IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

PRACTICA 3: CLASES JAVASCRIPT - CANVAS

Duración estimada: 200 min.

Objetivos:

- ***Manejo canvas con javascript***
- ***Clases, clases abstractas, métodos, etc***

Introducción:

1. **Canvas:**

Canvas significa en español algo así como lienzo y es básicamente eso, un área donde podemos dibujar como si fuera un lienzo.
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. Si el tamaño del CSS no respeta el ratio del canvas inicial, este aparecerá distorsionado.

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.

2. **Compatibilidad de canvas:**

El canvas se desarrolló inicialmente por Apple para su navegador Safari y luego fue utilizado y estandarizado por la organización WHATWG para incorporarlo a HTML 5. Posteriormente también ha sido adoptado por navegadores como Firefox y Opera.

Por lo que respecta a Chorme, es un navegador que utiliza el mismo motor de renderizado que Safari, por lo que también soporta el elemento Canvas

De entre los navegadores más habituales sólo nos queda por soportar canvas el siempre polémico Internet Explorer. La última versión del navegador en el momento de escribir este artículo, Internet Explorer 8, no soporta canvas con funciones nativas, pero existen diversos proyectos y plugins que pueden ampliar las funcionalidades del navegador para dar soporte a este nuevo elemento del HTML 5.

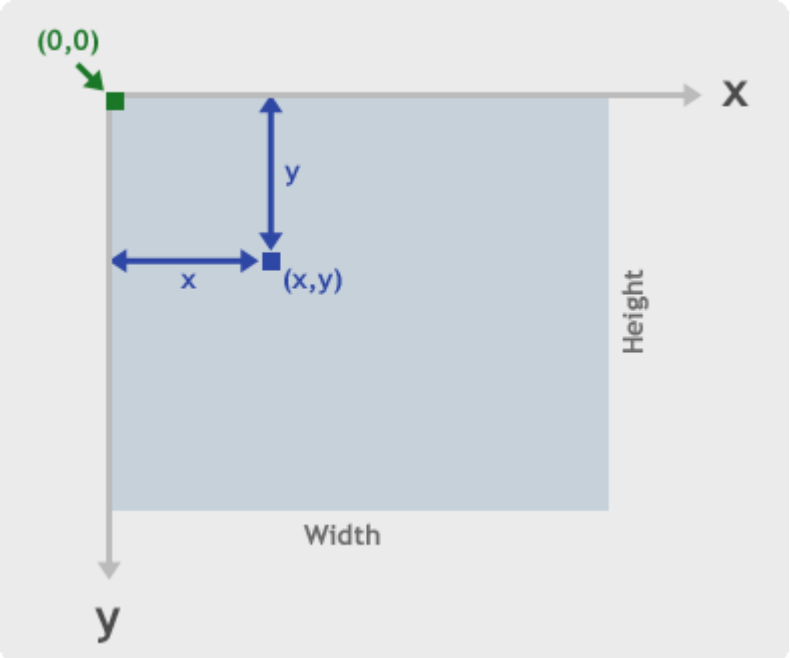
3. Aplicaciones de uso de Canvas

Canvas permite dibujar en la página y actualizar dinámicamente estos dibujos, por medio de scripts y atendiendo a las acciones del usuario. Todo esto da unas posibilidades de uso tan grandes como las que disponemos con el plugin de Flash, en lo que respecta a renderización de contenidos dinámicos. Las aplicaciones pueden ser grandes como podamos imaginar, desde juegos, efectos dinámicos en interfaces de usuario, editores de código, editores gráficos, aplicaciones, efectos 3D, etc.

4. Eje de coordenadas del canvas:

Para posicionar elementos en el canvas tenemos que tener en cuenta su eje de coordenadas en dos dimensiones, que comienza en la esquina superior izquierda del lienzo.
El lienzo producido por canvas tendrá unas dimensiones indicadas con los atributos width y height en la etiqueta CANVAS. Por tanto, la esquina superior izquierda será el punto (0,0) y la esquina inferior derecha el punto definido por (width-1,height-1), osea, el punto máximo de coordenadas marcado por su anchura y altura.

Podemos ver el siguiente diagrama para tener una idea exacta de las dimensiones y coordenadas en un canvas.



Cualquier punto dentro del canvas se calcula con la coordenada (x,y), siendo que la x crece según los pixel a la derecha y la y con los pixel hacia abajo.

Para dibujar cualquier tipo de forma en el canvas necesitaremos posicionarla con respecto a las coordenadas que acabamos de ver. En el ejemplo del artículo anterior, vimos que para dibujar un rectángulo necesitamos varios valores:

contexto.fillRect(10, 10, 100, 70);

Pasos para trabajar con canvas

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 metodo llamado getContext(), usado para obtener el contexto a renderizar y sus funciones de dibujo. getContext() toma un parametro, el tipo de contexto. Para graficos 2D, su especificacion es "2d".

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

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

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

5. Pintar Líneas y caminos

El modo de dibujar líneas es diferente al de dibujar cuadrados. De hecho, funciona como si del lápiz se tratara. Primero la posicionamos y la acercamos al papel, y vamos moviéndola mientras queramos que dibuje.

```
ctx.beginPath(); // Inicia la operación de dibujo.
ctx.moveTo(40, 40); // Coloca el puntero a 40, 40.
ctx.lineTo(340, 40); // Dibuja una línea hasta 340, 40.
ctx.closePath(); // Cierra el path.
ctx.stroke(); // Lo transforma en dibujo a línea.
```

6. Círculos

No existe un método específico para dibujar círculos con canvas, pero sí que tenemos las herramientas para dibujarlos. De este modo, podemos utilizar el método arco:

```
ctx.beginPath(); // Iniciamos el dibujo de un camino.
ctx.arc(100, 90, 50, 0, Math.PI*2, false); // Dibujamos un círculo.
ctx.closePath(); // Cerramos el camino.
ctx.fill(); // Llenamos de contenido.
```

El método arco admite seis parámetros. Los dos primeros son el punto central de la circunferencia; el segundo, el radio; el tercero, el ángulo inicial; el cuarto, el ángulo final (los ángulos en radianes) y, finalmente, un booleano para marcar si se debe dibujar en la dirección de las agujas del reloj o no.

Para poder convertir los grados en radianes, podemos emplear la siguiente fórmula:

```
var degrees = 1; // 1 grado
var radians = degrees * (Math.PI / 180); // 0,0175 radianes 360 grados 360 grados, son 2*Math.PI.
```

7. Estilos de línea(stroke) y de relleno(fill)

Otra de las herramientas fáciles es la posibilidad de personalizar los estilos de las líneas y rellenos que dibujamos, así:

```
ctx.fillStyle = "rgb(255, 0, 0)"; // Definiremos un color de relleno rojo.
ctx.strokeStyle = "rgb(255, 0, 0)"; // Definiremos un color de línea.
ctx.lineWidth = 20; // Define el grueso de la línea.
```

8. Texto

Disponemos de un método que nos permite escribir con el canvas:

```
var text = "Hello, World!";
ctx.font = "italic 20px serif";
ctx.fillText(text, 30, 80);
```

9. Añadiendoimágenes

Podemos añadir imágenes directamente a un objeto canvas. Para hacerlo:

```
var img = new Image();
img.src = "pathalaimatge.gif"
ctx.drawImage(img,0,0);
```

Al ser la imagen un objeto remoto, convendría esperar su descarga; así podríamos, utilizando jQuery, programar un pequeño preload:

```
var image = new Image();
image.src = "prueba.jpg";
$(image).load(function() { //Código jQuery
 ctx.drawImage(image, 0, 0);
});
```

10. Transformaciones

1. Transformaciones y escalas

Podemos transformar el modo como dibujamos un objeto, o ciertos objetos, empleando los métodos translate y rotate. Con el primero, transportamos las coordenadas 0,0 a otro punto. Así, si efectuamos:

ctx.translate(100,100);

La coordenada 0,0 pasará a ser la 100,100, y si nosotros dibujásemos un cuadrado en la 0,0,10,10, nos aparecería en la 100,100,110,110.

