

Tema 4 – Gráficos 2D: Canvas

PROGRAMACIÓN MULTIMEDIA.

G.I.M. UNIVERSITAT DE VALÈNCIA

INMACULADA COMA

Introducción: el canvas y el contexto gráfico.

Introducción

- En este tema veremos cómo crear gráficos basados en bitmaps con HTML5 haciendo uso del **canvas**.
- El canvas permite:
 - Crear gráficos 2d
 - Añadirles interactividad
 - Añadirles animaciones
 - Crear por tanto juegos o animación sobre web.

Gráficos bitmaps: canvas

- El uso de la etiqueta **<canvas>** junto con un **contexto gráfico** 2D nos permite crear gráficos tipo bitmap directamente en HTML5.
- **<canvas>** es únicamente un contenedor para gráficos.
 - Atributos: width y height

```
<canvas id="miCanvas" width="200" height="100">  
</canvas>
```

- Se utilizará conjuntamente con un lenguaje de script, normalmente Javascript para dibujar.

Gráficos bitmaps: canvas

- El canvas tiene asociado un **contexto gráfico** que mantiene los estados de dibujo:
 - La matriz de transformaciones que se aplican.
 - La región de recorte del dibujo.
 - Valores de atributos de dibujado (colores de relleno, de líneas, etc.)
- El método **getContext()** nos devuelve un contexto con propiedades y métodos para dibujar sobre un canvas.

Ejemplo

```
<body>
<canvas id="miPrimerCanvas" width="200" height="100"
  |style= "border:1px solid #000000">
</canvas>
<script>
var c=document.getElementById("miPrimerCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,200,100);
</script>
</body>
```

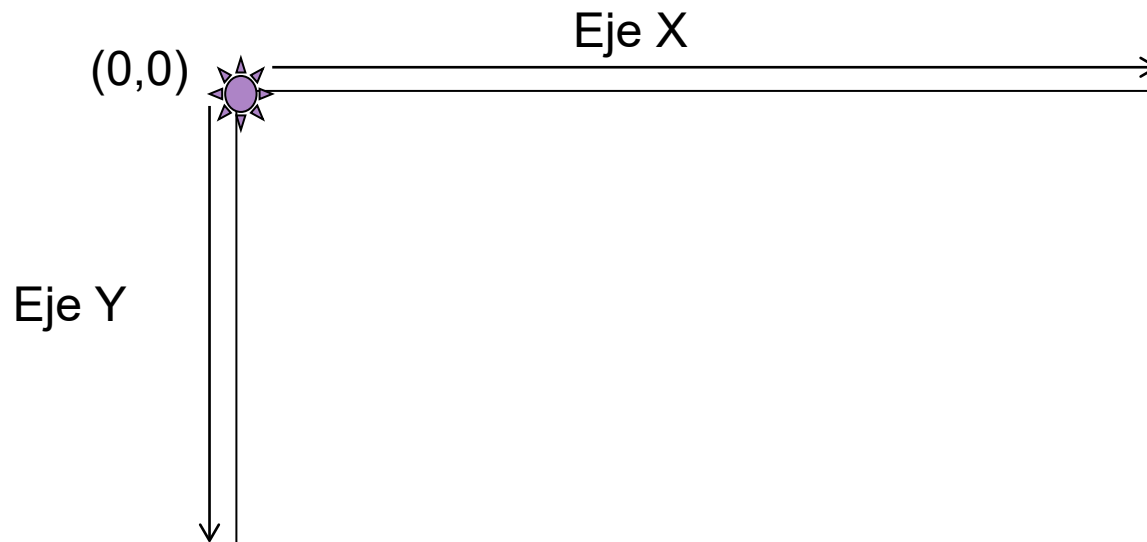


Contexto 2D

- El contexto 2D que obtenemos mediante **getContext('2D')** proporciona objetos, métodos y propiedades para dibujar sobre un canvas.
- Además, almacena una pila de estados de dibujo del canvas que contiene matrices de transformación y algunos atributos de dibujado.
- Los métodos **save()** y **restore()** permiten guardar y recuperar el estado.

Coordenadas del canvas

- El canvas es un rectángulo con dos dimensiones, con las coordenadas (0,0) en su esquina superior izquierda.



Guardando el contenido del canvas

- Podemos guardar el contenido generado en un canvas y exportarlo como un mapa de bits:
- `canvas.toDataURL('image/a.png');`

Dibujo básico sobre el canvas: líneas y figuras

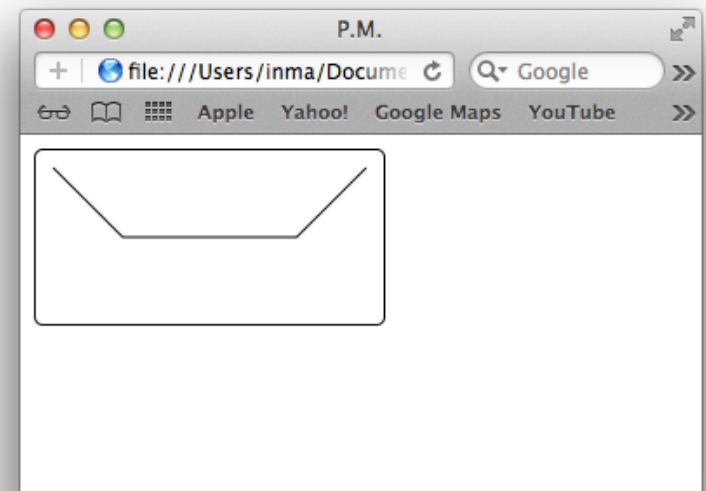
Dibujando objetos: líneas

- Para para dibujar **líneas** utilizamos dos métodos:
 - **moveTo(x,y)** : define posición inicial de la línea
 - **lineTo(x,y)** : define posición final
- Después tenemos que “pintar” la línea, podemos elegir dos formas:
 - **stroke()**:pinta el trazo de la línea
 - **fill()**:cierra la figura uniendo el punto final con el inicial y rellena la figura

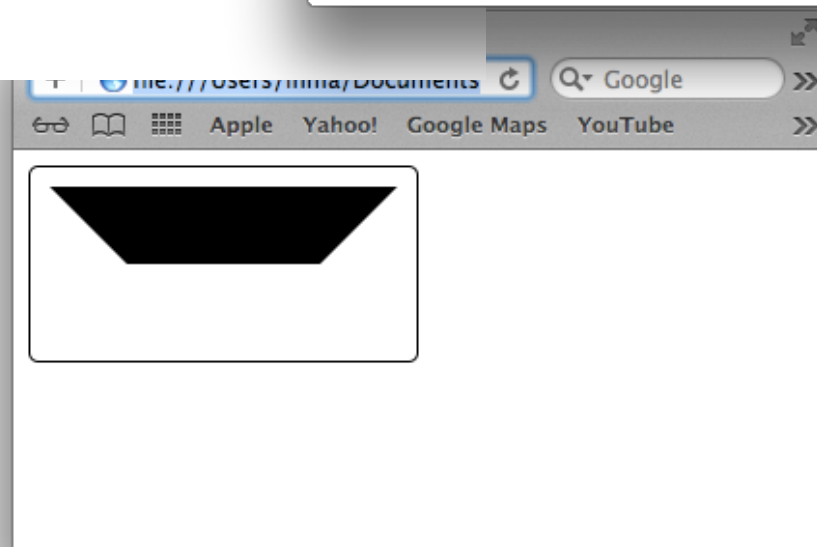
```
<body>
<canvas id="miPrimerCanvas" width="200" height="100"
  style="border-radius:5px ;border:1px solid #000000">
</canvas>
<script>
var c=document.getElementById("miPrimerCanvas");
var ctx=c.getContext("2d");
ctx.moveTo (10,10);
ctx.lineTo(50,50);
ctx.lineTo(150,50);
ctx.lineTo(190,10);
ctx.stroke();

</script>

</body>
```



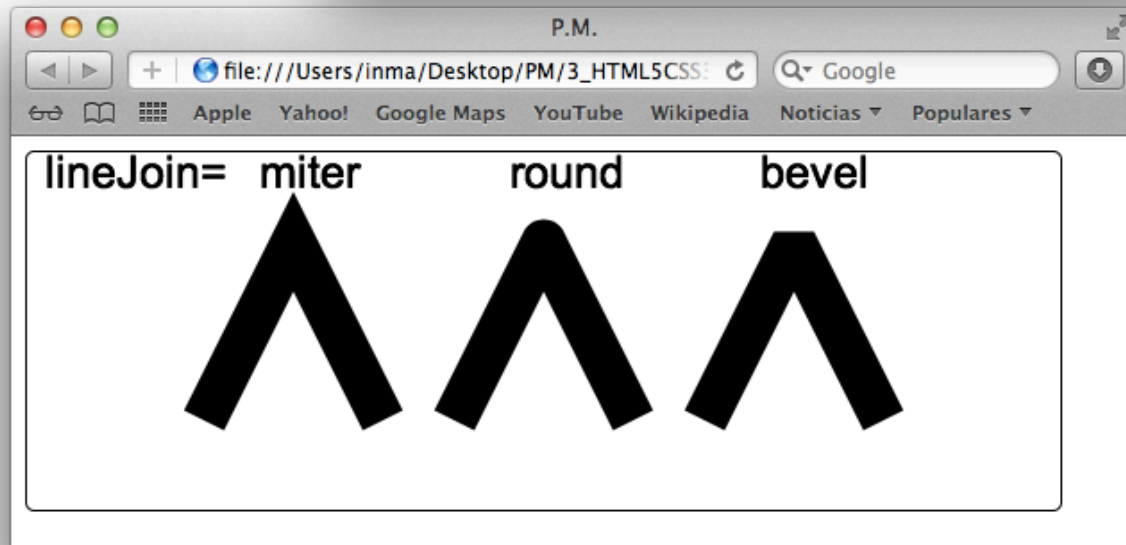
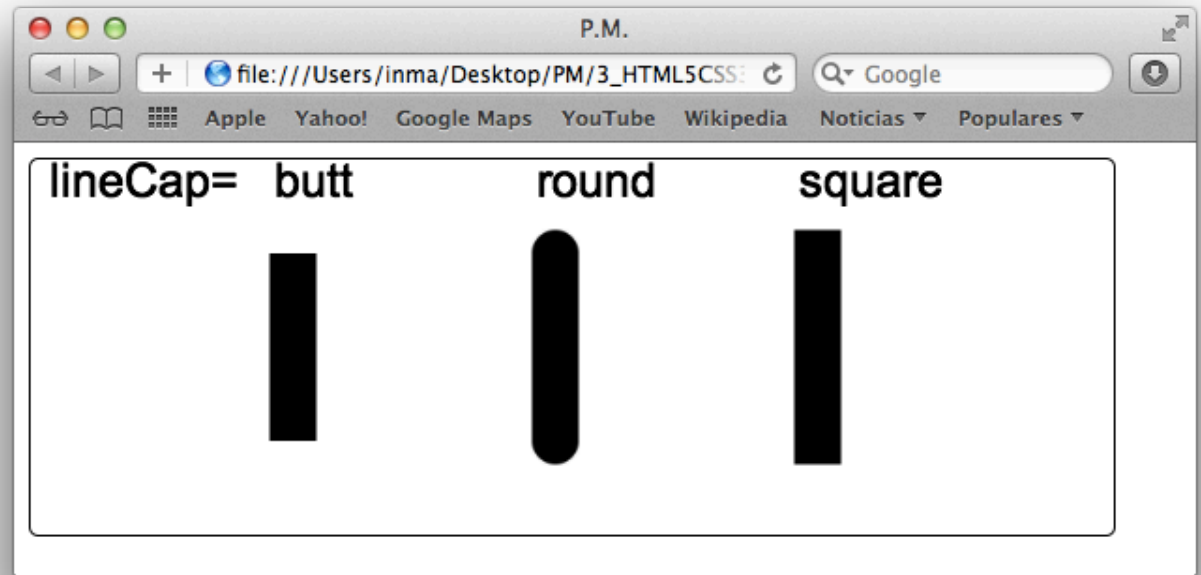
Misma figura con fill()



Propiedades de las líneas

- Además podemos seleccionar el estilo de la línea con los siguientes atributos:
 - `lineWidth = valor` : ancho de la línea
 - : entero > 0
 - `lineCap = valor` : aspecto del final de la línea
 - `butt` | `round` | `square`
 - `lineJoin = valor` : aspecto de la unión entre dos líneas
 - `bevel` | `round` | **`miter`**
 - `miterLimit = valor`
 - entero > 0
 - `setLineDash(segments)`
 - array de distancias de separación entre puntos de la línea
 - `segments = getLineDash()`
 - array de distancias de separación entre puntos de la línea

```
ctx.moveTo(280, 50);  
ctx.lineTo(280, 150);  
ctx.lineCap = 'round';  
ctx.stroke();
```



Formas: path

- Podemos crear figuras a base de líneas rectas o diferentes tipos de curvas usando un **path**.
- Este objeto nos permite en una misma forma geométrica contener diferentes curvas.
- Para ello empezaremos con:

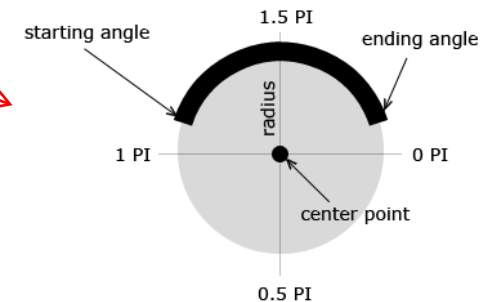
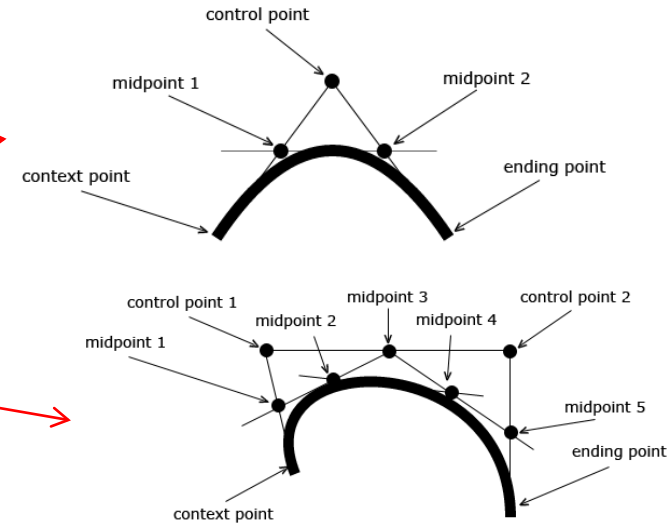
```
context.beginPath();  
context.moveTo(100, 20);
```

... formas a dibujar....

```
context.stroke(); // o context.fill();
```

Formas que podemos añadir al path

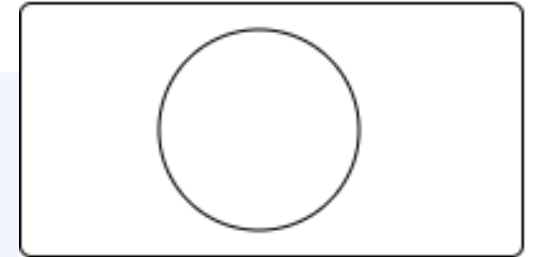
- `lineTo(x,y)`
- `quadraticCurveTo(cpx, cpy, x, y)`
- `bezierCurveTo`
`(cp1x, cp1y, cp2x, cp2y, x, y)`
- `arcTo(x1, y1, x2, y2, radius)`
- `arc(x, y, radius, startAngle, endAngle [, an`
`ticlockwise])`
- `rect(x, y, w, h)`
- `ellipse(x, y, radiusX, radiusY, rotation, star`
`tAngle, endAngle, anticlockwise)`



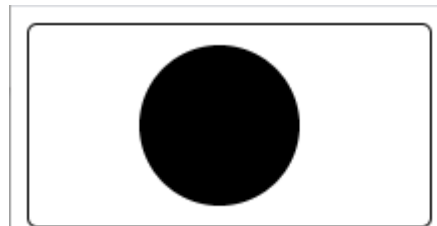
Ejemplo de path: círculo

- Utilizamos “arc” para crear un círculo

```
<body>
<canvas id="miPrimerCanvas" width="200" height="100"
  style="border-radius:5px ;border:1px solid #000000">
</canvas>
<script>
var c=document.getElementById("miPrimerCanvas");
var ctx=c.getContext("2d");
ctx.beginPath();
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
ctx.stroke();
</script>
```



- Lo mismo con `ctx.fill()` :



Otras funciones para dibujar el path

- Todas las funciones puede recibir un *path* como parámetro, o se aplican al path actual:
 - `beginPath()` – inicia el path o resetea el actual
 - `closePath()` – crea un path desde el punto inicial hasta el final
 - `stroke()` – dibuja la forma definida
 - `fill()` – dibuja la forma rellena
 - `clip()` – restringe el área de dibujo
 - `isPointInPath(x, y)` – devuelve true si el punto dado está en el path
 - `drawSystemFocusRing(element)` – si *element* tiene el foco dibuja un “focus ring” alrededor del path
 - `drawCustomFocusRing(element)`
 - `scrollpathIntoView()` – permite scroll del path en la vista (util para dispositivos con pequeñas pantallas)

```
<canvas id="myCanvas" width="578" height="200"></canvas>
<script>
  var canvas = document.getElementById('myCanvas');
  var context = canvas.getContext('2d');

  context.beginPath();
  context.moveTo(100, 20);

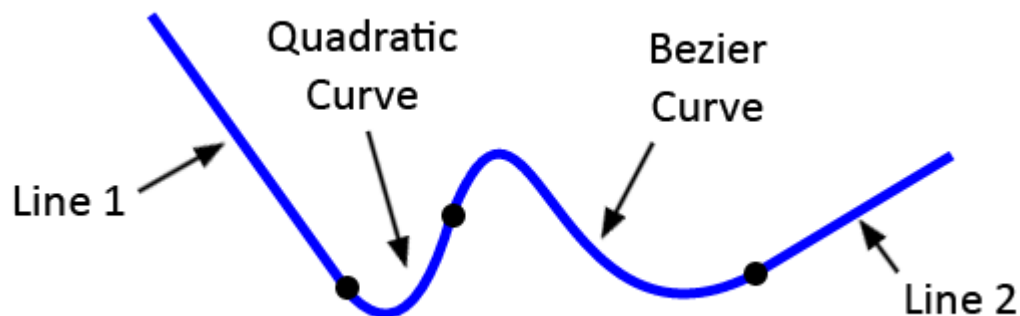
  // line 1
  context.lineTo(200, 160);

  // quadratic curve
  context.quadraticCurveTo(230, 200, 250, 120);

  // bezier curve
  context.bezierCurveTo(290, -40, 300, 200, 400, 150);

  // line 2
  context.lineTo(500, 90);

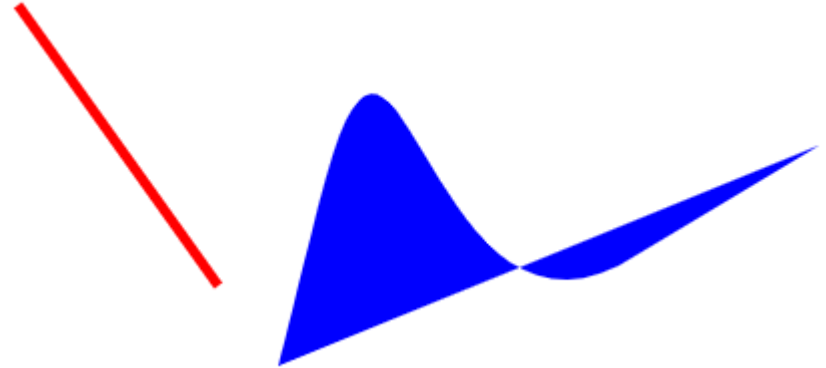
  context.lineWidth = 5;
  context.strokeStyle = 'blue';
  context.stroke();
</script>
```



Ejemplo de path

```
ctx.beginPath();  
  // line 1  
  ctx.moveTo(100, 20);  
  ctx.lineTo(200, 160);  
  ctx.strokeStyle = 'red';  
  ctx.lineWidth = 5;  
  ctx.stroke();  
ctx.closePath();
```

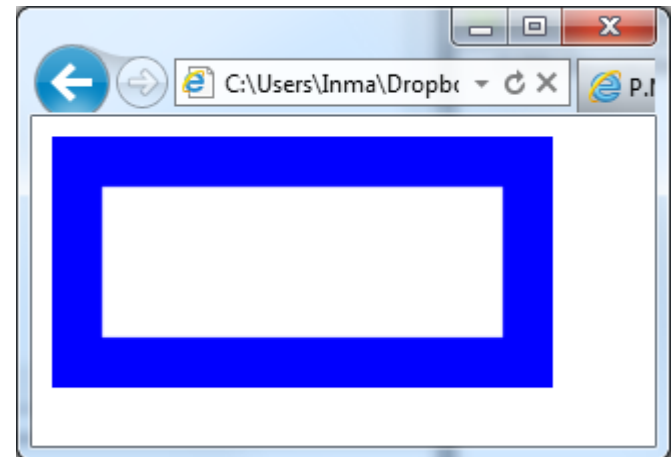
```
//Si empezamos un nuevo path no se juntan las figuras  
ctx.beginPath();  
  // quadratic curve  
  ctx.quadraticCurveTo(230, 200, 250, 120);  
  // bezier curve  
  ctx.bezierCurveTo(290, -40, 300, 200, 400, 150);  
  // line 2  
  ctx.lineTo(500, 90);  
  ctx.fillStyle = 'blue';  
  ctx.fill();  
ctx.closePath();
```



Rectángulos

- Dibujar un rectángulo en el canvas:
 - `clearRect(x, y, w, h)` – borra los píxeles del canvas del rectángulo dado
 - `fillRect(x, y, w, h)` – dibuja un rectángulo relleno utilizando el estilo de relleno definido
 - `strokeRect(x, y, w,)` – dibuja una caja utilizando el estilo definido

```
ctx.fillStyle = "blue";  
ctx.fillRect(0, 0, 200, 100);  
ctx.clearRect(20, 20, 160, 60);
```



Colores, estilos y sombras

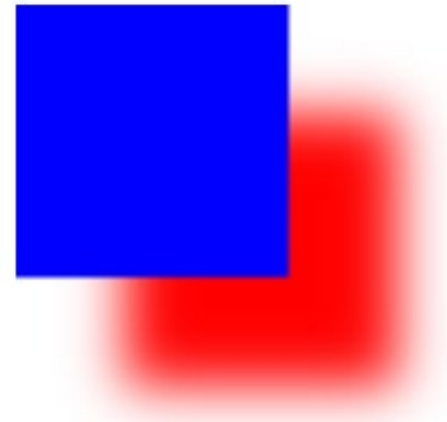
Estilo de relleno

- El estilo de relleno de las figuras o el aspecto de los contornos y las sombras los podemos elegir.
- `fillStyle = "valor"` – define el estilo de relleno de una figura rellena.
 - CSS color | gradient | pattern
`context.fillStyle = "blue"`
- `strokeStyle = "valor"` – define el estilo de dibujo de una forma.
 - CSS color | gradient | pattern

Sombras

- Sombras:
 - shadowColor = "color" – color utilizado en las sombras
 - shadowBlur = num – nivel de difuminado en las sombras
 - shadowOffsetX = num – distancia horizontal entre la sombra y la forma
 - shadowOffsetY = num - distancia vertical entre la sombra y la forma


```
ctx.shadowColor="red";  
ctx.shadowBlur=10;  
ctx.shadowOffsetX=20;  
ctx.shadowOffsetY=20;  
  
ctx.fillStyle ="blue";  
ctx.fillRect(0,0,50,50);
```



Basic color keywords:

Color names and sRGB values

Lista detallada de colores:
<http://dev.w3.org/csswg/css3-color/>

	Color name	Hex rgb	Decimal
	<i>black</i>	#000000	0,0,0
	<i>silver</i>	#C0C0C0	192,192,192
	<i>gray</i>	#808080	128,128,128
	<i>white</i>	#FFFFFF	255,255,255
	<i>maroon</i>	#800000	128,0,0
	<i>red</i>	#FF0000	255,0,0
	<i>purple</i>	#800080	128,0,128
	<i>fuchsia</i>	#FF00FF	255,0,255
	<i>green</i>	#008000	0,128,0
	<i>lime</i>	#00FF00	0,255,0
	<i>olive</i>	#808000	128,128,0
	<i>yellow</i>	#FFFF00	255,255,0
	<i>navy</i>	#000080	0,0,128
	<i>blue</i>	#0000FF	0,0,255
	<i>teal</i>	#008080	0,128,128
	<i>aqua</i>	#00FFFF	0,255,255

Colores: gradiente.

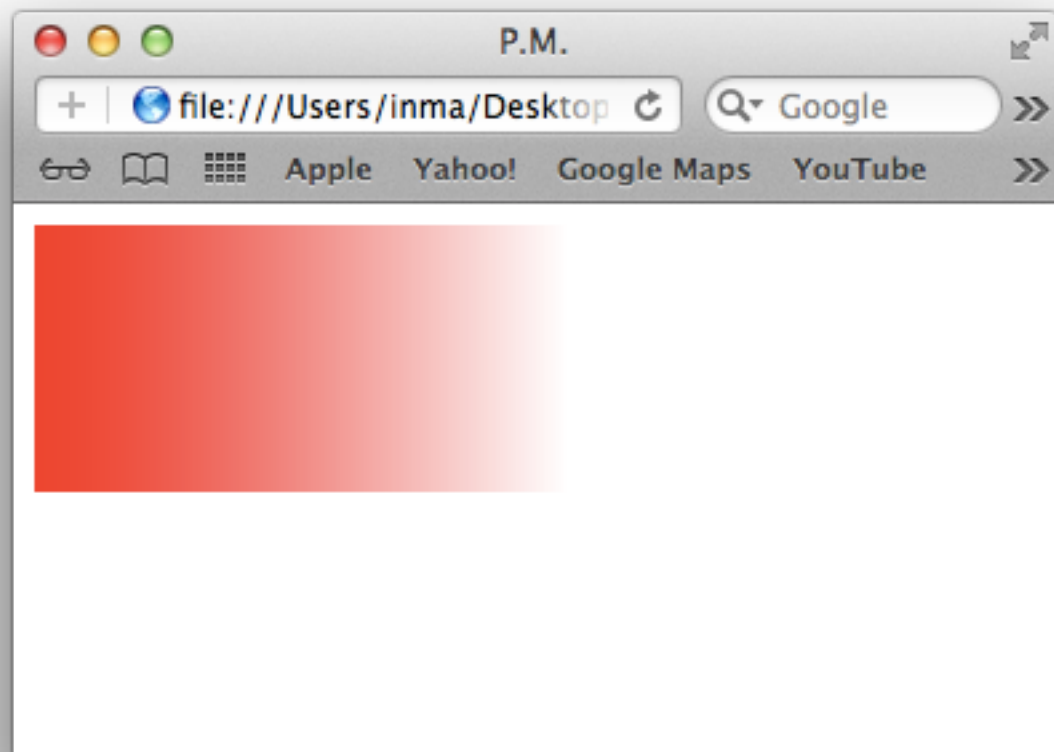
- `grad = ctx.createLinearGradient(x, y, x1,y1);`
- `grad = ctx.createRadialGradient(x, y, r, x1,y1, r1);`
- `grad.addColorStop(pos,color);` Se pueden añadir dos colores, el inicial (`pos =0`) y el final (`pos =1`).
- A continuación se dibuja una figura, eligiendo previamente el estilo de relleno como el gradiente creado

```
<body>
<canvas id="miPrimerCanvas" width="200" height="100" >
</canvas>
<script>
var c=document.getElementById("miPrimerCanvas");
var ctx=c.getContext("2d");

var grd=ctx.createLinearGradient(0,0,200,0);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

ctx.fillStyle=grd;
ctx.fillRect(0,0,200,100);
</script>

" "
```

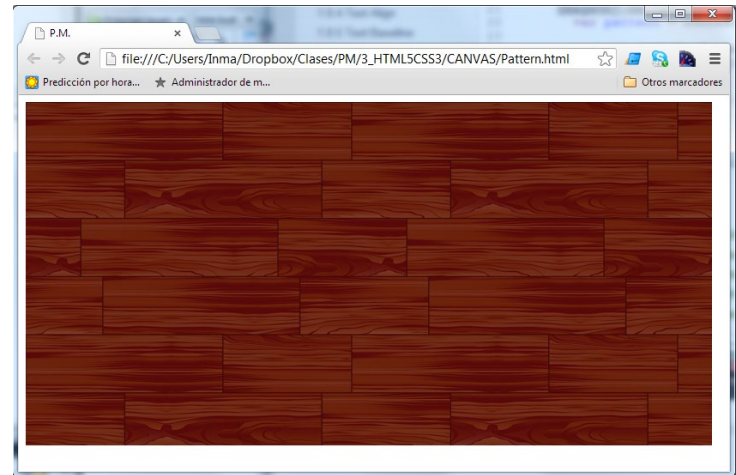


Patrones de relleno

- `pattern = createPattern(image, repetition)`
 - `image`: instancia de un elemento HTML tipo imagen, canvas o video
 - `repetition`: **repeat** | repeat-x | repeat-y | no-repeat

```
<script>
var canvas=document.getElementById("miCanvas");
var ctx=canvas.getContext("2d");
var image = new Image();
image.src = "woodpattern.png"

image.onload = function(){
var pat = ctx.createPattern(image,'repeat');
ctx.fillStyle = pat;
ctx.fillRect(0,0,canvas.width,canvas.height);
}
</script>
```



Texto e imágenes

Texto

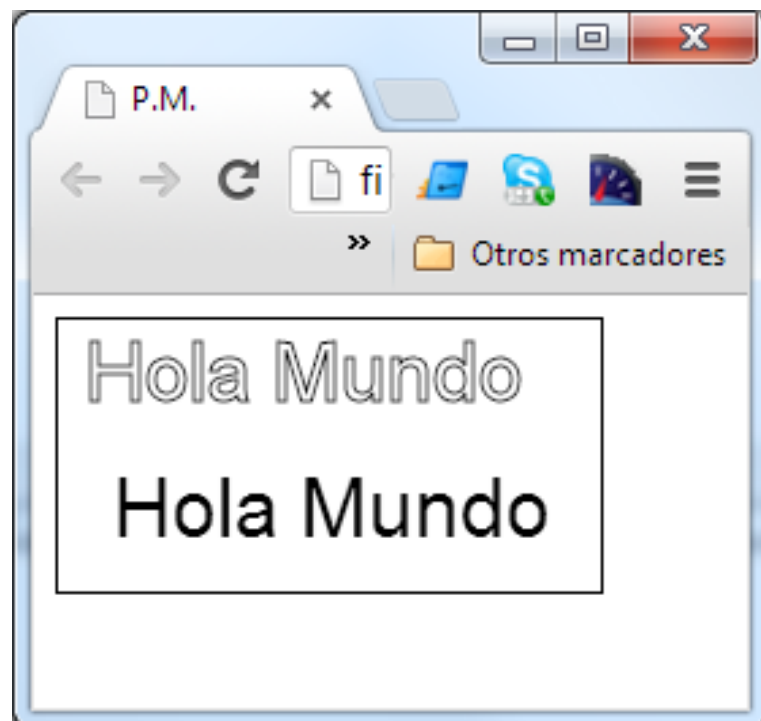
- Dibujar un texto en un canvas:
 - `fillText(texto, x, y)` – dibuja texto relleno en coordenadas
 - `strokeText(texto, x, y)` – texto hueco
- Estilos del texto, atributos:
 - `font` = “fuente” – define el tipo de fuente
 - Fuentes-> <http://www.w3.org/TR/css3-fonts/>
 - `textAlign` = “valor” – alineación del texto respecto a las coordenadas
 - **start** | end | left | right | center
 - `textBaseline` = “valor” – alineación vertical del texto
 - top | hanging | middle | **alphabetic** | ideographic | bottom
 - `measureText()` – devuelve el ancho de un texto
`ancho = ctx.measureText("Hola Mundo").width;`

Texto

```
<body>
<canvas id="miCanvas" width="200" height="100"
  style="border:1px solid #000000">
</canvas>
<script>
var c=document.getElementById("miCanvas");
var ctx=c.getContext("2d");

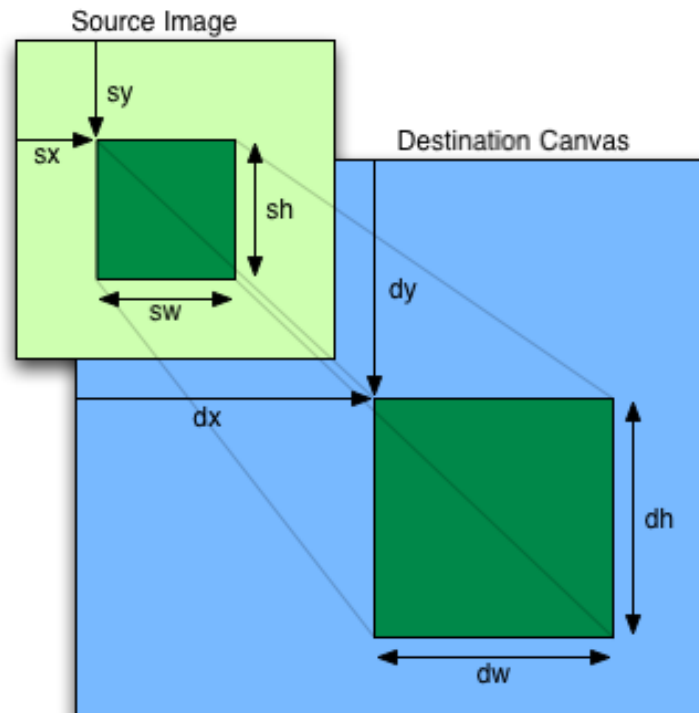
ctx.font="30px Arial";
ctx.textAlign ="start";
ctx.strokeText("Hola Mundo",10,30);

ctx.textAlign ="center";
ctx.fillText("Hola Mundo",100,80);
</script>
</body>
```



Imágenes

- `drawImage(image, dx, dy)`
- `drawImage(image, dx, dy, dw, dh)`
- `drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)`



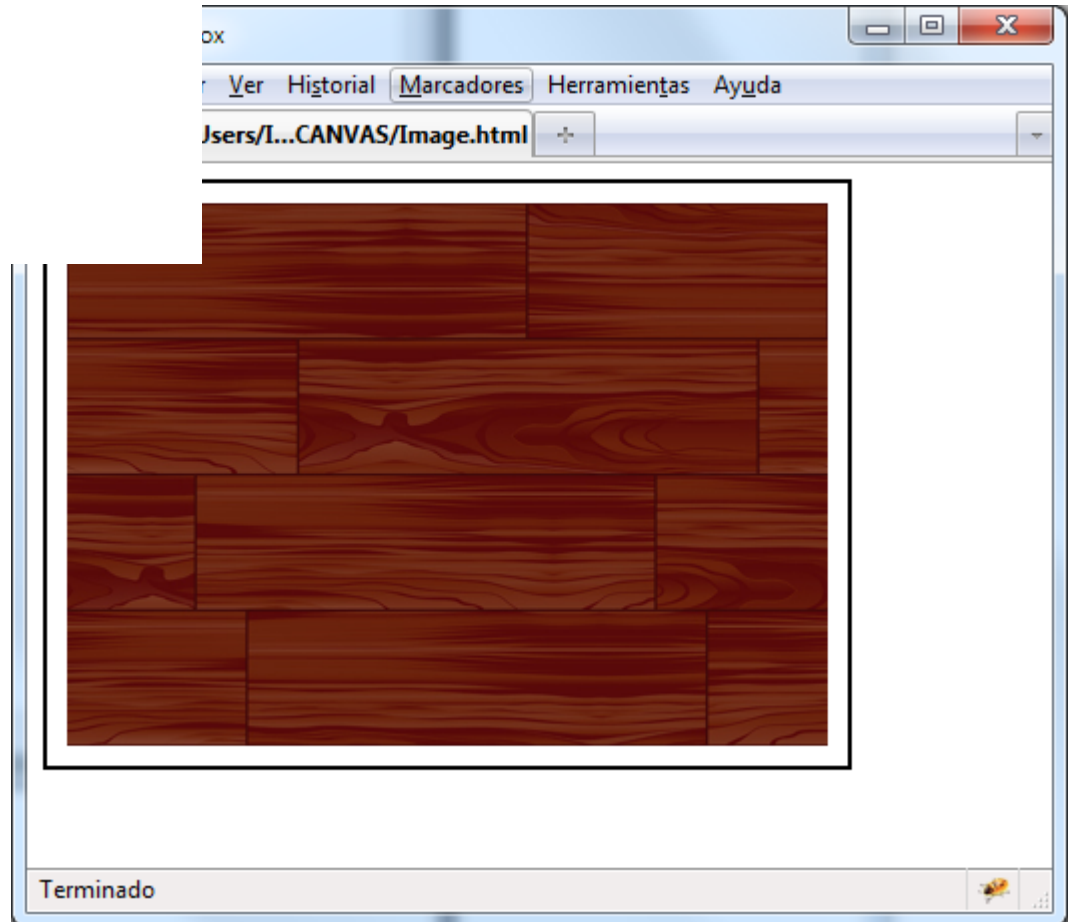

```
<script>
var canvas=document.getElementById("myCanvas");
var ctx=canvas.getContext("2d");

var image = new Image();
image.src = "woodpattern.png";

canvas.width = image.width +20;
canvas.height = image.height+20;

image.onload = function() {
ctx.drawImage(image,10,10);
}
```

onload-> para
asegurarse que
la imagen se ha
cargado cuando
intentamos
dibujarla



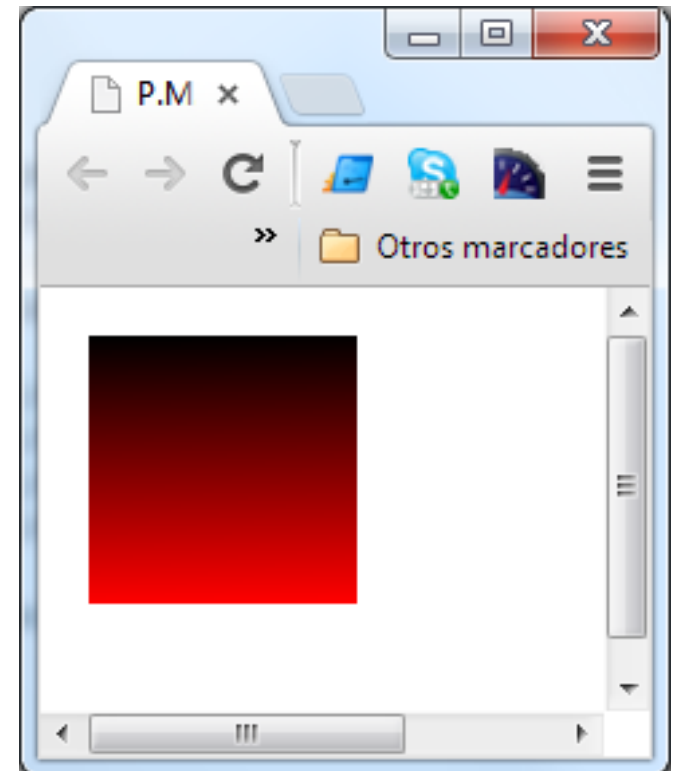
ImageData: Imágenes a nivel de píxel

- ImageData: objeto que permite crear imágenes definiendo el valor de sus píxeles.
 - `imagedata = createImageData(w, h)` – crea y devuelve un objeto *ImageData* y con los píxeles todos negros
 - `getImageData(x,y,w,h)` – devuelve un objeto *ImageData* a partir de un rectángulo del canvas
 - `putImageData(ImageData, dx,dy)` – dibuja un *ImageData* en un canvas
- Propiedades:
 - `width, height` : ancho y alto de un objeto *ImageData*
 - `data`: array unidimensional con valores RGBA de color.

ImageData

```
<script>
var c=document.getElementById("miCanvas");
var ctx=c.getContext("2d");

var imgData=ctx.createImageData(100,100);
var l = imgData.data.length;
for (var i=0;i<l;i+=4)
{
    imgData.data[i+0]=255*i/l;
    imgData.data[i+1]=0;
    imgData.data[i+2]=0;
    imgData.data[i+3]=255;
}
ctx.putImageData(imgData,10,10);
</script>
```



Transformaciones, colisiones, composiciones

Transformaciones

- Existe una matriz de transformaciones que se aplica sobre todos los objetos que se dibujan a continuación y puede ser modificada:
 - `scale(x,y)`
 - `rotate(angle)` – en radianes
 - `translate(x,y)` – modifica el punto inicial de los objetos sumándoles x e y
 - `transform(a,b,c,d,e,f)` – reemplaza la matriz de transformación
 - `setTransform(a,b,c,d,e,f)` – resetea la matriz a la de identidad y luego aplica la matriz

Transformaciones

- Cuidado con el orden, se aplican en orden inverso y el resultado no es el mismo dependiendo del orden.

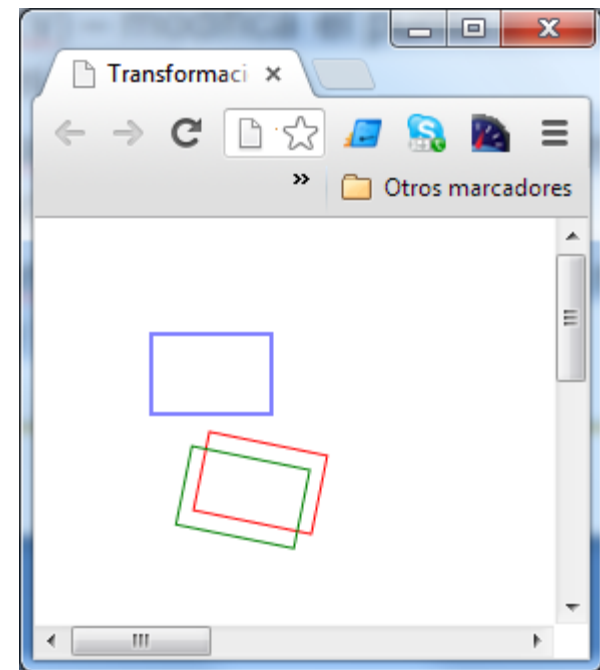
```
<script>
var c=document.getElementById("miPrimerCanvas");
var ctx=c.getContext("2d");
ctx.strokeStyle="blue";
ctx.strokeRect(50,50,60,40);

ctx.translate(40,40);
ctx.rotate(0.2);
ctx.strokeStyle="red";
ctx.strokeRect(50,50,60,40);

ctx.setTransform(1,0,0,1,0,0);

ctx.rotate(0.2);
ctx.translate(40,40);
ctx.strokeStyle="green";
ctx.strokeRect(50,50,60,40);

</script>
```



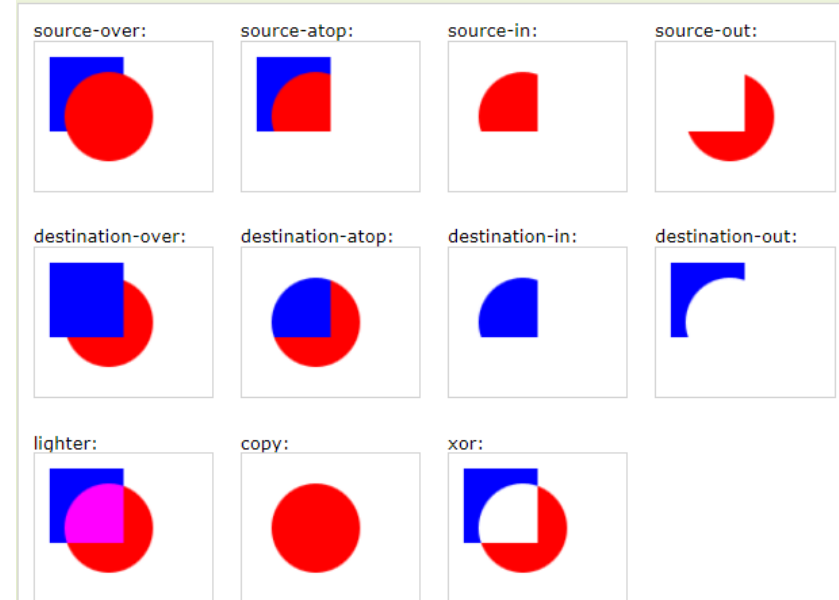
Composición

- `globalAlpha` = valor – modifica el valor de transparencia que se aplicará en el dibujado
- `globalCompositeOperation` = valor – define cómo una imagen nueva (`source`) se mezclará con una imagen ya existente (`destination`)
- `source-over` | `source-atop` | `source-in` | `source-out` | `destination-over` | `destination-atop` | `destination-in` | `destination-out` | `lighter` | `copy` | `xor`



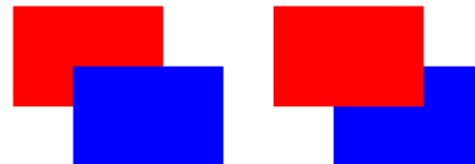
Example

All the `globalCompositeOperation` property values:



Composición

```
ctx.fillStyle = "red";  
ctx.fillRect(20, 20, 75, 50);  
ctx.globalCompositeOperation = "source-over";  
ctx.fillStyle = "blue";  
ctx.fillRect(50, 50, 75, 50);  
  
ctx.fillStyle = "red";  
ctx.fillRect(150, 20, 75, 50);  
ctx.globalCompositeOperation = "destination-over";  
ctx.fillStyle = "blue";  
ctx.fillRect(180, 50, 75, 50);
```



Interactividad y animaciones

Eventos y coordenadas del canvas

- Sobre el canvas podemos recoger eventos

```
<canvas id="myCanvas" width="800" height="400" >
</canvas>
```

```
<script>
```

```
var canvas=document.getElementById("myCanvas");
```

```
var ctx=canvas.getContext("2d");
```

```
function ProcessClick(event)
```

```
{
    x = event.clientX;
    y = event.clientY;
```



OJO CON LAS COORDENADAS!
Son del área cliente, no del canvas

```
}
```

```
canvas.addEventListener('click',ProcessClick,true );
```

```
</script>
```

```
</body>
```

```
</html>
```

Eventos y coordenadas del canvas

- Si queremos coordenadas relativas a la posición del canvas:

```
xCanvas = event.clientX - canvas.offsetLeft;  
yCanvas = event.clientY - canvas.offsetTop;
```

- O bien:

```
var rect = canvas.getBoundingClientRect();  
xCanvas = event.clientX - rect.left;  
yCanvas = event.clientY - rect.top;
```



Animaciones

- Podemos crear animaciones en el canvas de dos formas :
- Con los temporizadores (setInterval o setTimeout) que ya existían (forzando la llamada a la función de redibujado cada X milisegundos):
 - `intervalD = window.setInterval(función_callback, tiempo_mS);`
 - `window.clearInterval(ID);`
- Dentro de la función de callback sabremos que el tiempo transcurrido cada vez que se la llama es la variable `tiempo_mS`

Animación con setInterval

```
<script>
var theCanvas=document.getElementById("myCanvas");
var context=theCanvas.getContext("2d");

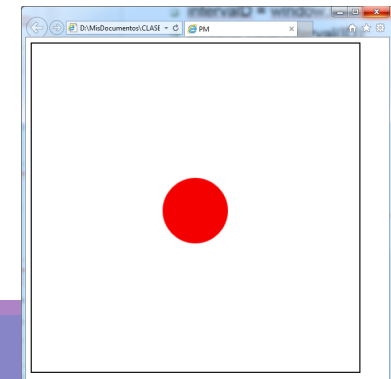
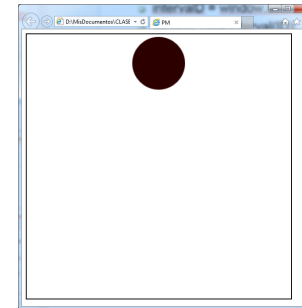
var speed = 5;
var y = 10;
var x = 250;
var intervalID;
var radio = 50;

window.addEventListener('load', eventWindowLoaded, false);
function eventWindowLoaded() {
    initAnimation();
}
function initAnimation(){
    intervalID = setInterval(drawScreen, 33);
}
function drawScreen(){
    context.clearRect(0,0,theCanvas.width, theCanvas.height);
    y += speed;

    context.fillStyle = "rgb("+ y +",0,0)";
    context.beginPath();
    context.arc(x,y,radio,0,Math.PI*2,true);
    context.closePath();
    context.fill();

    if(y > theCanvas.height/2)
        clearInterval(intervalID);
}
</script>
```

Ejemplo: pelota que cae hasta la mitad de la ventana y va cambiando de color.



Animaciones

- Con *requestAnimationFrame*, que deja al navegador que llame a la función de redibujado “cuando pueda”.
- Cuando la ventana no está visible no sigue redibujando optimizando el funcionamiento.
- Especificación (<http://www.w3.org/TR/animation-timing/>):
 - `long requestAnimationFrame(FrameRequestCallback f_callback);`
 - `void cancelAnimationFrame(long handle);`
 - `callback FrameRequestCallback = void (DOMHighResTimeStamp time);`

Animaciones

- *requestAnimationFrame* solo ejecuta una llamada a la *f_callback*. Si queremos que vuelva a ser llamada (porque estamos en una animación) pondremos el *requestAnimationFrame* dentro de la propia función.
- La *f_callback* recibe un parámetro con el tiempo transcurrido (*DOMHighResTimeStamp time*) que podemos utilizar para calcular el tiempo transcurrido.
- *High-res time stamp* es el tiempo transcurrido desde la carga de la página

Ejemplo con requestAnimationFrame y usando el time

```
<style>
div { position: absolute; left: 10px; padding: 50px;
    background: crimson; color: white }
</style>
<script>
var requestId = 0;

function animate(time) {
    document.getElementById("animated").style.left =
        (time - animationStartTime) % 2000 / 4 + "px";
    requestId = window.requestAnimationFrame(animate);
    document.getElementById("text").innerHTML = time/1000 + "secs";
}

function start() {
    animationStartTime = window.performance.now();
    requestId = window.requestAnimationFrame(animate);
}

function stop() {
    if (requestId)
        window.cancelAnimationFrame(requestId);
    requestId = 0;
}
</script>
<button onclick="start()">Click me to start!</button>
<button onclick="stop()">Click me to stop!</button>
<div id="animated"> <p id="text">Hello</p></div>
```

Click me to start! Click me to stop!

Hello

Click me to start! Click me to stop!

6.693947000021581secs

Ejemplo con requestAnimationFrame y usando el Date()

```
<body>
  <script>
    var canvas, context;
    init();
    animate();

    function init() {
      canvas = document.createElement( 'canvas' );
      canvas.width = 256;
      canvas.height = 256;
      context = canvas.getContext( '2d' );
      document.body.appendChild( canvas );
    }

    function animate() {
      requestAnimationFrame( animate );
      draw();
    }

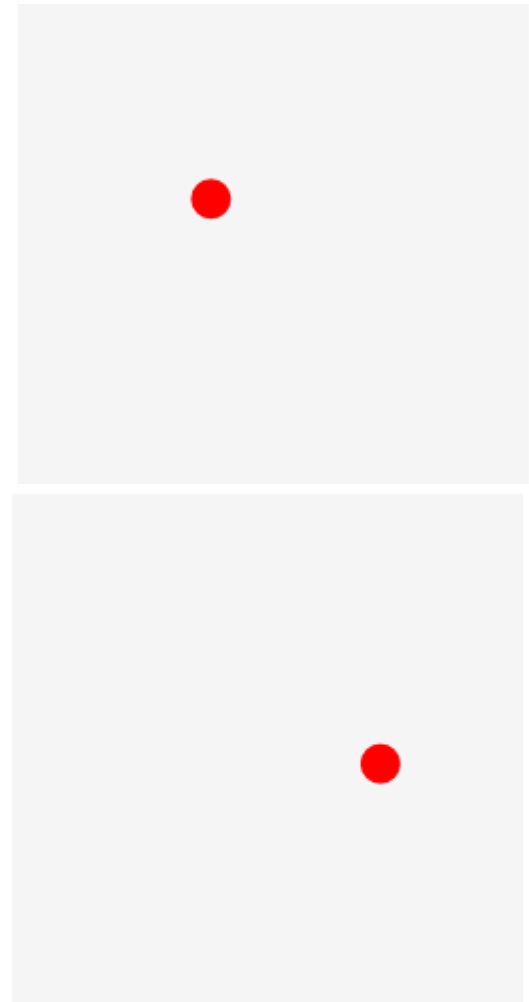
    function draw() {

      var time = new Date().getTime() * 0.002;
      var x = Math.sin( time ) * 96 + 128;
      var y = Math.cos( time * 0.9 ) * 96 + 128;

      context.fillStyle = 'rgb(245,245,245)';
      context.fillRect( 0, 0, 255, 255 );

      context.fillStyle = 'rgb(255,0,0)';
      context.beginPath();
      context.arc( x, y, 10, 0, Math.PI * 2, true );
      context.closePath();
      context.fill();

    }
  </script>
</body>
```



Bibliografía y referencias.

- Especificaciones: HTML Canvas 2D Context
 - <http://www.w3.org/TR/2dcontext/>
- HTML5 Canvas. S. Fulton. O'Reilly.
 - <http://proquest.safaribooksonline.com/book/web-development/html/9781449308032>
- Páginas de ayuda:
 - http://www.w3schools.com/tags/ref_canvas.asp
 - <https://developer.mozilla.org/es/docs/Web/HTML/Canvas>