Especialidad: Técnico en Programación

Módulo 3. Submódulo2. Desarrolla aplicaciones que se ejecutan en el cliente.

Prof: Hilda Lucía Rodríguez Gómez

Competencia Profesional: Usa JavaScript para manejar eventos

CANVAS EN HTML5

La API Canvas es probablemente la más compleja y extensa de todas las APIs incluidas dentro de la especificación HTML5. Provee varios métodos y propiedades para crear aplicaciones gráficas sobre el elemento **<canvas>**.

API (Application Programming Interface). Interfaz de Programación de Aplicaciones: código que indica a las aplicaciones cómo pueden mantener una comunicación entre sí.

Canvas nos permite dibujar, presentar gráficos en pantalla, animar y procesar imágenes y texto, y trabaja junto con el resto de la especificación para crear aplicacionesVcompletas e incluso video juegos en 2 y 3 dimensiones para la web.

El elemento <canvas> genera un espacio rectangular vacío en la página web (lienzo) en el cual serán mostrados los resultados de ejecutar los métodos provistos por la API. Cuando es creado, produce sólo un espacio en blanco, como un elemento ** ** vacío, pero con un propósito totalmente diferente. Es solo a través de Javascript y los nuevos métodos y propiedades introducidos por la API que esta superficie se transforma en algo práctico.

Sintaxis básica del elemento <canvas>

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Canvas API</title>
<script src="canvas.js"></script>
</head>
<body>
<section id="cajalienzo">
<canvas id="lienzo" width="500" height="300">
      elemento canvas
</canvas>
</section>
</body>
</html>
```

Métodos. Estos métodos son específicos de la API Canvas:

getContext(contexto)	crea el contexto para el lienzo. Puede tomar dos valores: **2d** y **3d** para gráficos en 2 y 3 dimensiones.
fillRect(x, y, ancho, alto)	dibujará un rectángulo sólido directamente en el lienzo en la posición indicada por **x,y** y el tamaño **ancho,alto**.
strokeRect(x, y, ancho, alto)	dibujará un rectángulo vacío (solo el contorno) directamente en el lienzo en la posición indicada por **x,y** y el tamaño **ancho,alto**.
clearRect(x, y, ancho, alto)	borra un área en el lienzo usando una figura rectangular declarada por los valores de sus atributos.
createLinearGradient(x1, y1, x2, y2)	crea un gradiente lineal para asignarlo a una figura como si fuese un color usando la propiedad **fillStyle**. Sus atributos solo especifican las posiciones de comienzo y final del gradiente (relativas al lienzo). Para declarar los colores involucrados en el gradiente, este método debe ser usado en combinación con **addColorStop()**.

createRadialGradient(x1, y1, r1, x2, y2, r2)	crea un gradiente radial para asignarlo a una figura como si fuese un color usando la propiedad **fillStyle**. El gradiente es construido por medio de dos círculos. Los atributos solo especifican la posición y radio de los círculos (relativos al lienzo). Para declarar los colores involucrados en el gradiente, este método debe ser usado en combinación con **addColorStop()**.
addColorStop(posición, color)	es usado para declarar los colores para el gradiente. El atributo **posición** es un valor entre 0.0 y 1.0, usado para determinar dónde el color comenzará la degradación.
beginPath()	es requerido para comenzar un nuevo trazado.
closePath()	puede ser usado al final de un trazado para cerrarlo. Generará una línea recta desde la última posición del lápiz hasta el punto donde el trazado comenzó. No es necesario usar este método cuando el trazado debe permanecer abierto o es dibujado en el lienzo usando **fill()**.
stroke()	es usado para dibujar un trazado como una figura vacía (solo el contorno).
fill()	es usado para dibujar un trazado como una figura sólida.
clip()	es usado para crear una máscara a partir de un trazado. Todo lo que sea enviado al lienzo luego de que este método es declarado será dibujado sólo si cae dentro de la máscara.
moveTo(x, y)	mueve el lápiz virtual a una nueva posición para continuar el trazado desde ese punto.
lineTo(x, y)	Este método agrega líneas rectas al trazado desde la posición actual del lápiz hasta el punto indicado por los atributos **x** e **y**.
rect(x, y, ancho, alto)	agrega un rectángulo al trazado en la posición **x,y** y con un tamaño determinado por **ancho,alto**.
arc(x, y, radio, ángulo inicio, ángulo final, dirección)	agrega un arco al trazado. El centro del arco es determinado por **x** e **y**, los ángulos son definidos en radianes, y la **dirección** es un valor booleano para determinar si el arco será dibujado en el mismo sentido o el opuesto a las agujas del reloj. Para convertir grados en radianes, use la fórmula: **Math.PI/180×grados**.
quadraticCurveTo(cpx, cpy, x, y)	agrega una curva Bézier cuadrática al trazado. Comienza desde la posición actual del lápiz y termina en el punto **x,y**. Los atributos **cpx** y **cpy** especifican la posición del punto de control que dará forma a la curva.
bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)	agrega una curva Bézier cúbica al trazado. Comienza desde la posición actual del lápiz y termina en el punto **x,y**. Los atributos **cp1x**, **cp1y**, **cp2x**, y **cp2y** especifican la posición de los dos puntos de control que darán forma a la curva.
strokeText(texto, x, y, máximo)	dibuja un texto vacío (solo el contorno) directamente en el lienzo. El atributo **máximo** es opcional y determina el máximo tamaño del texto en pixeles.
fillText(texto, x, y, máximo)	dibuja un texto sólido directamente en el lienzo. El atributo **máximo** es opcional y determina el máximo tamaño del texto en pixeles.
measureText(texto)	calcula el tamaño del área que un texto ocupará en el lienzo usando los estilos vigentes. La propiedad **width** es usada para retornar el valor.
translate(x, y)	mueve el origen del lienzo al punto **x,y**. La posición inicial del origen (0,0) es la esquina superior izquierda del área generada por el elemento **<canvas>**.
rotate(angle)	es usado para rotar el lienzo alrededor del origen. El ángulo debe ser declarado en radianes. Para convertir grados en radianes, use la fórmula: **Math.PI/180×grados**.
scale(x, y)	cambia la escala del lienzo. Los valores por defecto son (1.0, 1.0). Los valores provistos pueden ser negativos.
transform(m1, m2, m3, m4, dx, dy)	modifica la matriz de transformación del lienzo. La nueva matriz es calculada sobre la anterior.
setTransform(m1, m2, m3, m4, dx, dy)	modifica la matriz de transformación del lienzo. Reinicia los valores anteriores y declara los nuevos.
save()	graba el estado del lienzo, incluyendo la matriz de transformación, propiedades de estilo y la máscara.
restore()	restaura el último estado del lienzo grabado, incluyendo la matriz de transformación, propiedades de estilo y la máscara.
drawImage()	dibujará una imagen en el lienzo. Existen tres sintaxis posibles. La sintaxis **drawImage(imagen,x,y)** dibuja la imagen en la posición **x,y**. La sintaxis **drawImage(imagen,x,y,ancho,alto)** dibuja la imagen en la posición **x,y** con un nuevo tamaño declarado por **ancho,alto**. Y la sintaxis **drawImage(imagen, x1, y1, ancho1, alto1, x2, y2, ancho2, alto2)** toma una porción de la imagen original determinada por **x1,y1,ancho1,alto1** y la dibuja en el lienzo en la posición **x2,y2** y el nuevo tamaño **ancho2,alto2**.

getImageData(x, y, ancho, alto)	toma una porción del lienzo y la graba como datos en un objeto. Los valores del objeto son accesibles a través de las propiedades **width**, **height** y **data**. Las primeras dos propiedades retornan el tamaño de la porción de la imagen tomada, y **data** retorna la información como un array con valores representando los colores de cada pixel. Este valor puede ser accedido usando la fórmula **(ancho×4×y)+(x×4)**.
putImageData(datosImagen, x, y)	dibuja en el lienzo la imagen representada por la información en **datosImagen**.
createImageData(ancho, alto)	crea una nueva imagen en formato de datos. Todos los pixeles son inicializados en color negro transparente. Puede tomar datos de imagen como atributo en lugar de **ancho** y **alto**. En este caso la nueva imagen tendrá el tamaño determinado por los datos provistos.
createPattern(imagen, tipo)	crea un patrón desde una imagen que luego podrá ser asignado a una figura usando la propiedad **fillStyle**. Los valores posibles para el atributo **tipo** son **repeat**, **repeat-x**, **repeat-y** y **no-repeat**.

Propiedades. La siguiente lista de propiedades es específica para la API Canvas:

strokeStyle	declara el color para las líneas de las figuras. Puede recibir cualquier valor CSS, incluidas funciones como **rgb()** y **rgba()**.
fillStyle	declara el color para el interior de figuras sólidas. Puede recibir cualquier valor CSS, incluidas funciones como **rgb()** y **rgba()**. Es también usada para asignar gradientes y patrones a figuras (estos estilos son primero asignados a una variable y luego esa variable es declarada como el valor de esta propiedad).
globalAlpha	es usada para determinar el nivel de transparencia de las figuras. Recibe valores entre 0.0 (completamente opaco) y 1.0 (completamente transparente).
lineWidth	especifica el grosor de la línea. Por defecto el valor es 1.0.
lineCap -	determina la forma de la terminación de las líneas. Se pueden utilizar tres valores: **butt** (terminación normal), **round** (termina la línea con un semicírculo) y **square** (termina la línea con un cuadrado).
lineJoin	determina la forma de la conexión entre líneas. Se pueden utilizar tres valores: **round** (la unión es redondeada), **bevel** (la unión es cortada) y **miter** (la unión es extendida hasta que ambas líneas alcanzan un punto en común).
miterLimit	determina cuánto se extenderán las líneas cuando la propiedad **lineJoin** es declarada como **miter**.
font	es similar a la propiedad **font** de CSS y utiliza la misma sintaxis para declarar los estilos del texto.
textAlign	determina cómo el texto será alineado. Los posibles valores son **start**, **end**, **left**, **right** y **center**.
textBaseline	determina el alineamiento vertical para el texto. Los posibles valores son: **top**, **hanging**, **middle**, **alphabetic**, **ideographic** y **bottom**.
shadowColor	establece el color para la sombra. Utiliza valores CSS.
shadowOffsetX	declara la distancia horizontal entre la sombra y el objeto.
shadowOffsetY	declara la distancia vertical entre la sombra y el objeto.
shadowBlur	recibe un valor numérico para generar un efecto de difuminación para la sombra.
globalCompositeOperation	determina cómo las nuevas figuras serán dibujadas en el lienzo considerando las figuras ya existentes. Puede recibir varios valores: **source-over**, **source-in**, **source-out**, **source-atop**, **lighter**, **xor**, **destination-over**, **destination-in**, **destination-out**, **destination-atop**, **darker** y **copy**. El valor por defecto es **source-over**, lo que significa que las nuevas formas son dibujadas sobre las anteriores.