Lo mismo sucede con la rotación. Si nosotros realizamos:

ctx.rotate(Math.PI/4)

Rotaremos el canvas 45°, y todo lo que dibujemos nos aparecerá rotado.

2. Compositing

Componer elementos se refiere a la manera como se combinarán los nuevos elementos que añadamos al canvas con los que ya existen. Así, cuando dibujemos un objeto en el canvas, podremos decidir cómo queremos que este se superponga con el resto de los objetos existentes.
De este modo, cuando añadamos un nuevo objeto podremos decidir si queremos que se superponga o lo contrario. Y también si queremos que al superponerse le aplique un canal alpha o queremos que se haga un xor de píxeles. Así podemos definir el modelo de composición con:

ctx.globalCompositeOperation = "";

Donde el valor puede ser:

- *source-over La imagen origen se situará sobre el canvas.*
- *destination-over La imagen se situará debajo de los objetos existentes en el canvas.*
- *source-in El origen se dibujará en la zona donde existe superposición.*
- *destination-in El destino se dibujará en la zona donde existe superposición.*
- *lighter,copy,xor Lighter genera la superposición como color 255, blanco.*
- *Copy dibuja el origen en vez del destino.Cualquier parte que se superponga quedará transparente.*

3. Sombras:

También podemos definir y pintar sombras en un objeto mientras lo definimos. Para hacerlo, es necesario que definamos las propiedades adecuadas para que después, al generar el objeto, nos aparezcan correctamente:

- *ctx.shadowBlur = 20;*
- *ctx.shadowColor = "rgb(0, 0, 0)";*

- *ctx.shadowOffsetX = 10;*
- *ctx.shadowOffsetY = 10;*
- *ctx.fillRect(50, 50, 100, 100);*

En este caso, estamos generando una sombra de color negro con un desenfocado de 20px y desplazada 10px.

4. Guardar el estado actual como imagen

El canvas, a efectos prácticos, es como si tuviéramos una gran imagen y, como tal, la podemos guardar como imagen con el siguiente comando:

var dataURL = canvas.get(0).toDataURL();

que nos generará una imagen en formato png y codificada en base64. Al mismo tiempo, este string, que podemos enviar a un servidor y descodificar como imagen, también puede ser asignado a un objeto imagen para entonces abrirlo directamente con el navegador:

var img = $("<img></img>"); img.attr("src", dataURL);

canvas.replaceWith(img);

De este modo, si el usuario quiere guardar la imagen, simplemente haciendo clic con el botón derecho del ratón lo puede hacer, puesto que hemos convertido el objeto canvas (vectorial) en imágenes.

También podríamos generar un archivo descargable enviándola primero al servidor, realizando la conversión a binario allí, y lanzando las cabeceras adecuadas junto con los bytes de la imagen.

11. Animación:

El objeto canvas no nos proporciona un mecanismo para generar animaciones de manera directa, pero sí que lo podemos hacer gracias al Javascript. El principio con pseudo-código es:

1. dibujaremos.
2. esperaremos un intervalo de tiempo.
3. borraremos la pantalla.
4. actualizaremos los valores.

Volveremos al paso 1.

De este modo, podemos mostrar un pequeño ejemplo que demuestre el modo como podemos animar cosas: ***(Ejemplo 4)***

Secuencia y desarrollo:

Ejemplo1: Implementa el siguiente ejemplo que permite dibujar en un canvas

```
<html>
<head>
  <title>Probando canvas</title>
<script>
window.onload = function(){
  //Recibimos el elemento canvas
  var elemento = document.getElementById('micanvas');
  //Comprobación sobre si encontramos un elemento
  //y podemos extraer su contexto con getContext(), que indica compatibilidad con canvas
  if (elemento && elemento.getContext) {
    //Accedo al contexto de '2d' de este canvas, necesario para dibujar
    var contexto = elemento.getContext('2d');
    if (contexto) {
      //Si tengo el contexto 2d es que todo ha ido bien y puedo empezar a dibujar en el canvas
      //Comienzo dibujando un rectángulo
      contexto.fillRect(0, 0, 150, 100);
      //cambio el color de estilo de dibujo a rojo
      contexto.fillStyle = '#cc0000';
      //dibujo otro rectángulo
      contexto.fillRect(10, 10, 100, 70);
    }
  }
}

</script>
</head>

<body>

<canvas id="micanvas" width="200" height="100">
Este texto se muestra para los navegadores no compatibles con canvas.
<br>
Por favor, utiliza Firefox, Chrome, Safari u Opera.
</canvas>

</body>
</html>
```

