, la propiedad fill, le daríamos el valor freeze, como en el ejemplo anterior.

→ **animateMotion**

esta etiqueta permite hacer que un elemento se mueva siguiendo una ruta determinada. Con esta etiqueta hay varias opciones para animar el elemento, se pueden pasar los valores a través de from to, values y también por path y mpath.

- from to: es necesario pasar dos valores (la coordenada x, la coordenada y).
 - values: también es necesario pasarle dos valores (coordenada x, coordenada y).
 - path: se puede animar un elemento haciendo que siga el trazado marcado por el atributo path. Es conveniente recordar el apartado 2.3. donde se vio este elemento.
 - mpath: este método nos permite aprovechar un trazado ya definido en el documento. Para utilizarlo será necesario referenciarlo a través del atributo xlink:href.
 - repeatCount: nos permitirá indicar cuantas veces queramos que se repita una animación (iteraciones).
 - rotate: permite indicar si el objeto tendrá que girarse para seguir la trayectoria definida. Los valores que puede tomar son auto, auto-reverse y un número que indicará los grados que tiene que girar el objeto.
 - fill="freeze": la animación se ejecuta una sola vez. No hay que confundir este atributo con el atributo de relleno.

Resaltar que en SVG podemos definir identificadores para posteriormente utilizarlos, así podemos definir un id para la etiqueta `animateMotion` o para una figura definida con `circle` o `path`. Como posteriormente veremos en un ejemplo, estos identificadores nos permitirán definir el inicio y el fin de una animación.

Vistas las cuatro etiquetas que podemos utilizar para realizar animaciones vamos a analizar un par de ejemplos con `animateMotion`, así definiremos dos animaciones, una con `path` y otra con `mpath` también veremos como podemos configurar nuestra animación definiendo determinadas propiedades como: cuándo empezar, terminar o cómo debe comportarse el objeto animado. Analiza los comentarios que tiene el código e intenta simular la animación.

Código documentado de dos animaciones.

→ `animateTransform`

esta etiqueta permite hacer animaciones en las cuales el cambio que se produce en el elemento es debido a una transformación de las ya mencionadas con el atributo transform.

A continuación dejamos un ejemplo comentado.

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Animación con SVG</title>
</head>
<body>

<!-- Tenemos la animación de un rectángulo que empezará a los dos segundos o cuando hagamos click sobre él-->
<!-- finalizará cuando pasamos el ratón sobre la figura.-->
<!-- la animación dura tres segundos y se repite de forma indefinida.-->
<svg viewBox="-10 -10 610 510" width="600" height="500">

    <rect x="100" y="100" width="100" height="80" fill="blue">
        <animateTransform
            attributeType="XML"
            attributeName="transform"
            type="translate"
            values="200,100; 0,0; 0,200; 200,200"
            begin="2s;click" end="mouseover"
            dur="3s"
            repeatCount="indefinite" />
    </rect>
</svg>

<!-- Dibujamos una figura geométrica que gira sobre si misma durante 3 segundos de forma indefinida y finaliza cuando hacemos click sobre ella-->
<svg viewBox="-200 -200 600 600" width="400" height="400"
    xmlns="http://www.w3.org/2000/svg">

    <polygon points="60,130 80,130 100,180 80,230 60,230 40,180 60,130" fill="green">
        <animateTransform attributeName="transform"
            attributeType="XML"
            type="rotate"
            from="0 60 90"
            to="360 60 90"
            dur="3s"
            end="click"
            repeatCount="indefinite"/>
    </polygon>
</svg>
</body>
</html>

```

[Códigos de ejemplo \(zip - 3,89 KB\)](#)

Para saber más

Como material complementario y de refuerzo para trabajar estos conceptos, se proporcionan estos enlaces donde se muestran una serie de ejemplos sobre cómo animar elementos mediante SVG.

[Animación en SVG.](#)

[Ejemplos de animaciones con animateMotion.](#)

[Animaciones SVG](#)

Sobre las animaciones SVG <https://cosasdigitales.com/tutoriales-practicos-diseno-web/introduccion-animacion-vectorial-web-con-svg/>

Para finalizar dejamos un par de enlaces donde se muestra como realizar animaciones con SVG y CSS3:

[Animación de un logo.](#)

[Animación de una imagen.](#)

Anexo: Ejemplos.

En este anexo se muestran algunas ejercicios realizados por alumnos y alumnas del módulo.

En este código se crea una Game Boy con CSS, canvas y jQuery para dar una pequeña animación. La pantalla se enciende al hacer click sobre el botón start. Encendida la pantalla se ejecuta una animación al darle al botón arriba.



Noemí Guerrero Pintado (Curso 21/22). Game Boy (Dominio público)

[Código de Game Boy](#)

En el siguiente ejemplo se muestra una animación donde se simula un punto del famoso juego de ping-pong, se utiliza animaciones en CSS, dibujos con canvas y se asignan las animaciones con jQuery, al hacer click en el botón "Ver punto de partido".



José Migueles Sánchez (Curso 21/22). Ping-pong (Dominio público)

[Código de Ping Pong](#)

Condiciones y términos de uso de los materiales

Materiales desarrollados inicialmente por el Ministerio de Educación, Cultura y Deporte y actualizados por el profesorado de la Junta de Andalucía bajo licencia Creative Commons BY-NC-SA.



Antes de cualquier uso leer detenidamente el siguiente [Aviso legal](#)

Historial de actualizaciones

Versión: 03.00.02	Fecha de actualización: 28/02/24
-------------------	----------------------------------

Actualización de materiales y correcciones menores.

Versión: 03.00.00	Fecha de actualización: 28/06/22	Autoría: Jesús Moreno Ortiz
-------------------	----------------------------------	-----------------------------

Ubicación: Toda la unidad

Mejora (tipo 3): Actualmente en la unidad no hay nada relacionado con las animaciones en CSS, así habría que incluir un punto con al menos 3 subapartados donde se explique esta tecnología. Habría que incluir transformaciones, transiciones y animaciones preferentemente en 2D y poner una introducción a las animaciones en 3D. En cuanto a los contenidos, habría que actualizar sobre todo los puntos de introducción y ampliar un poco lo relacionado con SVG. Lo relacionado con CANVAS se dejaría casi igual, ya que esta tecnología quizás tenga menos relevancia hoy en día.

En cuanto a los cambios dentro de la unidad los podemos agrupar en tres tipos:

- Actualizar los contenidos de introducción, eliminando todo lo relacionado con tecnologías ya desfasadas como flash.
- Habría que actualizar la autoría de imágenes de toda la unidad e incluir nuevas imágenes, así hay páginas que no tienen ninguna.
- Relacionado con los ejemplos de la unidad, habría que añadir más. Además, muchos de ellos son poco ilustrativos, habría que poner ejemplos completos. Ahora mismo muchos de ellos son extractos de código. También habría que incluir imágenes con el resultado de dichos ejemplos. Habría que añadir también ficheros con el código ejemplo, sobre todo en aquellos ejemplos un poco más complejos.

Habría que reestructurar el índice de la unidad.

Habría que actualizar orientaciones del alumnado, mapa conceptual y banco de preguntas.

A continuación se explican de forma breve los cambios en los apartados actuales

Punto Título Explicación del cambio

0. Contenidos Multimedia Autoría de imágenes

1. Las animaciones en la web Autoría de imágenes y contenidos

1.1. Formatos de animaciones. Autoría y actualizar contenidos. Eliminar tecnologías que no se usan hoy en día.

1.2. Herramientas de programación. Hay que actualizar contenidos y autorías. Ampliar un poco más sobre las herramientas que permiten realizar animaciones.

1.3. API Canvas. Autoría y actualizar contenidos. contenidos.

1.3.1. Primeros pasos con Canvas. Autoría

Reestructurar apartado

1.3.2. Dibujar figuras. Autoría y rehacer ejemplos

1.3.3. Dibujar trazados. Autoría y rehacer ejemplos

1.3.4. Colores y sombras. Autoría y rehacer ejemplos

1.3.5. Añadir texto. Autoría y rehacer ejemplos

1.3.6. Aplicar un degradado. Autoría y rehacer ejemplos

1.3.7. Las transformaciones y manipulaciones de objetos. Autoría y ejemplos

1.3.8. Procesando imágenes. Autoría y ejemplos.

1.3.9. Animaciones en el lienzo. Autoría y ejemplos.

1.3.10. Vídeo en el lienzo. Eliminar y unificar en el punto anterior

2. Gráficos Vectoriales Escalables. Insertar imagen y autoría

2.1. Primeros pasos con SVG. Contenidos y poner un ejemplo para descargar.

2.2. Dibujar figuras básicas. Autoría y ejemplos

2.3. El elemento path.

2.4. Texto. Autoría Ejemplos

2.5. Gradientes. Autoría

2.6. Transformaciones. Autoría, contenidos y Ejemplos

2.7. Animaciones. Ampliar contenidos y poner ejemplos

Ubicación: Unidad 5

Mejora (Examen online): Se incluye examen actualizado en Otros materiales.

Ubicación: Todo

Mejora (Mapa conceptual): Se actualiza y mapa conceptual y preguntas

Ubicación: Todo

Mejora (Orientaciones del alumnado): Se actualiza las orientaciones con respecto a los contenidos actualizados.

Versión: 02.01.00	Fecha de actualización: 22/02/21	Autoría: Jesús Moreno Ortiz
-------------------	----------------------------------	-----------------------------

Ubicación: 2.7.

Mejora (tipo 2): Se explican atributos relacionados con la etiqueta animateMotion y se añade un ejemplo comentado con dos animaciones. Se elimina el iDevice recomendación y se añade un Debes conocer.

Ubicación: 2.3.

Mejora (tipo 2): Se explican atributos relacionados con la etiqueta path y se añade un ejemplo comentado.

Ubicación: 1.3.9.

Mejora (tipo 2): Se añade información sobre las funciones setInterval y clearInterval y se añade un ejemplo comentado sobre animación en canvas.

Ubicación: 2.2.

Mejora (tipo 2): Se añaden ejemplos para dibujar polígonos y se amplían los contenidos.

Ubicación: 1.3.10.

Mejora (tipo 1): Se añade un iDevice Para saber más.

Ubicación: 1.3.8.

Mejora (tipo 1): Se añade un ejemplo comentado.

Ubicación: 1.3.5.

Mejora (tipo 1): Se modifica el juego de caracteres del código ejemplo.

Ubicación: 1.3.2.

Mejora (tipo 1): Se modifica el juego de caracteres del código ejemplo.

Ubicación: 1.3.1.

Mejora (tipo 1): Se modifica el juego de caracteres del código ejemplo.

Ubicación: 1.3.

Mejora (tipo 1): Se modifica la comparativa Canvas Vs Flash

Ubicación: 1.2.

Mejora (tipo 1): Modifico los iDevices Debes Conocer y Para saber más y se elimina el recomendación.

Versión: 02.00.00	Fecha de actualización: 19/05/14	Autoría: Eliana Yemina Manzano Fernández
-------------------	----------------------------------	--

Actualización de contenidos de animaciones a HTML5.

Versión: 01.00.00	Fecha de actualización: 19/05/14
-------------------	----------------------------------

Versión inicial de los materiales.

