

HTML



HTML5: Canvas Un lienzo en la Web



Miguel Angel Cumpa Ascuña



HTML



Objetivo



3D, Graphics
& Effects



Objetivo de la sesión

- Presentar y describir las características de canvas
 - Citar los antecedentes que derivaron su creación
 - El efecto que tiene sobre otros estándares
 - Demostrar su uso a través de código para una aplicación

HTML



Antecedentes



3D, Graphics
& Effects



Imágenes / Canvas

- Para mostrar gráficos en la web utilizamos imágenes en formatos GIF, PNG, JPG
- Canvas no sólo nos permite generar gráficos sino podemos utilizar imágenes preexistentes



SVG / Canvas

- SVG es un estándar, mientras que Canvas es una nueva etiqueta HTML
- SVG está basado en XML mientras que Canvas es creado mediante `<canvas></canvas>` y su contenido es completado con Javascript

HTML



 3D, Graphics
& Effects

Canvas

Canvas

- Etiqueta o elemento en HTML5 que permite la generación de gráficos en forma dinámica por medio de programación dentro de una página.
 - Posee dos atributos width (ancho) y height (alto), el tamaño por defecto es 150.
 - Permite generar gráficos 2D, juegos, animaciones y composición de imágenes
 - SVG es otra etiqueta que cumple con funciones similares

HTML



1, 2, 3, Canvas



3D, Graphics
& Effects

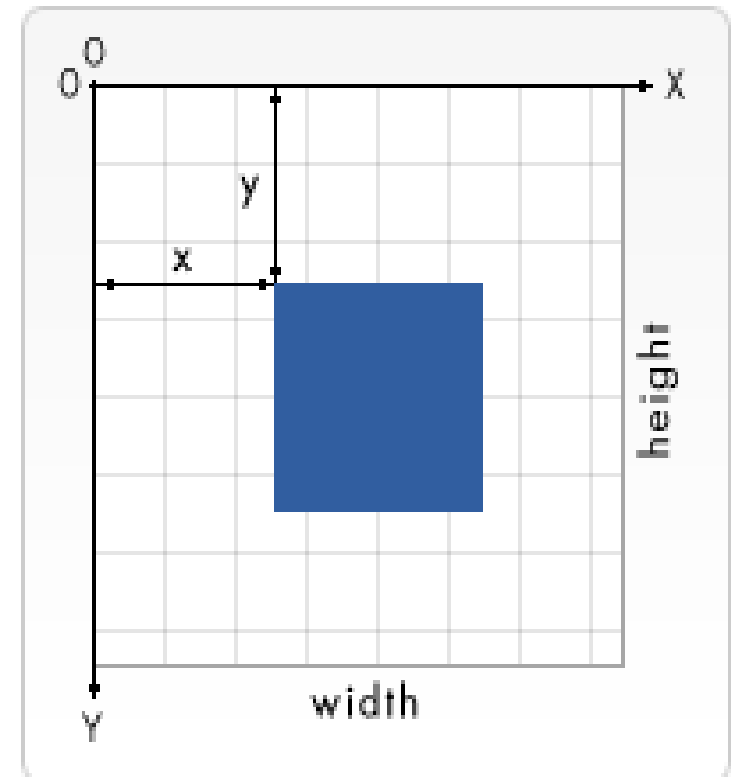


1, 2, 3, Canvas

- Antes de que podamos empezar a dibujar, tenemos que hablar de la *grilla de lienzo o espacio de coordenadas*
- Normalmente una unidad corresponde a 1px en canvas
- El origen de la grilla esta en la esquina superior izquierda (coordenada(0,0))

1, 2, 3, Canvas

- **Nota:** Podemos cambiar el origen a una posición diferente, gire la red e incluso la escala.





1, 2, 3, Canvas

Definir la etiqueta

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



1, 2, 3, Canvas

Verificar si el navegador soporta canvas

- ```
var canvas =
document.getElementById('tutorial');
if (canvas.getContext){
var ctx = canvas.getContext('2d');
 // dibuja con tú código aquí
} else {
 alert("Su navegador no soporta canvas :O");
}
```

# HTML



3D, Graphics  
& Effects

# Formas Básicas



# Formas Básicas

## Definir relleno

- **fillStyle** = "rgb(200,0,0)";  
Define el color de relleno
- **var ctx = canvas.getContext("2d");**  
**ctx.fillStyle = "rgb(200,0,0)";**



# Formas Básicas

## Dibujar formas básicas

- **fillRect(x,y,width,height)** : Dibuja un rectángulo con relleno
- **strokeRect(x,y,width,height)** : Dibuja un contorno rectangular
- **clearRect(x,y,width,height)** : Borra el área especificada y hace que sea totalmente transparente



# Formas Básicas

## Ejemplo

- ```
function draw(){  
    var canvas = document.getElementById('tutorial');  
    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);  
    }  
}
```

Dibujar líneas

- Internamente, las rutas se almacenan como una lista de sub-rutas (líneas, arcos, etc) que en conjunto forman una forma
- **BeginPath()** : La lista se pone a cero y podemos empezar a dibujar nuevas formas
- **ClosePath()** : Intenta cerrar la forma al trazar una línea recta desde el punto actual al principio. Si la forma ya se ha cerrado o no hay un solo punto en la lista esta función no hace nada.



Formas Básicas

- **Stroke()** : Se utiliza para dibujar una forma con contorno
- **Fill()** : Se utiliza para pintar una forma sólida
- **Nota:** Al llamar al método **Fill()**, las formas abiertas se cerrarán automáticamente y no es necesario utilizar el método **ClosePath()**



Formas Básicas

Coordenadas del pincel

- **MoveTo(x, y)**: Equivale a levantar el pincel de un punto y colocarlo en el siguiente. La función `moveTo` toma dos argumentos, **x** e **y**, que son las coordenadas del nuevo punto de partida.

Lineas

- **lineTo(x, y)** : Es un método que nos permitirá dibujar una línea recta.
Este método toma dos argumentos **x** e **y**, que son las coordenadas del punto final de la línea. El punto de partida depende de los trazados anteriores, donde el punto final de la ruta anterior es el punto de partida para la siguiente, etc. El punto de partida también se puede cambiar mediante el método `moveTo()`.