Ejemplo2: Implementa el siguiente ejemplo que permite dibujar en un canvas

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript">

window.onload = function() {


 //Recibimos el elemento canvas
var elemento = document.getElementById('tutorial');
var ctx = elemento.getContext('2d');


//Comprobación sobre si encontramos un elemento
//y podemos extraer su contexto con getContext(), que indica compatibilidad con canvas

 if (elemento && elemento.getContext) {
  //Accedo al contexto de '2d' de este canvas, necesario para dibujar
    if (ctx) {
                    //Si tengo el contexto 2d es que todo ha ido bien y puedo empezar a dibujar en el
canvas
                ctx.fillStyle = "rgb(200,0,0)";
                ctx.fillRect (10, 10, 50, 50);
                ctx.arc(100, 90, 50, 0, Math.PI*2, false); // Dibujamos un círculo.
                ctx.stroke(); // Lo transforma en dibujo a línea.
                }
        }

var canvas = document.getElementById('tutorial1');
var ctx1 = canvas.getContext('2d');
var text = "Hola Mundo";

//Comprobación sobre si encontramos un elemento
//y podemos extraer su contexto con getContext(), que indica compatibilidad con canvas

 if (canvas&& canvas.getContext) {
  //Accedo al contexto de '2d' de este canvas, necesario para dibujar
    if (ctx1) {
                ctx1.fillStyle = "rgb(255, 0, 0)"; // Definiremos un color de relleno rojo.
                ctx1.strokeStyle = "rgb(255, 0, 0)"; // Definiremos un color de línea.
                ctx1.lineWidth = 20; // Define el grueso de la línea.
                ctx1.font = "italic 20px serif";
                ctx1.fillText(text, 30, 80);
        }
}
}
</script>
```

```
</head>
<body>
 <canvas id="tutorial" width="150" height="150" style="border:1px solid grey"></canvas>
 <canvas id="tutorial1" width="150" height="150" style="border:1px solid grey"></canvas>

</body>
</html>
```

Ejemplo3: Implementa el siguiente ejemplo que permite dibujar en un canvas

```
<html>
<head>
  <title>Canvas segundo ejemplo</title>

<script>
//Recibe un identificador del elemento canvas y carga el canvas
//Devueve el contexto del canvas o FALSE si no ha podido conseguise
function cargaContextoCanvas(idCanvas){
  var elemento = document.getElementById(idCanvas);
  if(elemento && elemento.getContext){
    var contexto = elemento.getContext('2d');
    if(contexto){
      return contexto;
    }
  }
  return FALSE;
}

