HTML5 – CANVAS

<canvas> es un elemento HTML el cual puede ser usado para dibujar gráficos usando scripts (normalmente JavaScript). Este puede, por instancia, ser usado para dibujar gráficos, realizar composición de fotos o simples (y no tan simples) animaciones

<canvas> fue introducido primero por Apple para el Mac OS X Dashboard y después implementado en Safari y Google Chrome. Navegadores basados enGecko 1.8, tal como Firefox 1.5, que también soportan este elemento. El <canvas> es un elemento parte de las especificaciones de la WhatWG Web applications 1.0 mejor conocida como HTML5.

Antes de Empezar

Usar el elemento <canvas> no es algo muy díficil pero necesita saber y entender los aspectos básicos del HTML y JavaScript. El elemento <canvas> no está soportado en navegadores viejos, pero están soportado en la mayoría de las versiones más recientes de los navegadores. El tamaño por defecto del canvas es 300px * 150px [ancho (width) * alto (height)]. Pero se puede personalizar el tamaño usando las propiedades height y width de CSS. Con el fin de dibujar gráficos en el lienzo <canvas> se utiliza un objeto de contexto de JavaScript que crea gráficos sobre la marcha.

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 puede ser arbitrariamente redimensionado por CSS, pero durante el renderizado la imagen es escalada para ajustarse al tamaño de su layout.

Nota: Si su renderizado se ve distorsionado, pruebe especificar los atributos width y height explícitamente en los atributos del <canvas> , y no usando CSS.

El atributo id no está especificado para el elemento <canvas> pero es uno de los atributos globales en HTML el cual puede ser aplicado a cualquier elemento HTML (como class por ejemplo). Siempre es buena idea proporcionar un id porque esto hace más fácil identificarlo en un script.

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. Cuando no tenemos reglas de estilo aplicadas al canvas, este será completamente transparente.

Contenido alternativo

El elemento <canvas> se diferencia de un tag `<img>` por esto, similar a los elementos `<video>`, `<audio>` o `<picture>`, es fácil definir contenido alterno para mostrarse en navegadores viejos que no soporten el elemento <canvas>, como versiones de Internet Explorer previas a la versión 9 o navegadores de texto. Siempre debes proporcionar contenido alternativo para mostrar en estos navegadores.

Proporcionar contenido alternativo es muy explicito: solo debemos insertar el contenido alterno dentro del elemento <canvas>. Los navegadores que no soporten `<canvas>`ignoraran el contenedor y mostrarán el contenido indicado dentro de este. Navegadores que soporten `<canvas>` ignoraran el contenido en su interior (de las etiquetas), y mostrarán el canvas normalmente.
Por ejemplo, podremos proporcionar un texto descriptivo del contenido del canvas o proveer una imagen estática del contenido rederizado. Este puede verse como esto:

```
<canvas id="stockGraph" width="150" height="150">
  current stock price: $3.15 +0.15
</canvas>

<canvas id="clock" width="150" height="150">
  <img src="images/clock.png" width="150" height="150" alt=""/>
</canvas>
```

El contexto de renderización

`<canvas>` crea un lienzo de dibujo fijado que expone uno o mas contextos renderizados, los cuales son usados para crear y manipular el contenido mostrado. Nos centraremos en renderización de contextos 2D. Otros contextos deberán proveer diferentes tipos de renderizaciones; por ejemplo, WebGL usa un 3D contexto ("experimental-webgl") basado sobre OpenGL ES.

El canvas esta 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 method llamado `getContext()`, usado para obtener el contexto a renderizar y sus funciones de dibujo. `getContext()` toma un parámetro, el tipo de contexto. Para graficos 2D, su especificacion es "2d".

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

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

Comprobando soporte

Desde JavaScript puede comprobarse el soporte de canvas con un simple test de presencia del metodo `getContext()`:

```
var canvas = document.getElementById('tutorial');

if (canvas.getContext){
  var ctx = canvas.getContext('2d');
  // drawing code here
} else {
  // canvas-unsupported code here
}
```

Un esqueleto de plantilla

Esta es una plantilla minimalista, la cual usaremos como punto de partida para posteriores ejemplos.

```
<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>
```

El script incluye una función llamada draw(), la cual es ejecutada una vez finalizada la carga de la pagina, usando el evento load del documento. Esta función, o una parecida, podría también ser llamada usando `window.setTimeout()`,`window.setInterval()`, o cualquier otro manejador de evento.

Un simple ejemplo

Para comenzar, daremos un vistazo a un simple ejemplo que dibuja dos rectángulos que se intersectan, uno de los cuales tiene transparencia alpha.

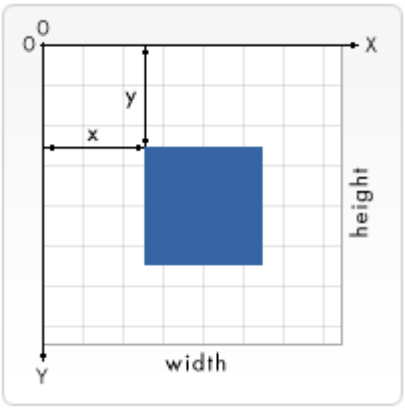
```
<html>
 <head>
  <script type="application/javascript">
    function draw() {
      var canvas = document.getElementById("canvas");
      if (canvas.getContext) {
        var ctx = canvas.getContext("2d");

        ctx.fillStyle = "rgb(200,0,0)";
        ctx.fillRect (10, 10, 55, 50);

        ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
        ctx.fillRect (30, 30, 55, 50);
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```

Este ejemplo se parece a esto:

Dibujando formas con canvas



Antes de que podamos empezar a dibujar, necesitamos hablar sobre la cuadrícula de canvas o espacio de coordenadas. La plantilla anterior tenía un elemento canvas con un height y un width de 150 pixeles. A la derecha, puedes ver este canvas con la cuadrícula normal superpuesta. Normalmente una unidad en la cuadrícula corresponde a un pixel en el elemento canvas. El origen de esta cuadrícula está posicionado en la esquina superior izquierda (coordenada (0,0)). Todos los elementos están posicionados de manera relativa a este punto, así que la posición de la esquina superior izquierda del cuadrado azul es de x pixeles desde la izquierda y y pixeles desde arriba (coordenada (x,y)).

Dibujando Rectángulos

`<canvas>` sólo soporta una forma primitiva: rectángulos. Otras formas deben ser creadas por la combinación de uno mas pasos. Afortunadamente, Tenemos una variedad de funciones que hacen posible componer formas muy complejas.

Primero veamos en el rectángulo. Aquí hay tres funciones que podemos usar en el canvas:

`fillRect(x, y, width, height)`
Dibuja un rectángulo relleno.

`strokeRect(x, y, width, height)`
Dibuja el contorno de un rectángulo.

`clearRect(x, y, width, height)`
Borra un área rectangular especificada, por lo que es totalmente transparente.

Cada una de estas tres funciones toma los mismos parámetros. X e Y especifican la posición del canvas (en relación con el origen) desde la esquina superior izquierda del rectángulo. También especifica los parámetros de anchura y altura que proporcionan el tamaño del rectángulo.

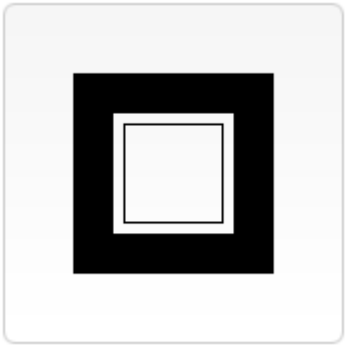
A continuación se muestra la función draw () de la página anterior, pero ahora haciendo uso de estas tres funciones.

Ejemplo de forma rectangular

```
function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext) {
    var ctx = canvas.getContext('2d');

    ctx.fillRect(25,25,100,100);
    ctx.clearRect(45,45,60,60);
    ctx.strokeRect(50,50,50,50);
  }
}
```

La producción de este ejemplo se muestra a continuación.



La función fillRect () dibuja un cuadrado grande negro de 100 píxeles en cada lado. La función clearRect () ,borra un cuadrado de 60x60 píxeles del centro, y luego strokeRect () es llamado para crear un contorno rectangular de 50x50 píxeles dentro del espacio despejado.

Dibujando caminos

Hacer formas usando caminos requiere una serie de pasos extra, primero se debe crear el camino y a continuación, usar las intrucciones de dibujo en el camino, finalmente se cierra el camino

Funciones para hacer esto:

`beginPath()`
Crea un nuevo camino, los futuros comandos de dibujo irán dentro del camino

`closePath()`
Cierra el camino, los futuros comandos de dibujo irán una vez más al contexto.

`stroke()`
Dibuja la forma

`fill()`
Dibuja una forma sólida

El primer paso es crear el camino llamando a la función `beginPath(), internamente los caminos, son almacenados como una lista de sub-caminos (líneas, arcos,etc) que juntos forman una figura. Cada vez que este método es invocado la lista se resetea y podemos empezar a dibujar.`

El segundo paso es llamar a los métodos que en realidad, especifican los caminos que serán dibujados. El tercer paso es cerrar el camino con `closePath().`

Dibujando un triángulo

Por ejemplo, el código para dibujar un triángulo sería algo así:

```
function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext){
    var ctx = canvas.getContext('2d');

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

El resultado sería el siguiente:

Moviendo el lápiz

Una función que en realidad no dibuja pero esmuy útil, es la función `moveTo()`

`moveTo(x, y)`
Mueve el lápiz a las coordenadas especificadas

```
function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext){
    var ctx = canvas.getContext('2d');

    ctx.beginPath();
    ctx.arc(75,75,50,0,Math.PI*2,true); // Outer circle
    ctx.moveTo(110,75);
    ctx.arc(75,75,35,0,Math.PI,false);   // Mouth (clockwise)
    ctx.moveTo(65,65);
    ctx.arc(60,65,5,0,Math.PI*2,true);  // Left eye
    ctx.moveTo(95,65);
    ctx.arc(90,65,5,0,Math.PI*2,true);  // Right eye
    ctx.stroke();
  }
}
```

El resultado es el siguiente:

Líneas

Para dibujar líneas rectas usamos la función `lineTo()` .

`lineTo(x, y)`

Dibuja una línea desde la posición actual a la posición especificada por x e y

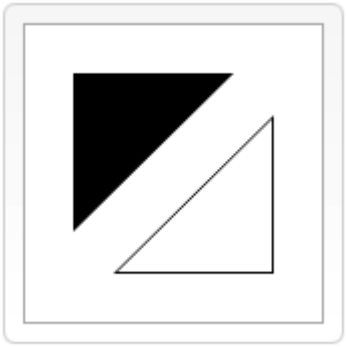
Este ejemplo dibuja dos triángulos, uno relleno y otro vacío:

```
function draw() {

  var canvas = document.getElementById('canvas');
  if (canvas.getContext){
    var ctx = canvas.getContext('2d');

    // Filled triangle
    ctx.beginPath();
    ctx.moveTo(25,25);
    ctx.lineTo(105,25);
    ctx.lineTo(25,105);
    ctx.fill();

    // Stroked triangle
    ctx.beginPath();
    ctx.moveTo(125,125);
    ctx.lineTo(125,45);
    ctx.lineTo(45,125);
    ctx.closePath();
    ctx.stroke();
  }
}
```



Arcos

Para dibujar arcos o círculos usaremos la función `arc()`

`arc(x, y, radius, startAngle, endAngle, anticlockwise)`
Dibuja un arco

Este método toma cinco parámetros **x** e **y** son las coordenadas del centro del círculo donde el arco será dibujado, radius corresponde con el radio del círculo. Los parámetros `startAngle` and `endAngle definen el puntos inicial y final del arco en radianes.` El parámetro `anticlockwise es booleano, cuando se establece a true, se dibuja el arco en el sentido contrario a las agujas del reloj.`

Note: Los ángulos son medidos en radianes no en grados, para convertir grados en radianes puedes utilizar la función JavaScript: `radians = (Math.PI/180)*degrees`.

A continuación podemos ver un ejemplo completo de dibujado de arcos:

```
function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext){
    var ctx = canvas.getContext('2d');

    for(var i=0;i<4;i++){
      for(var j=0;j<3;j++){
        ctx.beginPath();
        var x              = 25+j*50;               // x coordinate
        var y              = 25+i*50;               // y coordinate
        var radius         = 20;                    // Arc radius
        var startAngle     = 0;                     // Starting point on circle
        var endAngle       = Math.PI+(Math.PI*j)/2; // End point on circle
        var anticlockwise  = i%2==0 ? false : true; // clockwise or anticlockwise

        ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise);

        if (i>1){
          ctx.fill();
        } else {
          ctx.stroke();
        }
      }
    }
  }
}
```