Arcos

- **arc(x, y, radius, startAngle, endAngle, anticlockwise):**
Dibuja arcos o círculos.
La **x** e **y** son las coordenadas del centro del círculo. **radius** se explica por sí mismo. El **startAngle** y **endAngle** definen los puntos inicial y final del arco en radianes. El ángulo de inicio y cierre se miden desde el eje x. El **anticlockwise** es un valor booleano que cuando es true señala a la izquierda del arco, de lo contrario en el sentido de las agujas del reloj.
- **Nota:** Los ángulos en la función arc se miden en radianes, no en grados. Para convertir grados a radianes se puede utilizar la siguiente expresión JavaScript: `var = radianes (Math.PI/180) * grados.`

Rectángulos

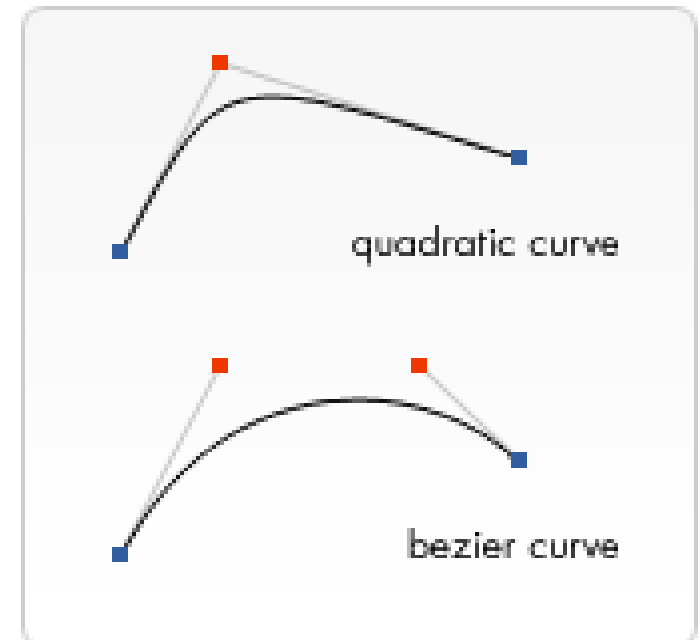
- **rect(x, y, width, height)**: Es un método que agrega un camino rectángulo con la lista de rutas. Es un método más directo para crear rectángulos.
Este método toma cuatro argumentos. Los parámetros **x** e **y** definen las coordenadas de la esquina superior izquierda de la trayectoria rectangular nueva. **width** y **height** definen la anchura y la altura del rectángulo.

Curvas Bézier y Cuadráticas

- `quadraticCurveTo(cp1x, cp1y, x, y)`
- `bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)`
- Los parámetros *x* e *y* en estos dos métodos son las coordenadas del punto final. *cp1x* y *cp1y* son las coordenadas del primer punto de control, y *cp2x* y *cp2y* son las coordenadas del segundo punto de control.

Formas Básicas

- Una curva cuadrática de Bézier tiene un inicio y un punto final (puntos azules) y un solo punto de control (punto rojo), mientras que una curva cúbica de Bézier se utilizan dos puntos de control.





Ejemplo - LineTo

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

Ejemplo - Arc

- ```
ctx.beginPath();
ctx.arc(75,75,50,0,Math.PI*2,true);
ctx.moveTo(110,75);
ctx.arc(75,75,35,0,Math.PI,false);
ctx.moveTo(65,65);
ctx.arc(60,65,5,0,Math.PI*2,true);
ctx.moveTo(95,65);
ctx.arc(90,65,5,0,Math.PI*2,true);
ctx.stroke();
```



# Formas Básicas

## Ejemplo - QuadraticCurveTo

- `ctx.beginPath();`  
`ctx.moveTo(75,25);`  
`ctx.quadraticCurveTo(25,25,25,62.5);`  
`ctx.quadraticCurveTo(25,100,50,100);`  
`ctx.quadraticCurveTo(50,120,30,125);`  
`ctx.quadraticCurveTo(60,120,65,100);`  
`ctx.quadraticCurveTo(125,100,125,62.5);`  
`ctx.quadraticCurveTo(125,25,75,25);`  
`ctx.stroke();`



# Formas Básicas

## Ejemplo - BezierCurveTo

- `ctx.beginPath();`  
`ctx.moveTo(75,40);`  
`ctx.bezierCurveTo(75,37,70,25,50,25);`  
`ctx.bezierCurveTo(20,25,20,62.5,20,62.5);`  
`ctx.bezierCurveTo(20,80,40,102,75,120);`  
`ctx.bezierCurveTo(110,102,130,80,130,62.5);`  
`ctx.bezierCurveTo(130,62.5,130,25,100,25);`  
`ctx.bezierCurveTo(85,25,75,37,75,40);`  
`ctx.fill();`

# HTML



3D, Graphics  
& Effects

# Imágenes



# Imágenes

- Utiliza las imágenes como fondo o como una foto de composición dinámica
- Las imágenes externas se pueden utilizar en cualquier formato PNG, GIF o JPEG

## Importando imágenes

- Referenciar al JavaScript Image object u otro elemento Canvas como recurso. No es posible utilizar las imágenes con sólo proporcionar una URL o ruta de acceso a ellos.
- Dibujamos la imagen usando la función ***drawImage***

## Usando imágenes de la misma página

- ***document.images***: accedemos a todas las imágenes de la pagina
- ***document.getElementsByTagName***: accedemos a las imágenes mediante el **tag**
- ***document.getElementById***: si conocemos el ID de la imagen



## Usando otros elementos Canvas

- Así como usamos a otras imágenes, usando el método `document.getElementsByTagName` ó el método `document.getElementById`
- Una de las prácticas mas útiles es utilizar un canvas como vista previa de uno mas grande

## Image Object

- ```
var img = new Image(); // crea el objeto Image
img.onload = function(){
    // Dibuja con tu codigo aquí
}
img.src = 'myImage.png'; // ruta de la imagen o
recurso
```

Embebiendo una imagen via data: URL

- Podemos incluir imágenes via data : URL
- La imagen se define en una cadena de encoded Base64
- Ventaja: el resultado de la imagen esta disponible sin otra petición al servidor
- Desventajas: las imágenes no son cacheadas y que mientras más pesada sea la imagen más larga será la cadena de definición



Ejemplo

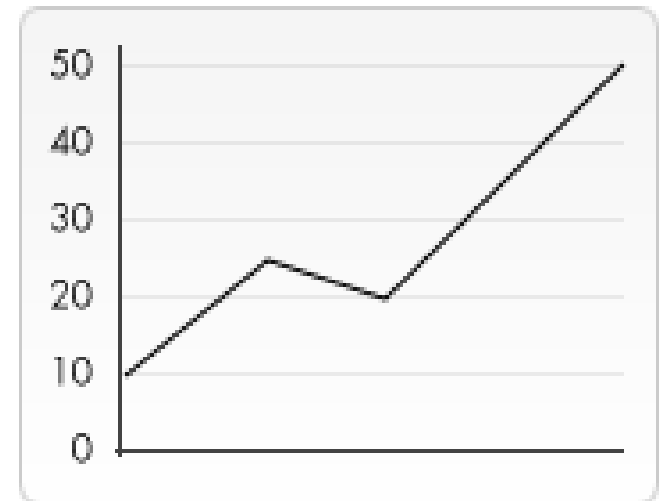
- `var img_src =
'data:image/gif;base64,R0lGODlhCwALAAIAAA
AA3pn/ZiH5BAEAAAEALAAAAAALAAAsAAAIU
hA+hkcuO4ImNVindo7qyrIXiGBYAOW==';`

Dibujar Imágenes

- **drawImage(image, x, y)** : Dibuja el recurso en el canvas
- **Image** es la referencia a nuestra imagen u objeto canvas, **x** e **y** son las coordenadas donde nuestra imagen sera colocada
- El método **drawImage** esta sobrecargado y tiene tres variantes

Ejemplo

- ```
function draw() {
 var ctx =
document.getElementById('canvas').getContext('2d');
 var img = new Image();
 img.onload = function(){
 ctx.drawImage(img,0,0);
 ctx.beginPath();
 ctx.moveTo(30,96);
 ctx.lineTo(70,66);
 ctx.lineTo(103,76);
 ctx.lineTo(170,15);
 ctx.stroke();
 }
 img.src = 'images/backdrop.png';
}
```



## Escalando

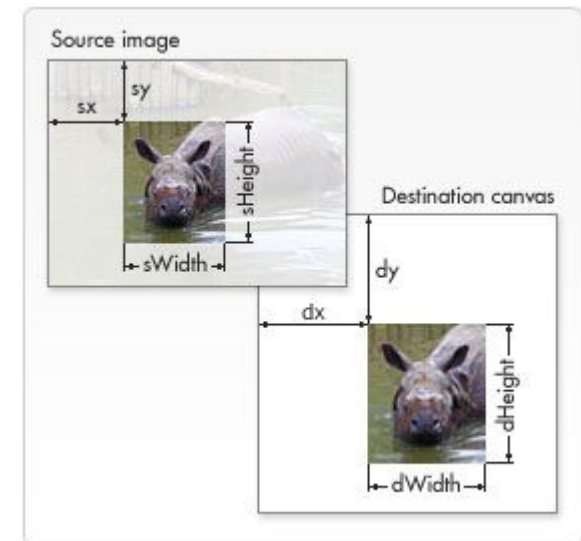
- **drawImage(image, x, y, width, height)**: Escala la imagen de acuerdo al ancho y alto dado
- ***width y height*** son los tamaños de la imagen en el canvas

## Ejemplo

- ```
function draw() {
  var ctx =
document.getElementById('canvas').getContext('2d');
  var img = new Image();
  img.onload = function(){
    for (var i=0;i<4;i++){
      for (var j=0;j<3;j++){
        ctx.drawImage(img,j*50,i*38,50,38);
      }
    }
  }
  img.src = 'images/rhino.jpg';
}
```

Rebanando

- `drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)`
- **image** es o bien una referencia a un objeto de imagen o una referencia a otro canvas. Para los otros ocho parámetros observar la imagen de la derecha. Los primeros cuatro parámetros definen ubicación y tamaño de la porción de la imagen de origen. Los últimos cuatro parámetros definen la posición y tamaño en el lienzo de destino.



Ejemplo

- ```
function draw() {
 var canvas = document.getElementById('canvas');
 var ctx = canvas.getContext('2d');
 // Draw slice
 ctx.drawImage(document.getElementById('source'),
33,71,104,124,21,20,87,104);
 // Draw frame
 ctx.drawImage(document.getElementById('frame'),0,0);
}
```

# HTML



# Color



3D, Graphics  
& Effects



# Color

- **StrokeStyle:** se utiliza para fijar el color del contorno de la forma
- **FillStyle:** es para fijar el color de relleno
- De forma predeterminada, el contorno y color de relleno que se establece es negro (color RGB hexadecimal #000000).

# Color

- Las valores válidos que puede introducir, de acuerdo a la especificación, deben ser valores de color CSS3. Cada uno de los ejemplos siguientes describen el mismo color.
- `// color: naranja`  
`ctx.fillStyle = "orange";`  
`ctx.fillStyle = "#FFA500";`  
`ctx.fillStyle = "rgb(255,165,0)";`  
`ctx.fillStyle = "rgba(255,165,0,1)";`



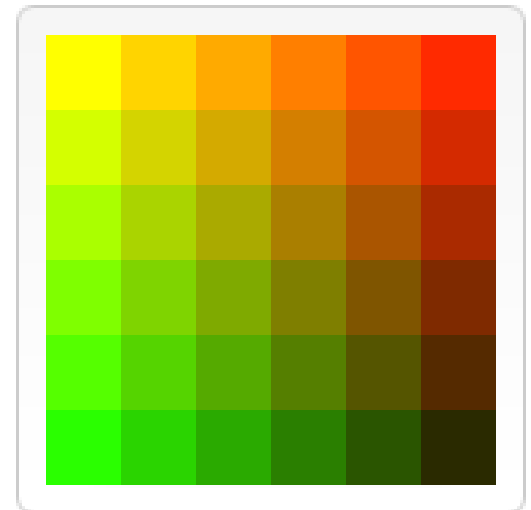
# Color

- **Nota 1:** En la actualidad no todos los CSS 3 valores de color son compatibles con el motor Gecko. Por ejemplo, los valores de color HSL (100%, 25%, 0) o RGB (0, 100%, 0) no están permitidos. Si nos atenemos a los de arriba, no habrá algún problema.

- **Nota 2:** Si establece **strokeStyle** o **fillStyle**, el nuevo valor se convierte en el valor predeterminado para todas las formas. Para cada forma que desee en un color diferente, tendrá que volver a asignar la propiedad **fillStyle** o **strokeStyle**.

## Ejemplo - fillStyle

- ```
function draw() {  
  var ctx = document.getElementById('canvas')  
  .getContext('2d');  
  for (var i=0;i<6;i++){  
    for (var j=0;j<6;j++){  
      ctx.fillStyle = 'rgb(' + Math.floor(255-42.5*i) + ',' +  
        Math.floor(255-42.5*j) + ',0)';  
      ctx.fillRect(j*25,i*25,25,25);  
    }  
  }  
}
```



Ejemplo - `strokeStyle`

- ```
function draw() {
 var ctx = document
.getElementById('canvas').getContext('2d');
 for (var i=0;i<6;i++){
 for (var j=0;j<6;j++){
 ctx.strokeStyle = 'rgb(0,' + Math.floor(255-42.5*i)
+ ',' + Math.floor(255-42.5*j) + ')';
 ctx.beginPath();
 ctx.arc(12.5+j*25,12.5+i*25,10,0,Math.PI*2,true);
 ctx.stroke();
 }
 }
}
```



## Transparencia

- Esto se hace mediante el establecimiento de la propiedad **globalAlpha**, o podemos asignar un color semi-transparente a **StrokeStyle** o **FillStyle**
- **globalAlpha** = transparency value

# Color

- Esta propiedad aplica un valor de transparencia a todas las formas dibujadas en el canvas. El rango válido de valores es de 0.0 (totalmente transparente) a 1.0 (totalmente opaco). De forma predeterminada, esta propiedad se establece en 1.0.
- La propiedad **globalAlpha** puede ser útil si desea dibujar varias formas en el lienzo con la transparencia similar.

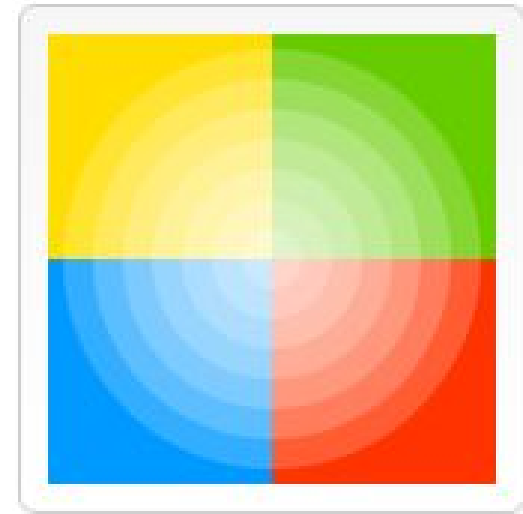
# Color

- El **strokeStyle** y **fillStyle** aceptan valores de color CSS 3, por ello, podemos utilizar, la función **rgba ()** es similar a la **rgb ()** pero que tiene un parámetro adicional que establece el valor de transparencia de este color en particular. El rango válido es de nuevo entre 0,0 (totalmente transparente) y 1.0 (totalmente opaco).
- **ctx.fillStyle = "rgba(255,165,0,0.5)";**



## Ejemplo - globalAlpha

```
• function draw() {
 var ctx = document.getElementById('canvas').getContext('2d');
 // dibujo el fondo
 ctx.fillStyle = '#FD0';
 ctx.fillRect(0,0,75,75);
 ctx.fillStyle = '#6C0';
 ctx.fillRect(75,0,75,75);
 ctx.fillStyle = '#09F';
 ctx.fillRect(0,75,75,75);
 ctx.fillStyle = '#F30';
 ctx.fillRect(75,75,150,150);
 ctx.fillStyle = '#FFF';
 // fijo el valor de la transparencia
 ctx.globalAlpha = 0.2;
 // dibujo circulos semitransparentes
 for (i=0;i<7;i++){
 ctx.beginPath();
 ctx.arc(75,75,10+10*i,0,Math.PI*2,true);
 ctx.fill();
 }
}
```



## Ejemplo - fillRect

- ```
function draw() {  
  var ctx = document.getElementById('canvas').getContext('2d');  
  // Draw background  
  ctx.fillStyle = 'rgb(255,221,0)';  
  ctx.fillRect(0,0,150,37.5);  
  ctx.fillStyle = 'rgb(102,204,0)';  
  ctx.fillRect(0,37.5,150,37.5);  
  ctx.fillStyle = 'rgb(0,153,255)';  
  ctx.fillRect(0,75,150,37.5);  
  ctx.fillStyle = 'rgb(255,51,0)';  
  ctx.fillRect(0,112.5,150,37.5);  
  // Draw semi transparent rectangles  
  for (var i=0;i<10;i++){  
    ctx.fillStyle = 'rgba(255,255,255,'+(i+1)/10+)';  
    for (var j=0;j<4;j++){  
      ctx.fillRect(5+i*14,5+j*37.5,14,27.5)  
    }  
  }  
}
```





Estilos de lineas

- `lineWidth = value`
`lineCap = type`
`lineJoin = type`
`miterLimit = value`

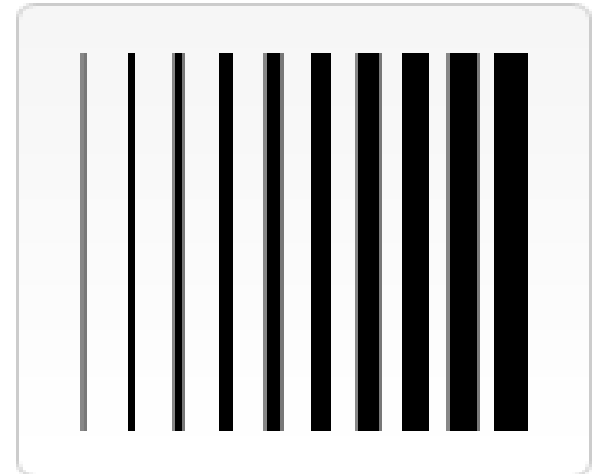


LineWidth

- Esta propiedad establece el grosor de la línea actual. Los valores deben ser números positivos. Por defecto, este valor se establece en 1,0 unidades.

Ejemplo - lineWidth

- ```
function draw() {
 var ctx=
 document.getElementById('canvas').getContext('2d');
 for (var i = 0; i < 10; i++){
 ctx.lineWidth = 1+i;
 ctx.beginPath();
 ctx.moveTo(5+i*14,5);
 ctx.lineTo(5+i*14,140);
 ctx.stroke();
 }
}
```



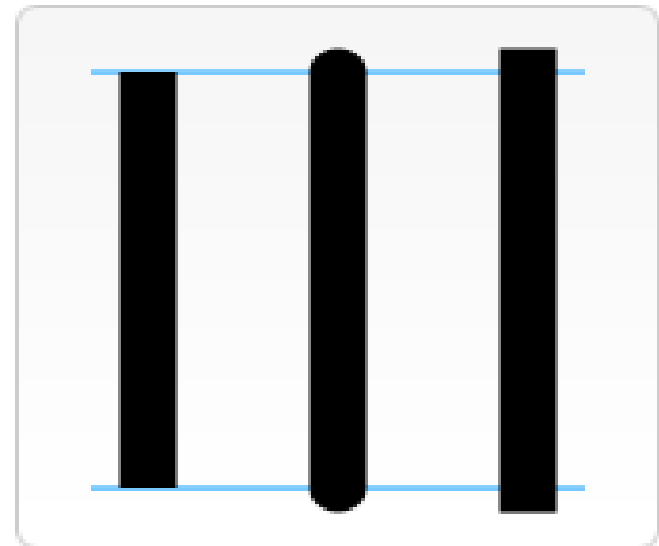
## lineCap

- La propiedad lineCap determina los puntos extremos. Hay tres posibles valores para esta propiedad: **butt**, **round** y **square**.
- `ctx.lineCap = valor;`



## Ejemplo - lineCap

```
• function draw() {
 var ctx = document.getElementById('canvas').getContext('2d');
 var lineCap = ['butt','round','square'];
 // dibuja guias
 ctx.strokeStyle = '#09f';
 ctx.beginPath();
 ctx.moveTo(10,10);
 ctx.lineTo(140,10);
 ctx.moveTo(10,140);
 ctx.lineTo(140,140);
 ctx.stroke();
 // dibuja lineas
 ctx.strokeStyle = 'black';
 for (var i=0;i<lineCap.length;i++){
 ctx.lineWidth = 15;
 ctx.lineCap = lineCap[i];
 ctx.beginPath();
 ctx.moveTo(25+i*50,10);
 ctx.lineTo(25+i*50,140);
 ctx.stroke();
 }
}
```



## lineJoin

- La propiedad **lineJoin** determina cómo dos líneas que conectan en forma se unen entre sí. Hay tres posibles valores para esta propiedad: **round**, **bevel** y **miter**. Por defecto esta propiedad se establece en **miter**.

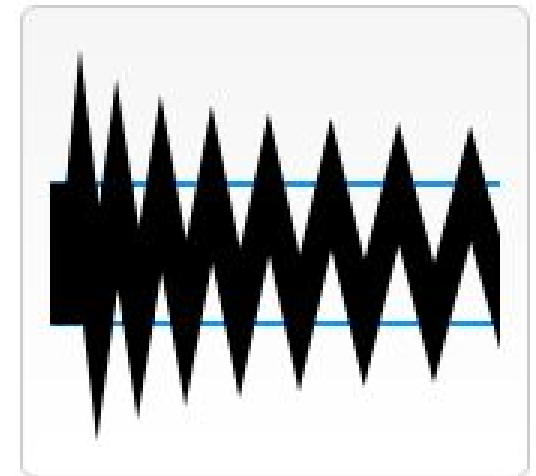
## Ejemplo - lineJoin

- ```
function draw() {  
  var ctx =  
document.getElementById('canvas').getContext('2d');  
  var lineJoin = ['round','bevel','miter'];  
  ctx.lineWidth = 10;  
  for (var i=0;i<lineJoin.length;i++){  
    ctx.lineJoin = lineJoin[i];  
    ctx.beginPath();  
    ctx.moveTo(-5,5+i*40);  
    ctx.lineTo(35,45+i*40);  
    ctx.lineTo(75,5+i*40);  
    ctx.lineTo(115,45+i*40);  
    ctx.lineTo(155,5+i*40);  
    ctx.stroke();  
  }  
}
```



MiterLimit

- La propiedad **miterLimit** determina hasta qué punto el punto de conexión exterior puede ser realizada desde el punto de conexión en el interior. Si dos líneas superan este valor se elaborará una unión biselada.





Degradados

- Creamos un objeto **canvasGradient** mediante uno de los métodos siguientes. Utilizamos este objeto para asignar a las propiedades `fillStyle` o `strokeStyle`.



Color

CreateLinearGradient(x1,y1,x2,y2)

- El método **createLinearGradient** toma cuatro argumentos que representan el punto de partida (**x1, y1**) y el punto final (**x2, y2**) del gradiente.

CreateRadialGradient(x1,y1,r1,x2,y2,r2)

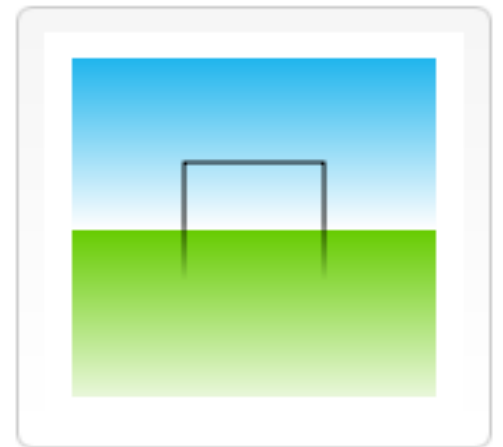
- El método createRadialGradient lleva seis argumentos. Los tres primeros argumentos definir un círculo con las coordenadas **(x1, y1)** y radio **r1** y el segundo, un círculo con las coordenadas **(x2, y2)** y el radio **r2**.

AddColorStop(position, color)

- Este método toma dos argumentos. La posición debe ser un número entre 0.0 y 1.0 y define la posición relativa de los colores en el degradado. El argumento de color debe ser una cadena que representa un color CSS (es decir, # FFF, rgba (0,0,0,1), etc.)

Ejemplo - createLinearGradient

- ```
function draw() {
 var ctx = document.getElementById('canvas').getContext('2d');
 // crea el degradado
 var lingrad = ctx.createLinearGradient(0,0,0,150);
 lingrad.addColorStop(0, '#00ABEB');
 lingrad.addColorStop(0.5, '#fff');
 lingrad.addColorStop(0.5, '#26C000');
 lingrad.addColorStop(1, '#fff');
 var lingrad2 = ctx.createLinearGradient(0,50,0,95);
 lingrad2.addColorStop(0.5, '#000');
 lingrad2.addColorStop(1, 'rgba(0,0,0,0)');
 // asigna el degradado al strokeStyle y fillStyle
 ctx.fillStyle = lingrad;
 ctx.strokeStyle = lingrad2;
 // draw shapes
 ctx.fillRect(10,10,130,130);
 ctx.strokeRect(50,50,50,50);
}
```





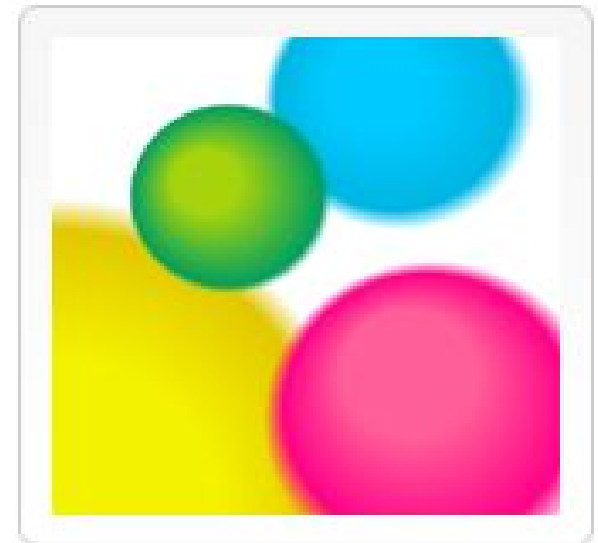
## Ejemplo - createRadialGradient

- ```
function draw() {  
  var ctx = document.getElementById('canvas').getContext('2d');  
  // crea el degradado  
  var radgrad = ctx.createRadialGradient(45,45,10,52,50,30);  
  radgrad.addColorStop(0, '#A7D30C');  
  radgrad.addColorStop(0.9, '#019F62');  
  radgrad.addColorStop(1, 'rgba(1,159,98,0)');  
  var radgrad2 = ctx.createRadialGradient(105,105,20,112,120,50);  
  radgrad2.addColorStop(0, '#FF5F98');  
  radgrad2.addColorStop(0.75, '#FF0188');  
  radgrad2.addColorStop(1, 'rgba(255,1,136,0)');  
  var radgrad3 = ctx.createRadialGradient(95,15,15
```



Ejemplo - createRadialGradient

- ```
radgrad3.addColorStop(0, '#00C9FF');
radgrad3.addColorStop(0.8, '#00B5E2');
radgrad3.addColorStop(1, 'rgba(0,201,255,0)');
var radgrad4 = ctx.createRadialGradient(0,150,50,0,140,90);
radgrad4.addColorStop(0, '#F4F201');
radgrad4.addColorStop(0.8, '#E4C700');
radgrad4.addColorStop(1, 'rgba(228,199,0,0)');
// dibuja formas
ctx.fillStyle = radgrad4;
ctx.fillRect(0,0,150,150);
ctx.fillStyle = radgrad3;
ctx.fillRect(0,0,150,150);
ctx.fillStyle = radgrad2;
ctx.fillRect(0,0,150,150);
ctx.fillStyle = radgrad;
ctx.fillRect(0,0,150,150);
}
```



## Patrones

- **CreatePattern(image,type)**
- Este método toma dos argumentos. La imagen es una referencia a un objeto imagen o un elemento de canvas diferente. La letra deberá ser una cadena que contiene uno de los siguientes valores: repeat, repeat-x,repeat-y y no-repeat.



# Color

- Una vez que hemos creado un modelo, lo podemos asignar a las propiedades `fillStyle` o `strokeStyle`.
- **Nota:** A diferencia del método `drawImage`, debe asegurarse de que la imagen que uso se carga antes de llamar a este método o el patrón se puede extraer correctamente.

## Ejemplo - createPattern

- ```
function draw() {  
  var ctx = document.getElementById('canvas').getContext('2d');  
  // crea un objeto image para usarlo en el pattern  
  var img = new Image();  
  img.src = 'images/wallpaper.png';  
  img.onload = function(){  
    // crea un pattern  
    var ptrn = ctx.createPattern(img,'repeat');  
    ctx.fillStyle = ptrn;  
    ctx.fillRect(0,0,150,150);  
  }  
}
```





Sombras

- **shadowOffsetX** = float
- **shadowOffsetY** = float
- **shadowBlur** = float
- **shadowColor** = color

- **shadowOffsetX** y **shadowOffsetY** indican hasta qué punto la sombra que se extienden desde el objeto en las direcciones **X** e **Y**, estos valores no se ven afectados por la matriz de transformación actual.
- Utilice valores negativos para que la sombra se amplie hacia arriba o hacia la izquierda, y los valores positivos para que la sombra se extienda hacia abajo o hacia la derecha. Estos son 0 por defecto.



Color

- **shadowBlur** indica el tamaño del desenfoque, este valor no se corresponde con un número de píxeles y no se ve afectada por la transformación de la matriz actual. El valor predeterminado es 0.
- **ShadowColor** es un color CSS, por omisión, es totalmente negro transparente.

Ejemplo - sombras

- ```
function draw() {
 var ctx =
document.getElementById('canvas').getContext('2d');
 ctx.shadowOffsetX = 2;
 ctx.shadowOffsetY = 2;
 ctx.shadowBlur = 2;
 ctx.shadowColor = "rgba(0, 0, 0, 0.5)";
 ctx.font = "20px Times New Roman";
 ctx.fillStyle = "Black";
 ctx.fillText("Sample String", 5, 30);
}
```

# HTML



3D, Graphics  
& Effects

# Transfor- maciones

## **save() y restore()**

- Se utilizan para guardar y recuperar el estado del canvas.
- Puede llamar al método save tantas veces como quieras.
- Cada vez que se llama al método de restauración el último estado guardado se devuelve de la pila y todos los ajustes guardados se restauran.



# Transformaciones

- **Los estados** del canvas se almacenan en una pila. Cada vez que el método **save** se llama el estado de dibujo actual se inserta en la pila:
  - Las transformaciones que se han aplicado.
  - Los valores de color, transparencia, patron, lineas, sombras.
  - La trayectoria actual de recorte.

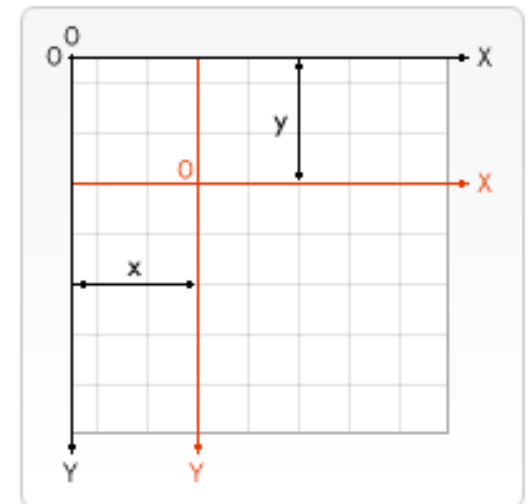
## Ejemplo – Save/Restore

- ```
function draw() {  
  var ctx = document.getElementById('canvas').getContext('2d');  
  ctx.fillRect(0,0,150,150); // dibuja un rectángulo  
  ctx.save();                // graba el estado por defecto  
  ctx.fillStyle = '#09F'    // hacemos cambios  
  ctx.fillRect(15,15,120,120); // dibujamos un nuevo rectángulo  
  ctx.save();                // guardamos el actual estado  
  ctx.fillStyle = '#FFF'    // hacemos cambios  
  ctx.globalAlpha = 0.5;  
  ctx.fillRect(30,30,90,90); // dibujamos un rectángulo  
  ctx.restore();            // restauramos al estado anterior  
  ctx.fillRect(45,45,60,60); // dibujamos un rectángulo con el estado anterior  
  ctx.restore();            // restauramos el estado original  
  ctx.fillRect(60,60,30,30); // dibujamos un rectángulo con el estado original  
}
```



`translate(x, y)`

- Este método toma dos argumentos. **x** es la cantidad que la **canvas** se mueve a la izquierda o a la derecha, **y** es la cantidad que se movió hacia arriba o hacia abajo.





Transformaciones

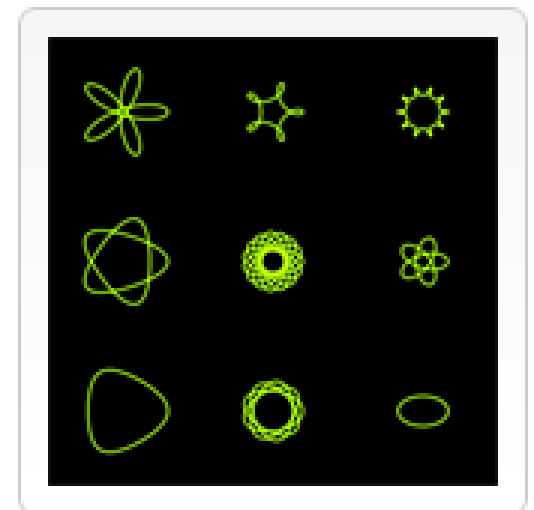
Ejemplo - translate

- ```
function draw() {
 var ctx = document.getElementById('canvas')
 .getContext('2d');
 ctx.fillRect(0,0,300,300);
 for (var i=0;i<3;i++) {
 for (var j=0;j<3;j++) {
 ctx.save();
 ctx.strokeStyle = "#9CFF00";
 ctx.translate(50+j*100,50+i*100);
 drawSpirograph(ctx,20*(j+2)/(j+1),-8*(i+3)/(i+1),10);
 ctx.restore();
 }
 }
}
```



# Transformaciones

```
• function drawSpirograph(ctx,R,r,O){
 var x1 = R-O;
 var y1 = 0;
 var i = 1;
 ctx.beginPath();
 ctx.moveTo(x1,y1);
 do {
 if (i>20000) break;
 var x2 = (R+r)*Math.cos(i*Math.PI/72) –
 (r+O)*Math.cos(((R+r)/r)*(i*Math.PI/72))
 var y2 = (R+r)*Math.sin(i*Math.PI/72) –
 (r+O)*Math.sin(((R+r)/r)*(i*Math.PI/72))
 ctx.lineTo(x2,y2);
 x1 = x2;
 y1 = y2;
 i++;
 } while (x2 != R-O && y2 != 0);
 ctx.stroke();
}
```



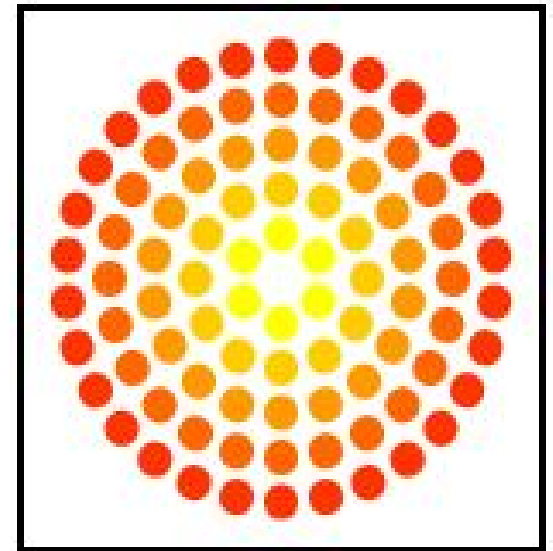


## `rotate(angle)`

- **Angle** es el ángulo de rotación del canvas. Un giro hacia la derecha mide en radianes.
- El punto central de rotación es siempre el origen del canvas.

## Ejemplo - rotate

```
• function draw() {
 var ctx = document.getElementById('canvas').getContext('2d');
 ctx.translate(75,75);
 for (var i=1;i<6;i++){ // Loop through rings (from inside to out)
 ctx.save();
 ctx.fillStyle = 'rgb('+ (51*i) + ',' + (255-51*i) + ',255)';
 for (var j=0;j<i*6;j++){ // draw individual dots
 ctx.rotate(Math.PI*2/(i*6));
 ctx.beginPath();
 ctx.arc(0,i*12.5,5,0,Math.PI*2,true);
 ctx.fill();
 }
 ctx.restore();
 }
}
```





# Transformaciones

## **scale(x, y)**

- Este método toma dos parámetros. **x** es el factor de escala en la dirección horizontal e **y** es el factor de escala en la dirección vertical.
- Ambos parámetros deben ser números reales, y no necesariamente positivo.
- Los valores menores a 1,0 reducir el tamaño de la unidad y los valores mayores a 1,0 aumentar el tamaño de la unidad.



## Ejemplo – scale

- .

# HTML



3D, Graphics  
& Effects

# Composición



# Composición

**globalCompositeOperation = type**

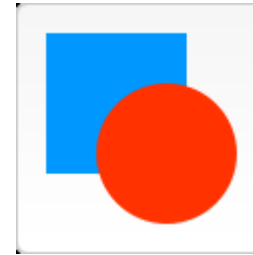
- **type** es una cadena que representa uno de las doce operaciones de composición.



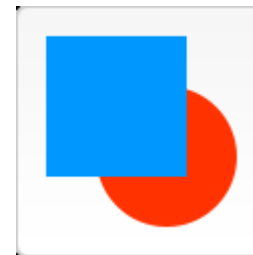
# Composición

**type:**

- source-over (default)



- Destination-over



- source-in





# Composición

**type:**

- destination-in
- source-out
- destination-out

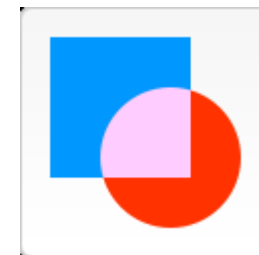
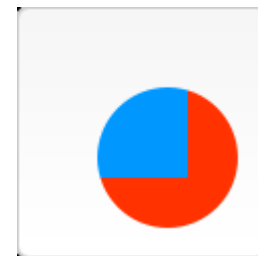
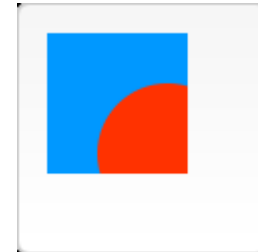




# Composición

**type:**

- Source-atop
- Destination-atop
- lighter





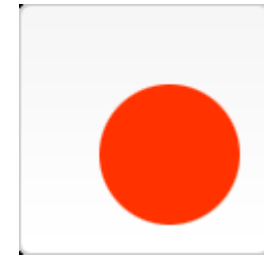
# Composición

**type:**

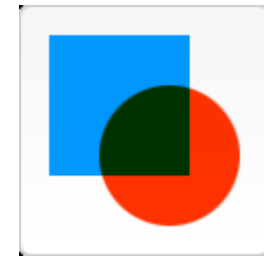
- Xor



- Copy



- Darker (Sin aplicación)



- **Nota:** En la actualidad el ajuste de **Copy** no hace nada en los navegadores basados en Gecko 1.8.



# Composición

## Ejemplo – globalCompositeOperation

- .



## Clip()

- Utilizamos el método de clip para crear un trazado de recorte nueva.



# Composición

## Ejemplo – clip

- .

# HTML



# Animaciones Básicas



3D, Graphics  
& Effects



# Animaciones Básicas

## Pasos básicos:

1. Borrar el canvas
2. Guardar el estado del canvas
3. Dibujar formas animadas
4. Restaurar el estado del canvas



# Animaciones Básicas

## Controlando una animación

- **setInterval(animateShape,milisegundos);**
  - Ejecuta repetidamente el código suministrado
- **setTimeout(animateShape,milisegundos);**
  - Se ejecuta una vez después de la cantidad de tiempo.
- **Nota:** Estas funciones son recomendadas si no se desea interacción con el usuario, de lo contrario usar los eventos de mouse y teclado.



## Ejemplo - Animaciones Básicas

- .

# HTML



# Preguntas



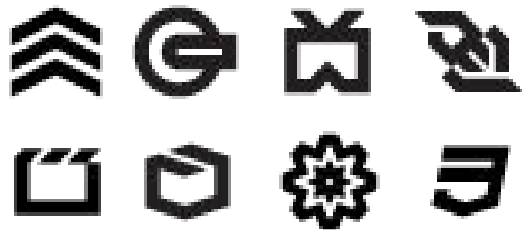
3D, Graphics  
& Effects

HTML



# Canvas

## Un lienzo en la Web



Miguel Angel Cumpa Ascuña  
miguelmiseck@mozilla.pe