window.onload = function(){
  //Recibimos el elemento canvas
  var contexto = cargaContextoCanvas('micanvas');
  if(contexto){
//Si tengo el contexto
//cambio el color de dibujo a azul
contexto.fillStyle = '#6666ff';
//dibujo un rectángulo azul
contexto.fillRect(10,10,50,50);
//cambio el color a amarillo con un poco de transparencia
contexto.fillStyle = 'rgba(255,255,0,0.7)';
//pinto un rectángulo amarillo semitransparente
contexto.fillRect(35,35,50,50);
  }
}
</script>
</head>
<body>
<canvas id="micanvas" width="100" height="100">
        Tu navegador no soporta canvas.
</canvas>
</body>
```

IES de Teis
Avda. de Galicia, 101
36216 – Vigo

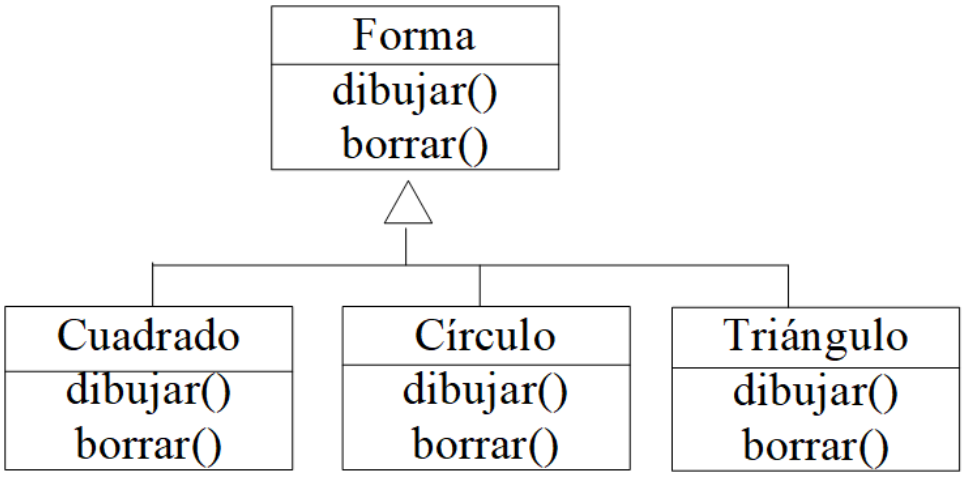
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

Ejemplo4: Implementa el siguiente ejemplo que permite animación con canvas

```
<html>
<head>
<title>Ejercicio Movimiento canvas</title>
<script type="text/javascript">
window.onload = function()
{
 var canvas = document.getElementById('ejercicio');
 var pt = canvas.getContext('2d');
 pt.fillStyle = "rgb(200,0,0)"; // formato del dibujo
 var inc = 1; // velocidad del movimiento
 // objeto que movemos...
 var o = { x: 10, y:10, width:10, height:10 };
 // ejecución periódica
 setInterval(function() {
 // si el objeto sobrepasara los límites del canvas,
 // cambiamos la orientación de este
            if(o.x >= 140 || o.x <=0)
                        inc = -inc
                        o.x += inc
                        limpia(pt)
                        dibuja( o, pt );
            }, 1);
}
dibuja = function(obj, pt)
{
// dibuja el objeto
pt.fillRect (obj.x, obj.y, obj.width, obj.height);
}
limpia = function(pt)
{
// limpia la pantalla y la prepara para el siguiente paso
 pt.clearRect(0,0,150,150)
}
</script>
</head>
<body>
<canvas width="150" height="150" style="border:1px solid grey" id="ejercicio"> </canvas>
</body>
</html>
```

1. **Ejercicio1: Implementa el siguiente ejercicio utilizando canvas y sus métodos, de tal forma que te permita dibujar un círculo rojo sobre una imagen.**

2. **Ejercicio 3: Debes crear la estructura de clases que aparece a continuación y dar funcionalidad al método dibujar utilizando para ello canvas**



1. **Crear la clase abstracta**

```
class Forma {
    void dibujar() {}
    void borrar(){}
}
class Circulo extends Forma {
    void dibujar(){
        document.write("Dibujo un círculo");
    }
    void borrar() {
        document.write("Borro un círculo");
    }
}
class Cuadrado extends Forma {
    void dibujar(){
        document.write ("Dibujo un cuadrado");
    }
    void borrar() {
        document.write("Borro un cuadrado");
    }
}
class Triangulo extends Forma {
    void dibujar(){
        document.write("Dibujo un triángulo");
    }
    void borrar() {
        document.write("Borro un triángulo");
    }
}
```

2. **Crear la clase punto con la siguiente estructura. Todas las figuras geométricas necesitan un punto para dibujarse.**

```
class Punto {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }

    static distancia(a, b) {
        const dx = a.x - b.x;
        const dy = a.y - b.y;

        return Math.sqrt(dx * dx + dy * dy);
    }
}
```

3. **Implementar el método dibujar para cada una de las figuras citadas con anterioridad. Para ello debes utilizar la clase canvas, de tal forma que en vez de mostrar el texto "Dibujo un cuadrado", se cambie por el siguiente código, el cual dibuja la figura en un canvas.**

```
ctx.fillRect(x, y, lado, lado);
```

Ejemplo de ejecución: