Tutorial Canvas

<canvas> es un elemento HTML el cual puede ser usado para dibujar gráficos usando scripts (normalmente JavaScript). Este puede, por ejemplo, ser usado para dibujar gráficos, realizar composición de fotos o simples animaciones.

Uso básico de Canvas

El elemento<canvas>

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

A primera vista, un elemento <canvas> es parecido al elemento <img>, con la diferencia que este no tiene los atributos src y alt. El elemento <canvas> tiene solo dos atributos - width y height. Ambos son opcionales y pueden ser definidos usando propiedades DOM. Cuando los atributos ancho y alto no estan especificados, el lienzo se inicializara con **300 pixels** ancho y **150 pixels** de alto.

El elemento **<canvas>** puede ser estilizado como a cualquier imagen normal (margin, border, background, etc). Estas reglas, sin embargo, no afectan a lo dibujado sobre el canvas.

<canvas> crea un lienzo de dibujo fijado que expone uno o más contextos renderizados, los cuales son usados para crear y manipular el contenido mostrado.

El canvas está inicialmente en blanco. Para mostrar alguna cosa, un script primero necesita acceder al contexto a renderizar y dibujar sobre este. El elemento **<canvas>** tiene un método llamado **getContext(),** usado para obtener el contexto a renderizar y sus funciones de dibujo. Para gráficos 2D, que son los que vamos a ver, su especificación es "2d".

```
var canvas = document.getElementById('tutorial');
var ctx = canvas.getContext('2d');
```

La primera línea regresa el nodo DOM para el elemento **<canvas>** llamando al método **document.getElementById().** Una vez tienes el elemento nodo, tu puedes acceder al contexto de dibujo usando su método **getContext().**

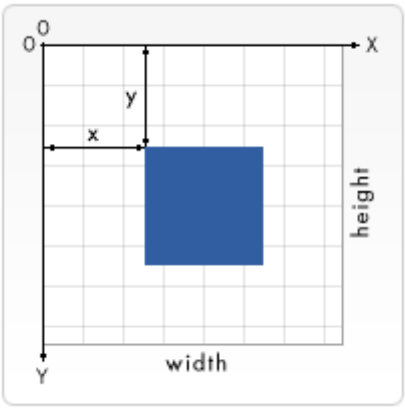
Plantilla Ejemplo:

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

Dibujando formas con canvas

La cuadrícula

Antes de empezar a dibujar, tenemos que hablar de la cuadrícula del canvas o del **espacio de coordenadas**. Nuestra estructura HTML de la página anterior tenía un elemento de canvas de 250 píxeles de ancho y 250 píxeles de alto.



Normalmente, 1 unidad en la cuadrícula corresponde a 1 píxel en el canvas. El origen de esta cuadrícula se sitúa en la esquina superior izquierda en la coordenada (0,0). Todos los elementos se colocan en relación con este origen. Así que la posición de la esquina superior izquierda del cuadrado azul se sitúa a x píxeles de la izquierda y a y píxeles de la parte superior, en la coordenada (x,y).

Dibujar rectángulos.

<canvas> sólo admite dos formas primitivas: rectángulos y trazados (listas de puntos conectados por líneas).

Hay tres funciones que dibujan rectángulos en el canvas:

- **fillRect(x, y, width, height)**. Dibuja un rectángulo relleno.
- **strokeRect(x, y, width, height)**. Dibuja un contorno rectangular.
- **clearRect(x, y, width, height).** Borra el área rectangular especificada, haciéndola totalmente transparente.

Cada una de estas tres funciones toma los mismos parámetros. **x** e **y** especifican la posición en el canvas (relativa al origen) de la esquina superior izquierda del rectángulo. **width** y **height** proporcionan el tamaño del rectángulo.

EJERCICIO1 Rectángulos: crear un canvas con dos rectángulos. El primero será un rectángulo lleno posicionado en x=y=10 y tamaño 100x100. El segundo será un contorno de rectángulo situado en x=y=150 y tamaño 50x50

Dibujar trazados (paths)

Un trazado es una lista de puntos, conectados por segmentos de líneas que pueden ser de diferentes formas, curvas o no, de diferente anchura y de diferente color. Un trazado, o incluso un sub-trazado, puede ser cerrado. Para hacer formas usando trazados, damos algunos pasos adicionales:

1. Primero, se crea el camino.
2. Luego, se utiliza comandos de dibujo para dibujar en el trazado.
3. Una vez creado el trazado, puedes trazar o rellenar el trazado para renderizarlo.

Las funciones utilizadas para realizar estos pasos son:

- **beginPath().** Crea un nuevo trazado.
- **Métodos de trazado (path).** Métodos para establecer diferentes trazados para los objetos.
- **closePath().** Añade una línea recta al trazado, que va al inicio del sub-trazado actual.
- **stroke().** Dibuja la forma trazando su contorno.
- **fill().** Dibuja una forma sólida rellenando el área de contenido del trazado.

El primer paso para crear un trazado es llamar a **beginPath()**. Cada vez que se llama a este método, la lista se restablece y podemos empezar a dibujar nuevas formas.

El segundo paso es llamar a los métodos que realmente especifican los trazados a dibujar. Los veremos a continuación.

El tercer paso, y opcional, es llamar a **closePath().** Este método intenta cerrar la forma dibujando una línea recta desde el punto actual hasta el inicio. Si la forma ya ha sido cerrada o sólo hay un punto en la lista, esta función no hace nada.

Nota: Cuando se llama a fill(), cualquier forma abierta se cierra automáticamente, por lo que no es necesario llamar a closePath(). Este **no** es el caso cuando se llama a stroke().

Mover la pluma.

Una función muy útil, que en realidad no dibuja nada sino que se convierte en parte de la lista de trazados descrita anteriormente, es la función **moveTo().** La mejor manera de pensar en esto es como levantar un bolígrafo o un lápiz de un lugar en un pedazo de papel y colocarlo en el siguiente.

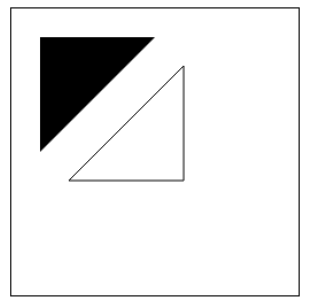
- **moveTo(x, y)** Mueve la pluma a las coordenadas especificadas por x e y.

Métodos de trazado.

Líneas. Para dibujar líneas rectas, utilice el método lineTo().

- **lineTo(x, y)** Dibuja una línea desde la posición actual de dibujo hasta la posición especificada por x e y.

EJERCICIO2 Triángulos. Dibuja dos triángulos, uno relleno que empezara en la posición x=y=25 y de 100px de lado y otro contorneado que empezara en x=y=150 y de 100px de lado.



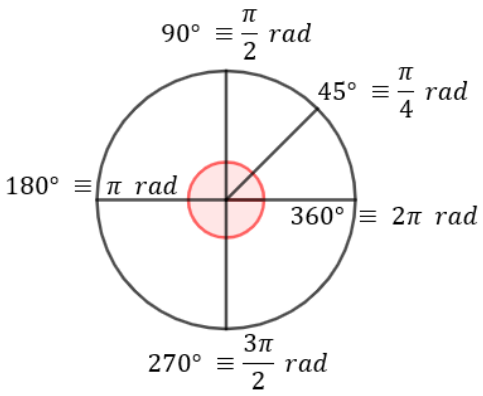
Arcos. Para dibujar arcos o círculos, utilizamos el método **arc().**

- **arc(x, y, radius, startAngle, endAngle, counterclockwise)**

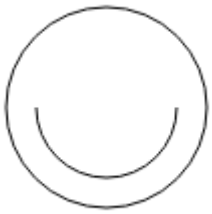
Dibuja un arco centrado en la posición *(x, y)* con radio *r* que comienza en *startAngle* y termina en *endAngle* yendo en la dirección indicada por *counterclockwise* (por defecto en el sentido de las agujas del reloj).

El método **arc**, que toma seis parámetros: x e y son las coordenadas del centro del círculo sobre el que se dibujará el arco. El parámetro **radio** se explica por sí mismo. Los parámetros **startAngle** y **endAngle** definen los puntos inicial y final del arco en radianes, a lo largo de la curva del círculo. Se miden desde el eje x. El parámetro **counterclockwise** es un valor Booleano que, cuando es **true**, dibuja el arco en sentido contrario a las agujas del reloj; en caso contrario, el arco se dibuja en sentido de las agujas del reloj.

Nota: Los ángulos en la función **arc** se miden en radianes, no en grados. Para convertir los grados en radianes puedes utilizar la siguiente expresión de JavaScript: **radianes = (Math.PI/180)*grados.**



EJERCICIO3 Arcos. Dibujar un circulo completo y en la parte baja un semicírculo como si fuera una cara con una sonrisa.



Rectángulos

Existe también el método **rect(),** que añade un trazado rectangular a un trazado actualmente abierto.

- **rect(x, y, width, height)**

Dibuja un rectángulo cuya esquina superior izquierda está especificada por (x, y) con el width y height especificados.

Antes de que se ejecute este método, se llama automáticamente al método **moveTo()** con los parámetros (x,y). En otras palabras, la posición actual de la pluma se restablece automáticamente a las coordenadas por defecto.

Objetos Path2D

Para simplificar el código y mejorar el rendimiento, el objeto **Path2D**, disponible en las versiones recientes de los navegadores, le permite almacenar en caché o grabar estos comandos de dibujo. De este modo, se pueden reproducir los trazados rápidamente.

- **Path2D()** El constructor **Path2D()** devuelve un objeto Path2D recién instanciado.

```
new Path2D();       // Objeto Path2D vacío
new Path2D(path);   // Copia de otro objecto Path2D
new Path2D(d);      // Path2D a partir de datos de un trazado.
```

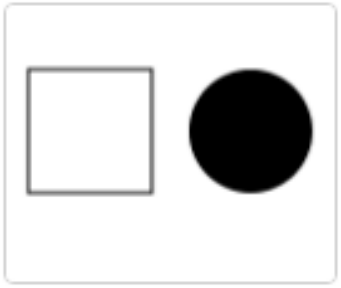
Todos los Métodos de trazado como moveTo, rect, arc o quadraticCurveTo, etc., están disponibles en los objetos **Path2D.**

Ejemplo:

```
const rectangle = new Path2D();
rectangle.rect(10, 10, 50, 50);

const circle = new Path2D();
circle.arc(100, 35, 25, 0, 2 * Math.PI);

ctx.stroke(rectangle);
ctx.fill(circle);
```



Aplicación de colores.

Colores

Si queremos aplicar colores a una forma, hay dos propiedades importantes que podemos usar: **fillStyley strokeStyle.**

- **fillStyle = color** Establece el estilo utilizado al rellenar formas.
- **strokeStyle = color** Establece el estilo de los contornos de las formas.

Color es una cadena que representa un CSS **<color>,** un objeto de degradado o un objeto de patrón. . De forma predeterminada, el trazo y el color de relleno se establecen en negro (valor de color CSS #000000).

Nota: cuando establece la propiedad **strokeStyle y/o fillStyle**, el nuevo valor se convierte en el valor predeterminado para todas las formas que se dibujan a partir de ese momento.

Ejemplos de color naranja:

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

Si añadimos este color al ejemplo anterior quedaria: