

# HTML Canvas

Escuela Técnica Superior de Ingeniería de Telecomunicación  
Universidad Rey Juan Carlos

gsync-profes (arroba) gsync.urjc.es

Mayo de 2021



©2021 GSyC  
Algunos derechos reservados.  
Este trabajo se distribuye bajo la licencia  
Creative Commons Attribution Share-Alike 4.0

# HTML Canvas

El *canvas* (lienzo) de HTML es un API incluido en HTML 5 para dibujar gráficos de mapas de bits en 2 dimensiones

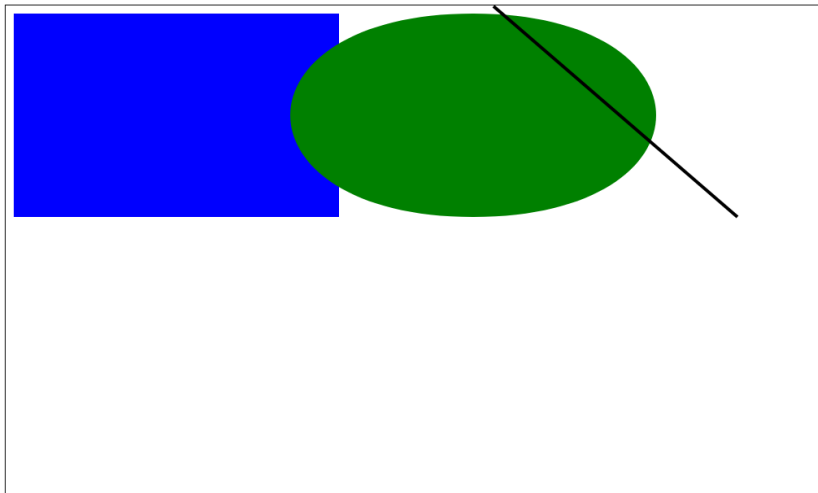
- Desarrollado originalmente por Apple en 2004 para el Mac OS X WebKit, adoptado posteriormente por Firefox, incluido por el WHATWG en el estándar HTML 5
- Muy sencillo, permite dibujar dinámicamente en javascript círculos, líneas, gradientes y texto sobre una matriz de pixeles. Además de cargar imágenes desde un fichero

- Para facilitar el uso del canvas, podemos usar jCanvas, un módulo de más alto nivel, basada en jQuery (no estándar ni parte de jQuery)  
Incluye polígonos, curvas, elipses, máscaras, capas y algunos otros elementos  
<https://projects.calebevans.me/jcanvas>
- Usar módulos tiene una consecuencia importante: no se pueden ver las páginas leyendo en local los ficheros. Es necesario servirlos desde un servidor web.
  - Para que el desarrollador vea su propio código no es necesario un servidor *de verdad* (apache, Nginx, etc), podemos utilizar servidores sencillos específicos para esto como p.e. el *Chrome Web Server* o cualquier otro<sup>1</sup>

---

<sup>1</sup>Consulta el tema [javascript..ii.pdf](#) para más detalles

# Ejemplo de uso de jCanvas



[http://ortuno.es/ejemplo\\_jcanvas.html](http://ortuno.es/ejemplo_jcanvas.html)

```
<!DOCTYPE html>
<html lang="es-ES">
<head>
  <meta charset="utf-8">
  <title>Ejemplo jCanvas</title>
  <style type="text/css">
    canvas {
      border: 1px solid black;
    }
  </style>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jcanvas/21.0.1/jcanvas.js"></script>
</head>

<body>

  <canvas id="mi_canvas" width="1000" height="600">
    <p>Texto alternativo para navegadores sin el API Canvas.</p>
  </canvas>

  <script type="module" src="ejemplo_jcanvas.js"> </script>
</body>
</html>
```

```
'use strict'
```

```
$('#canvas').drawRect({  
  fillStyle: "blue",  
  x: 10,  
  y: 10,  
  width: 400,  
  height: 250,  
  fromCenter: false  
});
```

```
$('#canvas').drawEllipse({  
  fillStyle: "green",  
  strokeWidth: 1,  
  x: 350,  
  y: 10,  
  width: 450,  
  height: 250,  
  fromCenter: false  
});
```

# Cambio de coordenadas

- Tanto con HTML canvas como con jCanvas tenemos una limitación muy importante: es necesario especificar las coordenadas en pixeles
- Esto es bastante incómodo. Nos obliga a diseñar nuestro gráfico para un tamaño concreto, estamos atados a ese tamaño. Si posteriormente necesitamos redimensionar, habría que rehacer todos los algoritmos

Lo razonable es usar dos *sistemas de coordenadas* y un *cambio de coordenadas*. Esto tendremos que programarlo nosotros



## Sistema de coordenadas gráfico

- Es el sistema de coordenadas que maneja el canvas HTML, la unidad es el pixel
- La posición 0,0 es la esquina superior izquierda del canvas
- Las dimensiones máximas dependen de cada canvas

## Sistema de coordenadas lógico

- Será un sistema de coordenadas gestionado por el programador de cada aplicación
- También lo llamaremos *sistema de coordenadas* (a secas)
- Serán coordenadas cartesianas, la más usadas en física y matemáticas: el valor 0 para la coordenada y se corresponde con la parte inferior de la pantalla
- Las dimensiones de este sistema (valores máximos y mínimos) las adaptaremos a lo que nos resulte conveniente para cada problema en particular

# Dimensiones del sistema (lógico) de coordenadas

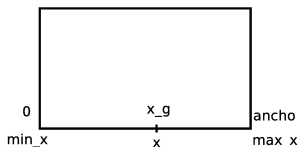
En cada problema, los valores máximos y mínimos de  $x$  e  $y$  tomarán los valores más convenientes. Ejemplos:

- En un gráfico sobre trigonometría,  $x$  e  $y$  podrán tomar valores entre  $-1$  y  $1$ , con el  $0,0$  en el centro de la pantalla
- Si vamos a trazar porcentajes, puede ser convenientes dar a la coordenada  $y$  valores entre  $0$  y  $100$
- Para gráficos en general podemos optar por valores entre  $0$  y  $1000$ , (independientes del tamaño del canvas gráfico)

A continuación veremos cómo cambiar el sistema de coordenadas, esto es, cómo obtener las coordenadas gráficas a partir de las coordenadas lógicas

# Cambio de coordenadas: eje de abscisas

- Coordenada horizontal (lógica)  
 $x$  toma valores entre  $min\_x$  y  $max\_x$
- Coordenada gráfica horizontal  
 $x\_g$  toma valores entre 0 y  $ancho$



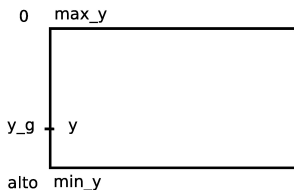
Observamos que  $\frac{x\_g}{ancho} = \frac{x - min\_x}{max\_x - min\_x}$

Por tanto, para obtener  $x\_g$  basta despejar

$$x\_g = \frac{x - min\_x}{max\_x - min\_x} ancho$$

# Cambio de coordenadas: eje de ordenadas

- Coordenada vertical (lógica)  
 $y$  toma valores entre  $min\_y$  y  $max\_y$
- Coordenada gráfica vertical  
 $y\_g$  toma valores entre 0 y  $alto$



Es análogo al caso anterior, solo que en las coordenadas gráficas el 0 está arriba y en las coordenadas lógicas el 0 está abajo, con lo que la expresión resulta

$$y\_g = \frac{max\_y - y}{max\_y - min\_y} * alto$$

# Corrección de la relación de aspecto

Además de los cálculos que acabamos de ver, posiblemente tendremos que corregir la *relación de aspecto* (*aspect ratio*)

[https://es.wikipedia.org/wiki/Relaci%C3%B3n\\_de\\_aspecto](https://es.wikipedia.org/wiki/Relaci%C3%B3n_de_aspecto)

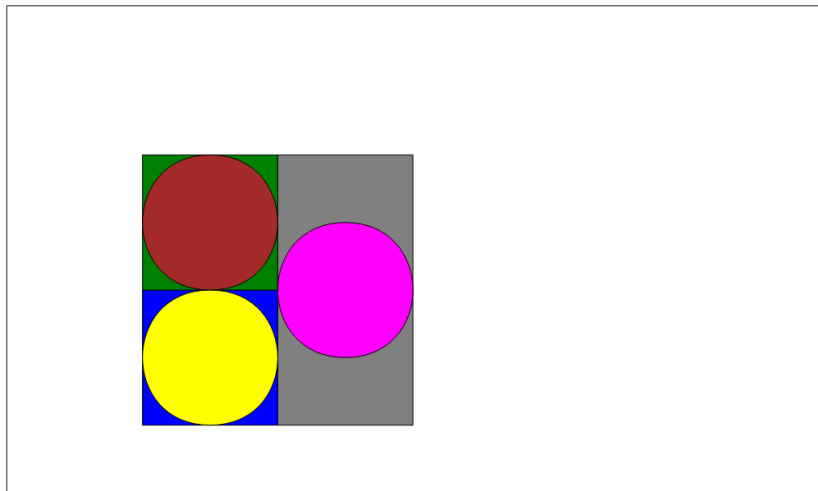
De lo contrario, los círculos tendrán forma de melón y los cuadrados serán rectangulares

- Si el ratio en los sistemas de coordenadas gráfico y lógico es el mismo, esto no es necesario
- Aunque sea diferente, para ciertos problemas puede que tampoco lo necesitemos. Por ejemplo un gráfico de barras o una nube de puntos

Para las prácticas de esta asignatura usaremos la vjCanvas

- Es una librería muy sencilla que permite trazar puntos, líneas, rectángulos y círculos, pero separando las coordenadas lógicas (virtuales) de las coordenadas gráficas del canvas HTML
- No es estándar, está hecha para esta asignatura
- Disponible en <http://ortuno.es/vjcanvas.js>

# Figuras básicas



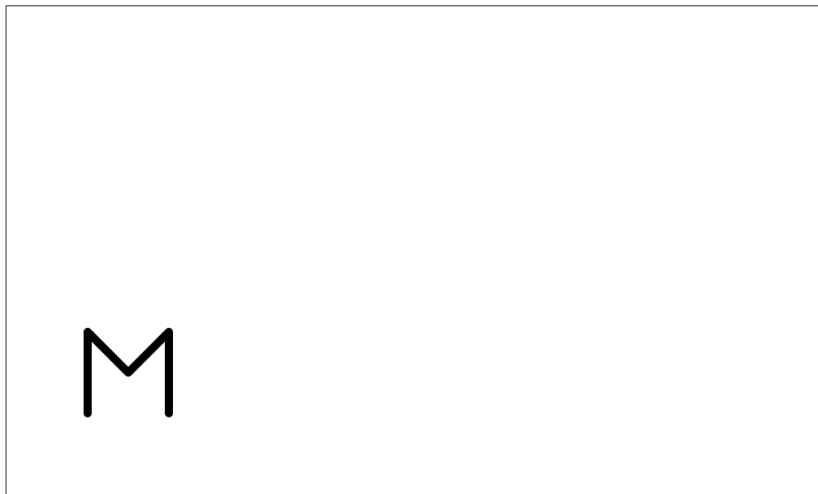
[http://ortuno.es/figuras\\_basicas.html](http://ortuno.es/figuras_basicas.html)

```
// Fijamos las coordenadas (lógicas)
let min_x = -200;
let max_x = 1000
let min_y = -100;
// max_y no lo especificamos, la librería lo calcula automáticamente
// para mantener las proporciones gráficas
vjcanvas.set_coords(min_x, max_x, min_y);

// Determinamos la posición y el tamaño de cada elemento
// (círculos y rectángulos)
let x, y, ancho, alto, diametro, color, color_borde, ancho_borde;
x = 0;
y = 0;
ancho = 200;
alto = 200;
color = "blue";
color_borde = "black";
ancho_borde = 1;
vjcanvas.rectangle(x, y, ancho, alto, color, color_borde, ancho_borde)
[...]
vjcanvas.circle(x, y, diametro, color, color_borde, ancho_borde);
[...]
```



# Líneas



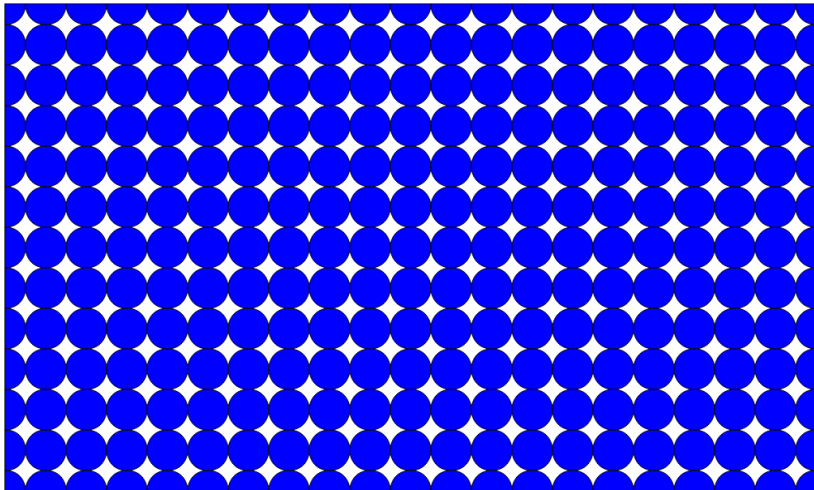
<http://ortuno.es/lineas.html>

```
let min_x = 0;
let max_x = 1000
let min_y = 0;
vjcanvas.set_coords(min_x, max_x, min_y);

let color = "black";
let ancho = 10;
let puntos = [
    [100,100],
    [100,200],
    [150,150],
    [200,200],
    [200,100],
];

vjcanvas.line(puntos, color, ancho);
```

# Mosaicos

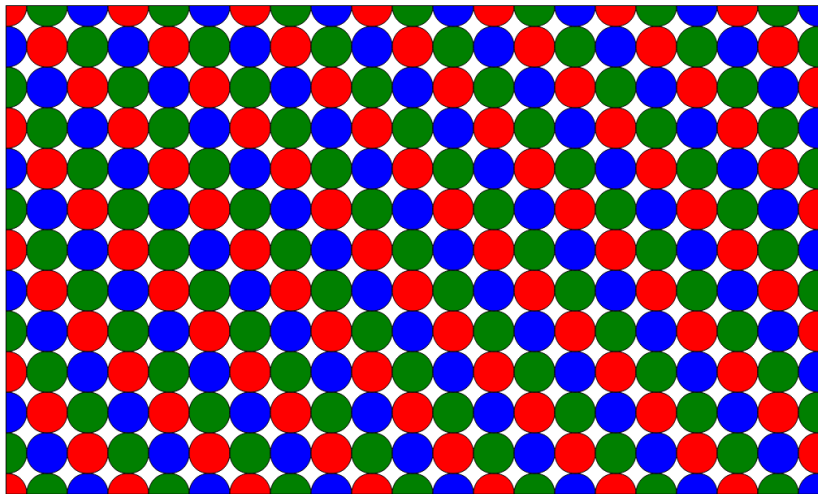


[http://ortuno.es/mosaico\\_01.html](http://ortuno.es/mosaico_01.html)

```
diametro = 50;
color = "blue";
for (let x=0; x <= 1000; x += diametro)
  for (let y=0; y <= 600; y += diametro) {
    vjcanvas.circle(x, y, diametro, color);
  }
```

En la librería vjcanvas

- Las coordenadas de los círculos siempre se refieren al centro
- Las coordenadas de los rectángulo siempre se refieren a la esquina inferior izquierda
- Los polígonos tienen el parámetro booleano *from\_center*, para indicar si las coordenadas son las del centro o las de la esquina

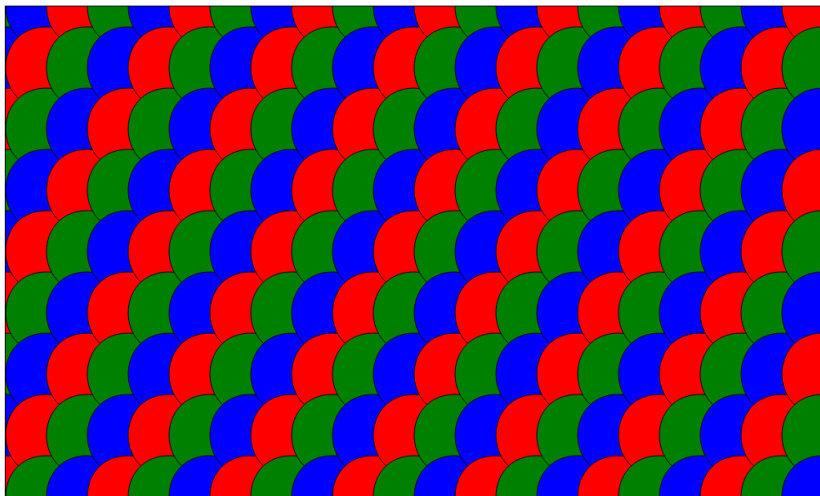


[http://ortuno.es/mosaico\\_02.html](http://ortuno.es/mosaico_02.html)

```
diametro = 50;
let colores = ['red', 'green', 'blue'];
let i_color = 0; // indice color
for (let x=0; x <= 1000; x += diametro)
  for (let y=0; y <= 600; y += diametro) {
    vjcanvas.circle(x, y, diametro, colores[i_color]);
    ++ i_color;
    if (i_color === colores.length)
      i_color = 0;
  }
```

También podemos usar colores en formato RGB o HSL

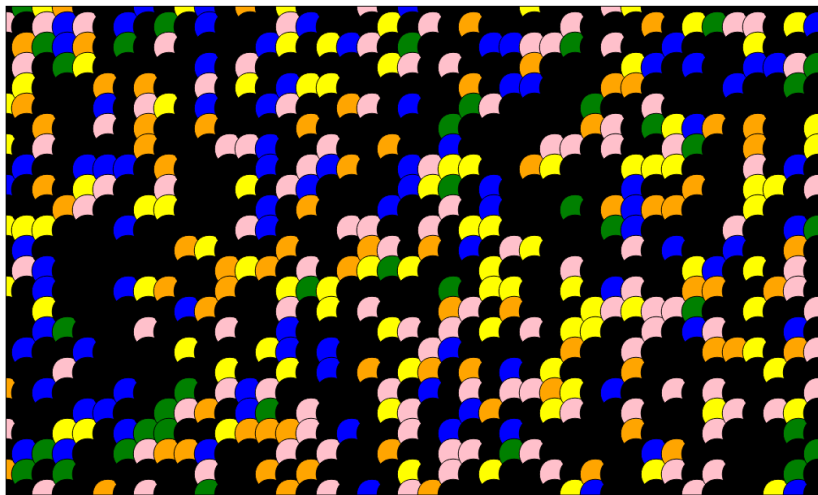
```
colores02 = ['rgb(0, 255, 255)', 'rgb(255, 0, 255)'];
colores03 = ['hsl(60, 100%, 50%)', 'hsl(0, 100%, 50%)'];
```



[http://ortuno.es/mosaico\\_03.html](http://ortuno.es/mosaico_03.html)

```
radio = 50;
let colores = ['red', 'green', 'blue'];
let i_color = 0; // indice color
for (let x=-50; x <= max_x; x += radio)
  for (let y=max_y; y >= -100; y -= radio*1.5) {
    vjcanvas.circle(x, y, radio * 2, colores[i_color]);
    ++ i_color;
    if (i_color === colores.length)
      i_color = 0;
  }
```

