Materia: Web Security.
Tema: Texto a **`canvas`**.

1. Introducción

- El atributo `font` especifica el estilo, tamaño y fuente del texto.
- El atributo `textBaseLine` especifica los puntos de alineación del texto. Hay seis valores de atributos diferentes: `top`, `hanging`, `middle`, `alphabetic`, `ideographic` y `bottom`.
- El método `fillText` dibuja el texto en el `canvas`. Este método recibe tres argumentos. El primero es el texto que se va a dibujar en el `canvas`. Los argumentos segundo y tercero son las coordenadas `x` y `y`. Puede incluir el cuarto argumento opcional `anchuraMaxima` para limitar la anchura del texto.
- El atributo `textAlign` especifica la alineación horizontal del texto relativa a la coordenada `x` del texto. Hay cinco posibles valores para el atributo `textAlign`: `left`, `right`, `center`, `start` (el predeterminado) y `end`.
- El atributo `lineWidth` especifica el grosor del trazo que se utiliza para dibujar el texto.
- El `strokeStyle` especifica el color del texto.
- Si usa `strokeText` en vez de `fillText` se dibuja texto en contorno en vez de relleno.

Valor	Descripción
`top`	Parte superior del cuadro `em`
`hanging`	Línea base colgante
`middle`	Parte media del cuadro `em`
`alphabetic`	Línea base alfabética (el valor predeterminado)
`ideographic`	Línea base ideográfica
`bottom`	Parte inferior del cuadro `em`

Valores de `textBaseline`.

Valor	Descripción
`left`	El texto está alineado a la izquierda.
`right`	El texto está alineado a la derecha.
`center`	El texto está centrado.
`start (el valor predeterminado)`	EJ texto está alineado a b izquierda si el inicio de la línea es de izquierda a derecha; el texto se alinea a la derecha si el inicio del texto es de derecha a izquierda.
`end`	El texto se alinea a la derecha si el final de la línea es de izquierda a derecha; se alinea a la izquierda si el final de la línea es de derecha a izquierda.

Valores de `textAlign`.

EJEMPLO 14.
Archivo texto.html:

```
<!-- Dibujar texto en un canvas. -->
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>Texto</title>
	</head>
	<body>
		<canvas id = "texto" width = "230" height = "100"
			style = "border: 1px solid black;">
		</canvas>
		<script>
			var canvas = document.getElementById("texto");
			var contexto = canvas.getContext("2d");
			// dibujar la primera linea de texto
			contexto.fillStyle = "red";
```

```
                contexto.font = "italic 24px serif";
                contexto.textBaseline = "top";
                contexto.fillText("Canvas HTML5", 0, 0);
                // dibujar la segunda linea de texto
                contexto.font = "bold 30px sans-serif";
                contexto.textAlign = "center";
                contexto.lineWidth = 2;
                contexto.strokeStyle = "navy";
                contexto.strokeText("Canvas HTML5", 115, 50);
          </script>
    </body>
</html>
```

2. Ajuste del tamaño del elemento canvas para llenar la ventana del navegador

- Use una hoja de estilos CSS para establecer el valor de position del `canvas` en `absolute` y establecer su anchura (`width`) y su altura (`height`) al 100%, en vez de usar coordenadas fijas.
- Use la función `draw` de JavaScript para dibujar el `canvas` cuando se despliegue la aplicación.
- Use el método `fillRect` para dibujar el color en el `canvas`. Las coordenadas `x` y `y` son `0,0`: la esquina superior izquierda del `canvas`. El valor `x1` es `contexto.canvas.width` y el valor `y1` es `contexto.value.height`, por lo que sin importar el tamaño de la ventana, el valor `x1` siempre será la anchura del `canvas` y el valor `y1` la altura del `canvas`.

EJEMPLO 15.
Archivo llenarventana.html:

```
<!-- Ajuste del tamaño de un canvas para llenar la ventana. -->
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8">
        <title>Llenar la ventana</title>
        <style type = "text/css">
            canvas { position: absolute; left: 0px; top: 0px;
                width: 100%; height: 100%; }
        </style>
    </head>
    <body>
        <canvas id = "ajustam"></canvas>
        <script>
            function dibujar(){
                var canvas = document.getElementById("ajustam");
                var contexto = canvas.getContext("2d");
                contexto.fillStyle = "yellow";
                contexto.fillRect(
                    0, 0, contexto.canvas.width, contexto.canvas.height );
            } // fin de la función dibujar
            window.addEventListener( "load", dibujar, false );
        </script>
    </body>
</html>
```

Canvas amarillo llena la ventana del navegador.

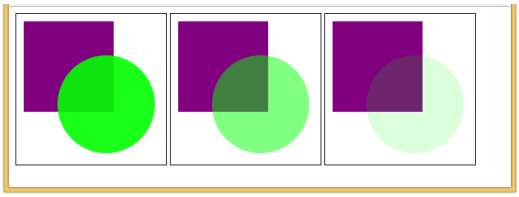
3. Transparencia alfa

- El valor del atributo `globalAlpha` puede ser cualquier número entre `0` (transparencia total) y `1` (el valor predeterminado, que es opacidad total).

EJEMPLO 16.
Archivo alfaa.html:

```
<!-- Uso del atributo globalAlpha en un canvas. -->
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>Transparencia alfa</title>
	</head>
	<body>
		<!-- valor alfa 0.7S -->
		<canvas id = "alfa" width = "200" height = "200"
			style = "border: 1px solid black;">
		</canvas>
		<script>
			var canvas = document.getElementById("alfa");
			var contexto = canvas.getContext("2d");
			contexto.beginPath();
			contexto.rect( 10, 10, 120, 120);
			contexto.fillStyle = "purple";
			contexto.fill();
			contexto.globalAlpha = 0.9;
			contexto.beginPath();
			contexto.arc(120, 120, 65, 0, 2 * Math.PI, false );
			contexto.fillStyle = "lime";
			contexto.fill();
		</script>
		<!-- valor alfa0.5 -->
		<canvas id = "alfa2" width = "200" height = "200"
			style = "border: 1px solid black;">
		</canvas>
		<script>
			var canvas = document.getElementById("alfa2");
			var contexto = canvas.getContext("2d");
			contexto.beginPath();
			contexto.rect(10, 10, 120, 120);
			contexto.fillStyle = "purple";
			contexto.fill();
			contexto.globalAlpha = 0.5;
			contexto.beginPath();
			contexto.arc( 120, 120, 65, 0, 2 * Math.PI, false );
			contexto.fillStyle = "lime";
			contexto.fill();
		</script>
		<!-- 0.15 alpha value -->
		<canvas id = "alfa3" width = "200" height = "200"
			style = "border: 1px solid black;">
```

```
            </canvas>
            <script>
                  var canvas = document.getElementById("alfa3");
                  var contexto = canvas.getContext("2d");
                  contexto.beginPath();
                  contexto.rect(10, 10, 120, 120);
                  contexto.fillStyle = "purple";
                  contexto.fill();
                  contexto.globalAlpha = 0.15;
                  contexto.beginPath();
                  contexto.arc(120, 120, 65, 0, 2 * Math.PI, false );
                  contexto.fillStyle = "lime";
                  contexto.fill();
            </script>
      </body>
</html>
```



globalAlpha=0.9 globalAlpha=0.5 globalAlpha=0.15

4. Composición

- La composición nos permite controlar la distribución de las figuras e imágenes en capas en un `canvas` mediante el uso de dos atributos: el atributo `globalAlpha` y el atributo `globalCompositeOperation`.
- Hay 11 valores para el atributo `globalCompositeOperation`. El origen es la imagen que se va a dibujar en el `canvas`. El destino es el mapa de bits actual en el `canvas`.
- Si usa `source-in`, se muestra la imagen de origen cuando se traslapan las imágenes y ambas son opacas. Ambas son transparentes cuando no hay traslape.
- Si usa `source-out`, si la imagen de origen es opaca y el de destino es transparente, se muestra la de origen en donde se traslapan las imágenes. Ambas son transparentes en donde no hay traslape.
- `source-over` (el valor predeterminado) coloca la imagen de origen sobre la de destino. La imagen de origen se muestra cuando es opaca y se traslapan. La imagen de destino se muestra cuando no hay traslape.
- `destination-atop` coloca la imagen de destino sobre la imagen de origen. Si ambas son opacas, se muestra la de destino en donde se traslapan. Si la imagen de destino es transparente pero la de origen es opaca, se muestra la imagen de origen cuando se traslapan. La imagen de origen es transparente en donde no haya traslape.
- `destination-in` muestra la imagen de destino en donde se traslapan las imágenes y ambas son opacas. Ambas imágenes son transparentes cuando no hay traslape.
- Al usar `destination-out`, si la imagen de destino es opaca y la imagen de origen es transparente, se muestra la de destino en donde se traslapan. Ambas son transparentes cuando no hay traslape.
- `destination-over` coloca la imagen de destino sobre la imagen de origen. La imagen de destino se muestra en donde sea opaca y se traslapan. Se muestra la imagen de origen en donde no hay traslape.
- `lighter` muestra la suma del color de la imagen de origen y el color de la imagen de destino (hasta el valor de color RGB máximo de 255) en donde se traslapan las imágenes. Ambas son normales en cualquier otra parte.
- Al usar `copy`, si las imágenes se traslapan, sólo se muestra la imagen de origen (se ignora el destino).
- Con `xor`, bs imágenes son transparentes en donde se traslapan y normales en cualquier otra parte.

Valor	Descripción
`source-atop`	La imagen de origen se coloca sobre la imagen de destino. Si ambas son opacas, se muestra el origen en donde se traslapan. Si el origen es transparente pero la imagen de destino es opaca, se muestra la

	imagen de destino en donde se traslapan. La imagen de destino es transparente en donde no haya traslape.
source-in	Se muestra la imagen de origen cuando se traslapan las imágenes y ambas son opacas. Ambas son transparentes cuando no hay traslape.
source-out	Si la imagen de origen es opaca y la de destino es transparente, se muestra la de origen en donde se traslapan las imágenes. Ambas son transparentes en donde no hay traslape.
source-over (predeterminado)	La imagen de origen se coloca sobre la imagen de destino. La de origen se muestra cuando es opaca y se traslapan las imágenes. La de destino se muestra cuando no hay traslape.
destination-atop	La imagen de destino se coloca sobre la imagen de origen. Si ambas son opacas, se muestra la imagen de destino en donde se traslapan. Si la imagen de destino es transparente pero la de origen es opaca, se muestra la imagen de origen cuando se traslapen las imágenes. La de origen es transparente en donde no haya traslape.
destination-in	Se muestra la imagen de destino en donde se traslapan las imágenes y ambas son opacas. Ambas son transparentes cuando no hay traslape.
destination-out	Si la imagen de destino es opaca y la imagen de origen es transparente, se muestra la de destino en donde se traslapan. Ambas son transparentes cuando no hay traslape.
destination-over	Se coloca la imagen de destino sobre la imagen de origen. La de destino se muestra en donde sea opaca y se traslapan las imágenes. Se muestra la de origen en donde no hay traslape.
lighter	Muestra la suma del color de la imagen de origen y el color de la imagen de destino (hasta el valor de color RGB máximo de 255) en donde se traslapan. Ambas son normales en cualquier otra parte.
copy	Si las imágenes se traslapan, sólo se muestra la de origen (se ignora el destino).
xor	Imagen de origen `xor` (or exclusivo) destino. Las imágenes son transparentes en donde se traslapan y normales en cualquier otra parte.

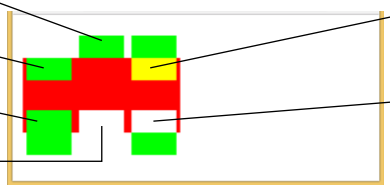
EJEMPLO 17.

Archivo composición.html:

```
<!-- Composición en un canvas. -->
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>ComposiciAoacute;n</title>
	</head>
	<body>
		<canvas id = "compuesto" width = "220" height = "200">
		</canvas>
		<script>
		function dibujar(){
			var canvas = document.getElementById("compuesto");
			var contexto = canvas.getContext("2d" );
			contexto.fillStyle = "red";
			contexto.fillRect(5, 50, 210, 100);
			// source-atop
			contexto.globalCompositeOperation = "source-atop";
			contexto.fillStyle = "lime";
			contexto.fillRect(10, 20, 60, 60);
			// source-over
			contexto.globalCompositeOperation = "source-over";
			contexto.fillStyle = "lime";
			contexto.fillRect(10, 120, 60, 60);
			// destination-over
			contexto.globalCompositeOperation = "destination-over";
			contexto.fillStyle = "lime";
			contexto.fillRect(80, 20, 60, 60);
			// destination-out
			contexto.globalCompositeOperation = "destination-out";
```

```
                contexto.fillStyle = "lime";
                contexto.fillRect(80, 120, 60, 60);
                // lighter
                contexto.globalCompositeOperation = "lighter";
                contexto.fillStyle = "lime";
                contexto.fillRect(150, 20, 60, 60);
                // xor
                contexto.globalCompositeOperation = "xor";
                contexto.fillStyle = "lime";
                contexto.fillRect(150, 120, 60, 60);
          } // fin de la función dibujar
          window.addEventListener( "load", dibujar, false );
          </script>
    </body>
</html>
```

Rojo, `destination-over`, traslape de imágenes y lima en donde no hay traslape.
Lima, `source-atop`, traslape de figuras y transparencia en cualquier otra parte.
Lima, `source-over`, traslape de figuras y en donde no hay traslape.
`destination-out`, transparencia en traslape de figuras y en donde no hay traslape.



Amarillo, `lighter`, traslape (suma de rojo y lima). Ambas son normales en cualquier otra parte.
`xor`, transparencia en traslape de imágenes. Ambas son normales en cualquier otra parte.

5. Juego del cañón

- El elemento audio de HTML5 puede contener varios elementos `source` para el archivo de audio en varios formatos, de modo que se pueda soportar la reproducción de los sonidos entre varios navegadores.
- Puede crear sus propias propiedades en objetos de JavaScript con sólo asignar un valor al nombre de una propiedad.
- La detección de colisiones determina si la bala de cañón chocó con alguno de los bordes del `canvas`, con el bloqueador o con una sección del objetivo. Los marcos de trabajo de desarrollo de juegos por lo general proveen herramientas de detección de colisiones integradas más sofisticadas.

6. Métodos `save` y `restore`

- El estado del `canvas` incluye su estilo actual y transformaciones, que se mantienen en una pila.
- El método `save` se usa para guardar el estado actual del contexto.
- El método `restore` restaura el contenido a su estado anterior.

EJEMPLO 18.
Archivo saveyrestore.html:

```
<!-- Guardar el estado actual y restaurar el estado anterior. -->
<!DOCTYPE html>
<html>
      <head>
            <meta charset = "utf-8">
            <title>Save y restore</title>
      </head>
      <body>
            <canvas id = "save" width = "400" height = "200">
            </canvas>
            <script>
                  function dibujar(){
                        var canvas = document.getElementById("save");
                        var contexto = canvas.getContext("2d");
                        // dibujar rectángulo y guardar la configuración
```

```
                        contexto.fillStyle = "red"
                        contexto.fillRect(0, 0, 400, 200);
                        contexto.save();
                        // modificar la configuración y guardar de nuevo
                        contexto.fillStyle = "orange"
                        contexto.fillRect(0, 40, 400, 160);
                        contexto.save();
                        // modificar la configuración de nuevo
                        contexto.fillStyle = "yellow"
                        contexto.fillRect(0, 80, 400, 120);
                        // restaurar a la configuración anterior y dibujar nuevo rectángulo
                        contexto.restore();
                        contexto.fillRect(0, 120, 400, 80);
                        // restaurar a la configuración original y dibujar nuevo rectángulo
                        contexto.restore();
                        contexto.fillRect(0, 160, 400, 40);
                    }
                    window.addEventListener( "load", dibujar, false );
                </script>
            </body>
</html>
```

El primer rectángulo es rojo.
El segundo rectángulo es naranja.
El tercer rectángulo es amarillo.
El cuarto rectángulo se restaura a naranja.
El quinto rectángulo se restaura a rojo.

7. Una nota sobre SVG

- Los gráficos vectoriales están formados de primitivas geométricas escalables, como segmentos de línea y arcos.
- SVG (Gráficos de vector escalables) está basado en XML, por lo que usa una metodología declarativa: usted le dice qué quiere y SVG lo crea para usted. El elemento `canvas` de HTML5 está basado en JavaScript, por lo que usa una metodología imperativa: usted le dice cómo crear sus gráficos mediante la programación en JavaScript.
- Con SVG, cada parte separada de su gráfico se convierte en un objeto que puede manipularse a través del DOM.
- SVG es más conveniente para gráficas multiplataforma, lo cual se está volviendo muy importante con la proliferación de los "factores de forma", como equipos de escritorio, notebooks, smartphones, tabletas y diversos dispositivos de propósito especial como los sistemas de navegación de automóviles.