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.

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.

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, etc. 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.

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.

Hay distintos formatos de archivo que permiten visualizar animaciones en la Web. Algunos como el GIF y el SVG ya se mencionaron en unidades anteriores, pero hay otros como:

- El SWF que es un formato de archivo de gráficos vectoriales creado por la empresa Macromedia (actualmente Adobe Systems) que suele emplearse en la Web debido a que con un tamaño reducido de archivo permite una gran interactividad de los usuarios con la web incluso a los que tienen un ancho de banda escaso.
- El MNG que es un formato de fichero, libre de derechos, para imágenes animadas y es una extensión del formato de imagen PNG.

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.

Tenemos varias formas de añadir animación a la Web:

- Usando diferentes formatos de archivo que permiten añadir animación, tal y como ya dijimos en el apartado anterior: GIF, MNG, SVG y SWF.
- Utilizando el potencial de las reglas de estilo para simular la animación de los botones, enlaces, elementos de formulario, etcétera.
- Hacer uso de lenguajes de programación dinámicos.

El mercado de la Web es muy amplio y el abanico de software para la realización de animaciones es muy extenso y en constante evolución. La tabla siguiente refleja algunos de los programas empleados por las personas dedicadas al diseño de interfaces cuando quieren incorporar animaciones a su diseño. El uso de uno u otro estará determinado por el tipo de animación a realizar y el presupuesto que tengamos.

GIFs animados

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, nitidez, que permite usar 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. Algunas son las siguientes:

- **Microsoft GIF Animador** 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.

Archivo Flash: swf (.fla)

Durante más de una década, *.swf* ha sido el formato más atractivo de la web (los archivos *.fla* son los archivos editables). Los archivos flash (*.swf*) son creados, inicialmente, con la herramienta Adobe Flash (antes Macromedia Flash), aunque existen otras herramientas con las que también pueden crearse.

La ventaja de *.swf* es que es un formato vectorial e integra tanto imagen, texto, vídeo y audio. Además, incluye un lenguaje de programación *ActionScript* que hace que estas animaciones puedan interactuar, al reconocer, por ejemplo, eventos de ratón.

Actualmente, el uso de este formato en la web está viéndose reducido por dos razones principales:

- La dificultad de que el contenido de los *.swf* puedan ser indexados en su totalidad por los motores de búsqueda.
- Desde el auge de iPad el uso de .swf en páginas web se ha reducido considerablemente. El motivo, es que hasta la fecha, Apple se niega 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 disponibles 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.

Shockwave

Es un formato muy popular para presentar contenidos animados más complejos que Flash (juegos, entre otros). Realmente, ambos fueron creados por la misma compañía (Macromedia). Shockwave permite crear aplicaciones más complejas que con Flash. 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.

1.2.- Herramientas de programación.

Además de los formatos vistos en la sección anterior, existen muchas otras formas de incluir animaciones en un sitio web usando código de programación. En esta sección se mostrarán dos de las más utilizadas actualmente, anque existen muchas más, dependiendo del lenguaje que se quiera utilizar para ello.

HTML y JavaScript

HTML Dinámico (JavaScript y otros): una primera alternativa para crear animaciones que pueda interactuar con los usuarios en una página web 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.

Un caso concreto de este tipo de solución es usar HTML con jQery.jQuery es una biblioteca de JavaScript, 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. En la siguiente unidad "Contenidos web interactivos" se detalla jQuery.

En Internet existen una gran cantidad de tutoriales y código gratuito para hacer animaciones con esta tecnología. Además, la siguiente unidad se centra en ella para implementar la interacción en la web. Un ejemplo de animaciones con jQuery es el siguiente: donde se simula el comportamiento de los elementos de la pantalla de un dispositivo móvil, al desbloquearse la

Este código ha sido desarrollado por la alumna Florina Roxana Marinca de Diseño de Interfaces Web en la modalidad presencial.

Ejemplo 1.- SmartPhone. (rar - 233,49 KB)

Recomendación

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 este framework, en este caso empezamos por la propia.

jquery.com

HTML5 y CSS3

Gracias a CSS3 se puede realizar transiciones, es decir pasar de un estado inicial a un estado final, por ejemplo con el cambio del tamaño de un elemento o del color. En el siguiente enlace se pueden apreciar algunas de las propiedades a las cuales se les puede aplicar una transición. La transición no solo se puede aplicar a una propiedad, sino a varias al mismo tiempo, pudiendo definir parámetros como el tiempo de la misma o la velociad. A continuación se deja un enlace con ejemplos de transiciones.

Una vez que sabemos como realizar una transición, otra de las posibilidad que nos proporciona CSS3 es la de realizar animaciones, para ello tendremos que utilizar la etiqueta @keyframes. Con esta etiqueta, podremos definir una serie de fotogramas clave, donde indicaremos el estado que tendrá el objeto o elemento que deseamos animar. Posteriormente indicaremos como queremos que se realiza dicha animacón y a quien se le aplicara. En el siguiente enlace se puede ver de forma más detalla cómo realizar una animación con @keyframes.

HTML5, CSS3 y JavaScript

Como ya se ha comentado al hablar de archivos .swf, Flash ha sido durante más de una década el elemento estrella de la web. Sin embargo, desde la aparición de los dispositivos móviles (concretamente iPhone e iPad) y su rápida extensión, Flash ha perdido fuelle. Esto ha coincidido con la aparición e implementación por parte de los navegadores de HTML5. Este lenguaje, combinado con CSS3 y JavaScript permite crear animaciones tan potentes como Flash. Además, HTML5 tiene como ventaja respecto a Flash que es soportado por la mayoría de los navegadores actuales de forma nativa y que la web tiende en un futuro a implementarlo completamente en todos.

El uso de HTML5, CSS3 y JavaScript tiene una estructura muy similar a la vista en el ejemplo anterior. Cualquier desarrollador con conocimientos de estos lenguajes puede escribir código para animar sus creaciones. Además, si se apoya en la multitud de páginas web disponibles en Internet con trucos para hacer animaciones o librerías de alto nivel como (jQuery), el resultado es aún más rápido.

Sin embargo, las nuevas tendencias para crear animaciones con estos tres elementos no van por el camino de que el desarrollador escriba todo el código. Lo que muchas aplicaciones proponen actualmente, y seguramente sea una pauta para el futuro son dos alternativas:

- **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 (por ejemplo, .swf) 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:** En el año 2013 Google sacó la versión Beta de su nueva herramienta. Esta permite la creación de contenido animado en **HTML5,** mejorando las posibilidades de crear publicidad sin el uso excesivo de Flash.
  - **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.

Todas estas herramientas, marcan la tendencia de lo que será el desarrollo de animaciones en un futuro. Sin embargo, aunque puedan parecer una panacea, todavía quedan muchas cosas que el diseñador tiene que tener en cuenta a la hora de usarlas, debido al estado en el que se encuentra HTML5 en los navegadores.

Debes conocer

A continuación se deja un enlace donde se pueden aprender más sobre las transiciones y las animaciones con CSS.

Transiciones y animaciones con 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 softwarese puede encontrarse en el siguiente enlace:

Google Web Designer.

Por otro lado, para generar gráficos vectoriales, podemos trabajar con "Inkscape" (como vimos en la unidad de trabajo cuatro). Toda la información que se proporciona de forma oficial sobre este software se puede encontrar en el siguiente enlace:

Inkscape

1.3.- API Canvas.

Previamente a crear animaciones en HTML5 es necesario conocer cuáles son las novedades que incluye el lenguaje al respecto.

HTML5 introduce varias APIs (interfaces de programación de aplicaciones) para proveer acceso a poderosas librerías incluidas en los navegadores. Y, en el caso de generar e imprimir gráficos en pantalla, crear animaciones o manipular imágenes y videos la API con la que se trabaja es Canvas.

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 , cuando hablamos de Canvas nos referimos a toda la API que incluye un conjunto de funciones para dibujar líneas, rectángulos, círculos, etc. así podemos rotar, escalar, transformar, etc.

El elemento canvas permite 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 ahora estaban reservadas a los desarrolladores en Flash, con la ventaja que para usar canvas no será necesario ningún plugin en el navegador, lo que mejorará la disponibilidad de esta nueva aplicación. Por ello, la utilización del elemento **<canvas>** está dirigida, mayormente, a los programadores que usan JavaScript. En efecto, todo lo que se puede crear con este elemento se hará con JavaScript. Sin embargo, este módulo está orientado al diseño web, y no a la programación en JavaScript, así que solamente se verán los elementos básicos.

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.

Recomendación

A estas alturas, sobra decir que harán falta algunos conocimientos básicos de Javascript para poder trabajar con

Librosweb.es

w3scholls.com

1.3.1.- Primeros pasos con Canvas.

1. Definir el elemento canvas.

Para comenzar, dentro del cuerpo de la página se tiene que incluir la etiqueta <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"><br />Su navegador no soporta el elemento canvas.
</canvas>
```

Es un elemento relativamente simple, lo único que hay que especificar al usarlo son los siguientes atributos: width (ancho) y height (alto) declaran el tamaño del lienzo en pixeles. Estos atributos son necesarios 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 atributos, de forma opcional, mediante hojas de estilo para 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:

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

Así se visualizará el elemento canvas por ahora:

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 dibuar formas y trazados en un elemento <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 HTML 5</title>
  <style>
    #ele_canvas{
      border: solid 3px #4682B4;
      background: #FFFFF0;
    }
  </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 carga 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:

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

Recomendación

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

w3schools.com

Iniciación a canvas.

Aprendiendo canvas.

1.3.2.- 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 ya definidas como pueden ser rectángulos y circunferencias.

Dibujando 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 rellenos de color. Y, sus parámetros tienen el mismo objetivos que en la función anterior: posición y dimensiones.
- 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.

Dibujando círculos u arcos:

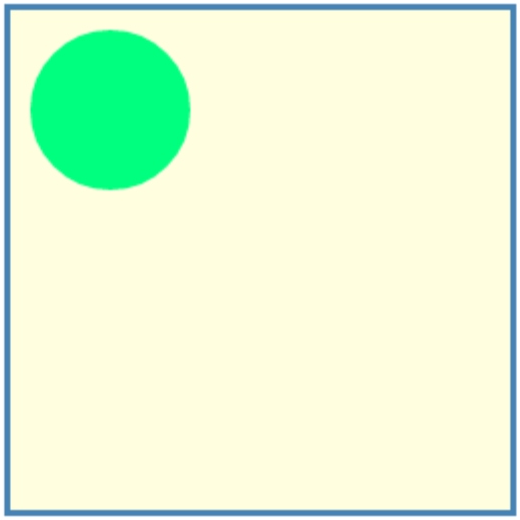
Para dibujar un arco o un círculo en la posición (x,y), se utiliza la función arc(x, y, radio, ángulo inicio, ángulo final, dirección) donde sus parámetros son:

- 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.
- direccción: valor boleano 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 circunferncia sería el resultado obtenido al ejecutar el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Documento HTML 5</title>
  <style>
    #ele_canvas{
      border: solid 3px #4682Bd;
      background: #EFFFF0;
    }
  </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);
      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>
```

La visualización de este código sería la siguiente:



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.

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

1.3.3.- 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:

- 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):

```
//Defina el objeto que referencia al canvas
var elemento=document.getElementById('ele_canvas');
        <br />// Crea el contexto del canvas
dibujo=elemento.getContext('2d');

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

//Defina 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<br />
dibujo.lineTo(90,120);

//Cierra el camino creado anteriormente
dibujo.closePath();

//dibuja el contorno del trazado definido anteriormente
dibujo.stroke();
```

El resultado visual del trazado escrito mediante este código sería el siguiente:

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

```
//creamos el objeto que referencia al canvas
var elemento=document.getElementById('ele_canvas');
 <br />// creamos el contexto
dibujo=elemento.getContext('2d');

// inicializamos el dibujo
dibujo.beginPath();
<br />//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);

// cierra el trazado definido y rellena la figura.
dibujo.fill();
```

El resultado visual sería el siguiente:

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 rectángulos y trayectos II.

Dibujar caminos.

1.3.4.- Colores y sombras.

Para trabajar con color en los elementos canvas existen distintas posibilidades, por ahora se verá cómo modificar el color con el que se rellena o se dibuja trazados:

- Existe un atributo del contexto del canvas que permite cambiar el color de relleno: fillStyle()

```
dibujo.fillStyle="#000099";
```

- El atributo strokeStyle() funciona de la misma forma que fillStyle con la diferencia que permite definir el trazado del dibujo.

```
dibujo.strokeStyle="#000099";
```

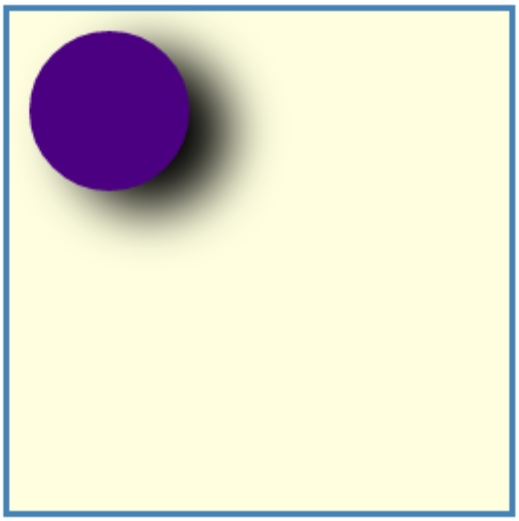
Además de colores, mediante la API Canvas también se pueden generar sombras mediante las siguientes cuatro propiedades:

- **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:

```
var miCanvas=document.getElementById("elg_canvas");
var dibujo=miCanvas.getContext("2d");
dibujo.fillStyle="#4BC082";
dibujo.shadowBlur=40;
dibujo.shadowColor="black";
dibujo.shadowOffsetX=20;
dibujo.shadowOffsetY=10;

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

dibujo.fill();
```

El resultado visual sería:



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

1.3.5.- 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.

Este sería un código de ejemplo para trabajar con texto:

```
<!DOCTYPE HTML>

<html lang="es">

<head>
  <meta charset="UTF-8">
<br /> <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="72px Mincyn";

        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>
```

Y, el resultado visual sería el siguiente:

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.

1.3.6.- Aplicar un degradado.

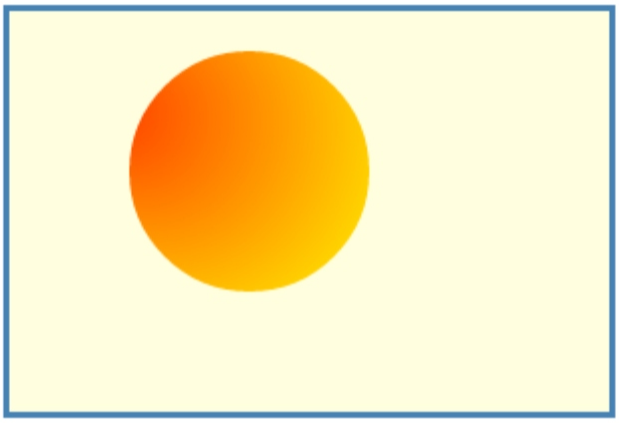
De la misma forma que en CSS3, los gradientes en la API Canvas pueden ser lineales o radiales, y pueden incluir puntos de terminación para combinar colores:

- 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.
- 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.

Así si aplicamos un degradado al círculo anterior mediante el siguiente código:

```
function dibujarCirculo()
{ var /* 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:



1.3.7.- Las transformaciones y manipulaciones de objetos.

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).

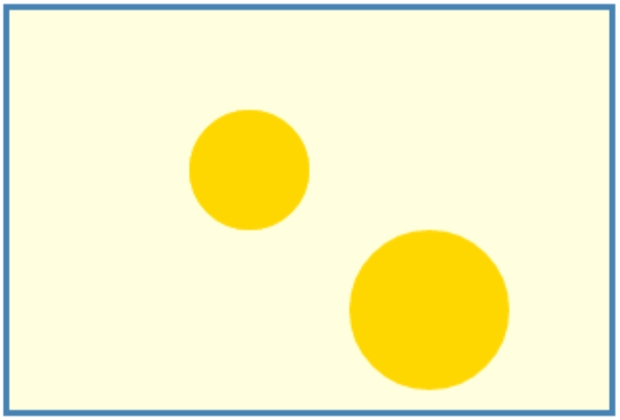
El elemento canvas permite "transformar" los objetos dibujados en el lienzo con los métodos:

- 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 dibujoenmiCanvas=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);
```

Y, el resultado visual sería el siguiente:



- rotate(angulo): Este método de transformación rotará el lienzo alrededor del origen tantos ángulos como sean especificados.
- 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.
- Otros.

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("dig_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

1.3.8.- Procesando imágenes.

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

Para dibujar una imagen en el lienzo se utiliza el método drawImage(), que pertenece al objeto contexto del 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.

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 ponen en funcionamiento el método drawImage():

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="iso-8859-1">
    <title>Documento HTML5</title>
        <style>
            #ele_canvas{
                border: solid 3px #4682B4;
                background: #FFFFE0;
        }
        </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() {
                                    dibujo.drawImage(this, 0, 0);
                        dibujo.drawImage(this, 0, 400,100,100);
                        dibujo.drawImage(this, 400,0,100,100, 350,300,200,200);
                        /* posicionamos la imagen original invertida y desplazada */
                        dibujo.translate(110,500);
                        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>
```

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

1.3.9.- 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 funcion 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 acontinuación 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 ser 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 crear 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 círuclo que cambia de color y tamaño.
- En el segundo canvas, comienza a ejecurarse al cargarse la página, en el se muestra una barra de progreso con un porcentaje.

```
<!DOCTYPE html>

<html lang="es">

<head>
```

```
<meta charset="UTF-8">

<title>Documento HTML5</title>

<style>


    /* Definimos el fondo y el borde de nuestros canvas*/
     #canvas_1,#canvas_3{
          border: solid 3px #4682B4;
          background: #FFFFE0;
      }
    #canvas_2{
          border: solid 3px #4682B4;
          background:red;
      }
</style>



 <script>
      /*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_1");
      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 radio raadio
      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 posción en el eje x del canvas */
        /* y será la posción 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_3");
    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 = "gray";
        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 repeticón 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_1" 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_2" width="10" height="10" onclick="movimiento()">
   Su navegador no soporta el elemento canvas.
</canvas>
<br /><canvas id="canvas_3" width="300" height="300">
   Su navegador no soporta el elemento canvas.
</canvas>

</body>
</html>
```

Debes conocer

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

Animación de un objeto.

A continuación mostramos como se puede realizar la animación de una imagen.

Animación de un a imagen con canvas.

1.3.10.- Videos en el lienzo.

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

Para saber más

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

Videos en Canvas

2.- Gráficos Vectoriales Escalables.

SVG del inglés Scalable Vector Grafics significa Gráficos Vectoriales Escalables es una aplicación para hacer dibujos, banners, gráficos, etc., tanto estáticos como animados, mediante gráficos vectoriales. Es decir, permite hacer las mismas cosas que canvas, aunque de forma diferente. Ya que, con canvas los dibujos se crean mediante píxeles (mapa de bits) y con SVG son creados mediante gráficos vectoriales.

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.
- 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.

Ventajas de SVG.

- Las imágenes SVG se pueden crear y editar con cualquier editor de texto.
- Las imágenes SVG se pueden buscar, indexar, con guión, y se comprimen.
- Las imágenes SVG se pueden imprimir con alta calidad en cualquier resolución.

Compatibilidad con los navegadores.

SVG es compatible con todos los navegadores en sus versiones modernas, si bien en Internet Explorer sólo es compatible con las versiones 9 en adelante. Para versiones anteriores es necesario usar un plugin.

2.1.- Primeros pasos con SVG.

En HTML5, puede incrustar elementos SVG directamente en su página HTML, ya que SVG disponde de una serie de etiquetas y atributos para introducir los elementos:

Mediante las siguientes líneas de código, con la etiqueta

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

</svg>
```

A este elemento es necesario indicarle los atributos width y height (ancho y alto), aunque también sería posible especificarlo mediante CSS. Ya que el código SVG está formado por etiquetas, al igual que HTML, también es posible aplicarle reglas de estilo CSS y JavaScript.

Una forma efectiva y de más posibilidades que tenemos con svg es definir los gráficos o contenido por medio de el 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>
```

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

- Como imagen, referenciando el archivo desde el atributo src:

```
<img src="dibujo.svg" alt="imagen svg" width="200" height="200">
```

- Ccomo iframe, referenciando el archivo mediante el atributo src:

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

- Con la etiqueta embed, referenciando el archivo mediante el atributo src:

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

- Con la etiqueta object, referenciando el archivo mediante el atributo:

```
<object data="dibujo.svg" width="200" height="200"></object>
```

En cualquier caso, es necesario incluir la ruta del archivo, y el width y height de la ventana donde se verá el SVG.

Para saber más

SVG se encuentra en estado de recomendación:

Recomendación del W3C

2.2.- Dibujar figuras básicas.

Para crear distintas formas con los gráficos SVG, existen una serie de etiquetas XML para crearlas la Tabla 1 se enumeran estos elementos:

Tabla 1. 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.

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 medido en píxeles.

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

Así mediante el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="iso-8859-1">
    <title>Documento HTMVL-5</title>

<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>
```

Obtendríamos la siguiente imagen:

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 pixeles 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=400 height=400>
    <rect width="200" height="100" fill="YellowGreen" stroke="Green" stroke-width=2 />
<svg>
```

El resultado visual sería el siguiente:

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>
```

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

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 ver en nuestra página (vería nuestro visitante). En caso de no definir viewBox (no es obligatorio) nuestras dimensiones de referencia serán las indicadas por los parámetros width y heigth. En nuestro ejemplo como se puede apreciar hemos definido un area de trabajo que va desde -10, -10 a 310,310, en total tendremos una zona de trabajo de 320px por 320px mientras que solo se visualizará en nuestra página 300px por 300px. A continuación dejamos un enlace donde podemos ver de forma más detallada diferentes ejemplos del uso del atributo viewBox.

Debes conocer

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

Figuras básicas con SVG.

2.3.- 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 Tabla 2 se hace 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="Orange-Red" />
</svg>
```

El resultado visual sería el siguiene:

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.

- stroke: permite definir el color de nuetro 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 la línea se prolonga) y square (redondeado con la línea prolongada).
- stroke-dasharray: esta propiedad permitira definir valores discontinuos, así se dibujará parte de línea, posteriormente se produce una discontinuidad en e trazado y se volverá a dibujar la línea continua. stroke-dasharray="20 5". Este atributo con este valor dibujará una línea de 20px, después se dejarán 5 pixeles en blanco y se volverá a repetir el patrón, podemos poner tantos valores como queramos stroke-dasharray="20 10 10 5". Dibujaresmo 20px dejaremos un espacio de 10px se dibujará 10px, 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.

Estas propiedades, son propiedades CSS que pueden utilizare con @keyframe para realizar 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>Document</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 100 150 150 200 300" style="stroke:blue; stroke-width:10; fill:none;" stroke-da
  <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 v-50 z" style="stroke:green; stroke-width:3; fill:none;" x°></path>
  </svg>


</body>
</html>
```

Recomendación

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

Rutas en SVG.

SVG path

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

2.4.- Texto.

Además de las formas básicas, usted también 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>
```

Recomendación

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

SVG

2.5.- 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>
```

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 difierentes.
- Los gradientes verticales se crean cuando X1 y X2 son iguales y y1 e y2 difierentes.
- Los gradientes angulares se crean cuando x1 y x2 son diferentes y Y1 e Y2 difierentes.

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:

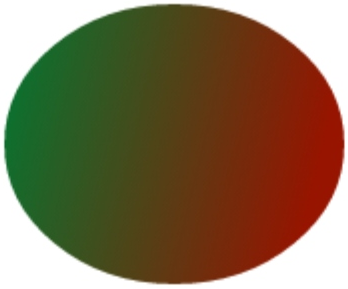
- El atributo offset tiene como valor un número decimal del 0 al 1. Coloca un marcador en la línea que va desde el punto de inicio (en el 0) al punto final (en el 1) del degradado.
- El atributo `color-stop` indica el color que habrá en ese punto. Entre un punto y otro se produce el cambio progresivo 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(130,20,0);stop-opacity:1"/>
    </linearGradient>
  </defs>

</svg>
```

El resultado visual sería el siguiente:



- **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"` : Coordendas "x" e "y" del centro del círculo de degradado. En estos atributos y los siguientes podemos indicar una medida, un número (píxeles) 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.

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.

Gradientes radiales de color

Gradiente lineal Canvas.

2.6.- 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 par las transformaciones son los mismos que en CSS3. Para ello se utiliza el atributo transform, al cual se la aplican como valor diferentes métodos.

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

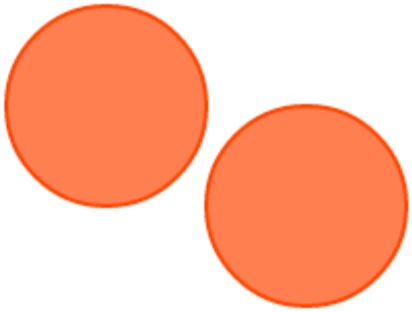
  transform="translate(x,y)"

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

  Si además, lo que se necesita trasladar es un conjunto de elementos: habrá que englobralos 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="270" height="200">
  <g>
    <circle cx=90 cy=90 r=50 stroke="OrangeRed" stroke-width=2 fill="Coral">
  </g>

  <g transform="translate(100,50)">
    <circle cx=90 cy=90 r=50 stroke="OrangeRed" stroke-width=2 fill="Coral">
  </g>
</svg>
```

El resultado visual sería el siguiente:



- Escalado. Método 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.

- Rotación. Método 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 reliaza la rotación del texto dibujado:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="300" height="200">
  <g>
  <text x="100" y="90" style="font: italic bold 1em tahoma; fill: darkblue;"> ROTAR </text>
  </g>

  <g transform="rotate(40)">
  <text x="100" y="90" style="font: italic bold 1em tahoma; fill: darkblue;"> ROTAR  </text>
  </g>
</svg>
```

- Sesgado. Método skew. permite deformar el elemento vairando el ángulo de sus coordenadas. Su sintaxis es la siguiente:

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

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

- El método 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 angulo cuya tangente es la indicada. El parámetro "c" hace lo mismo con el eje "y".

Autoevaluación

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

- [ ] matrix
- [ ] scale()
- [ ] rotate()
- [ ] translate()

Mostrar retroalimentación

2.7.- 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>
```

donde tenemos algunos de los siguientes atributos:

- attributeType: Indica el tipo de atributo al que se le aplicará la animacion, como valores podeos 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

- set: esta etiqueta no produce una animación sino un cambio de estado instantánteo de un elemento. Utiliza los mismos atributos que animate, sin embargo el cambio no es gradual.

- 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 vió 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.

- 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.

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.

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

<svg viewBox="-10 -10 610 510" width="600" height="500">

      <!-- Ejemplo de animación con animateMotion con path-->
      <!-- En este caso dibujos un trazado con la etiqueta path-->
      <path d="m150 150 v50 h50 v50 h50 v50 h50 v-50 h50 v-50 h50 v-50 h50 v-50 h-50 v-50 h-50 v-50 h-50 v50 h-50 v50 h-50 v50 z" s

      <!-- Dibujamos una pelota-->
      <circle id="pelota" cx=0 cy=0 r=5 stroke="black" stroke-width=2 fill="aqua">

      <!-- Creamos la animación de tal forma que la pelota seguira la ruta path que hemos definido en animateMotion y que es igual a la qu
      <!-- Hemos definido un identificador pelota en la figura creada con la etiqueta circle -->
      <!-- La animación comenzará cuando hagamos click en la pelota id="pelota"-->
         <animateMotion id="animacion_1" begin="pelota.click"  path="m150 150 v50 h50 v50 h50 v50 h50 v-50 h50 v-50 h50 v-50 h50 v
      <!-- Le hemos asignado un identificado a la animación que nos servira para indicarle a la siguiente animación que empiece cuando ter
      </circle>


      <!-- Ejemplo de animación con animateMotion con mpath-->
      <!-- Definimos una transformación para desplazar nuestro origen, no es necesario realizar esto -->
      <g transform="translate(0,20)">
         <!-- Definimos una ruta para ver como se desplaza nuestro objeto -->
         <path id="ruta" d="M150 350 v25 h25 v25 h25 v25 h25 v-25 h25 v-25 h25 v-25 h25 v-25 h-25 v-25 h-25 v-25 h-25 v25 h-25 v25
         <!-- Definimos un rectángulo que animaremos y le asignamos un identificador -->
         <rect id="rectangulo" cx=0 cy=0 width="10" height="10" stroke="green" stroke-width=2 fill="gold">
            <!-- Definimos nuestra animación e indicamos que empiece cuando termine animacion_1, además indicamos que el objeto se qu
            <animateMotion begin="animacion_1.end"  dur="3s"  repeatCount="2" rotate="auto" fill="freeze">
         <!-- Asignamos a la animación la ruta que va a aseguir-->
         <mpath xlink:href="#ruta"/>
         </animateMotion>
      </g>
</svg>
</body>
</html>
```

Debes conocer

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.

Para saber más

Toda la información que proporciona la W3C sobre la etiqueta animate se encuentra disponible en el siguiente sitio:

Animate por W3C

Y, como material extra para profundizar en este apartado se recomienda el último capítulo del tutorial que se ha ido recomendando en esta unidad:

Más sobre animaciones en SVG.

Para finalizar dejamos el enlace a un artículo donde se muestra como animar un logo, el cual lleva por título:

Como animar un logo y no morir en el intento.

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.

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Junta de Andalucía

Antes de cualquier uso leer detenidamente el siguente Aviso legal

Historial de actualizaciones

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 amplian 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 idevice 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.