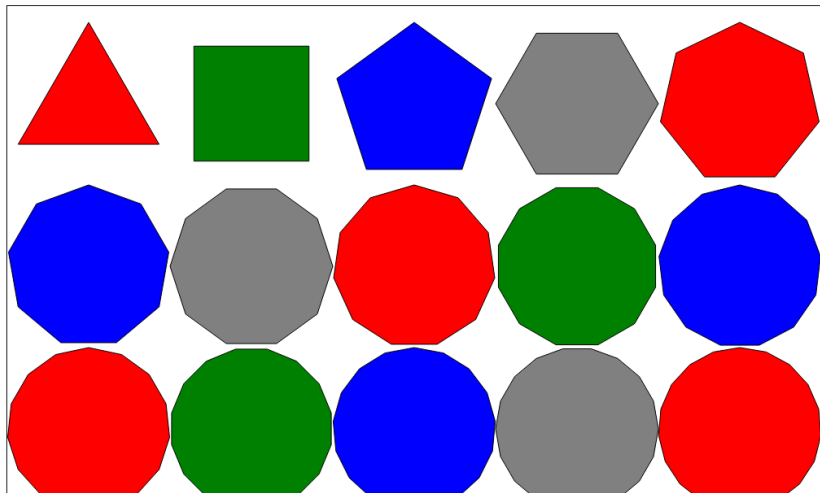




[http://ortuno.es/mosaico\\_04.html](http://ortuno.es/mosaico_04.html)

```
diametro = 25;
let colores;
colores = ['green', 'blue', 'pink', 'yellow', 'orange', 'black'];
// Repetimos el negro, para que predomine
colores = colores.concat(['black', 'black', 'black', 'black', 'black'])
console.log(colores);
let i_color = 0; // indice color
for (let x=-50; x <= max_x; x += diametro)
  for (let y=max_y; y >= -100; y -= diametro*1) {
    i_color = Math.floor(Math.random() * colores.length)
    vjcanvas.circle(x, y, diametro * 1.5, colores[i_color]);
  }
```

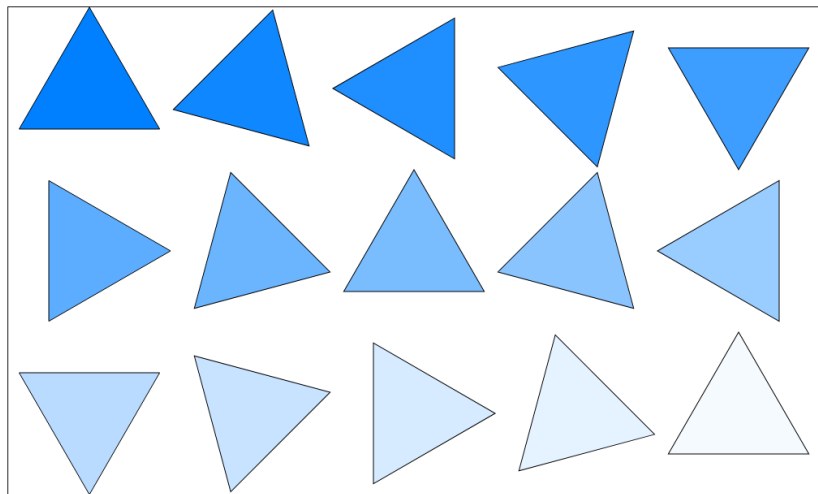
# Polígonos



<http://ortuno.es/poligonos.html>

```
let radio = 100;
let colores = ['red', 'green', 'blue', 'grey'];
let i_color = 0; // indice color
let lados = 3;
for (let y=580; y >= 0; y -= radio*2)
  for (let x=0; x <= 1000; x += radio*2){
    vjcanvas.polygon(x, y, lados, radio , colores[i_color]);
    i_color = i_color + 1;
    if (i_color === colores.length) {
      i_color = 0;
    }
    lados += 1;
  }
}
```

# Rotaciones y degradados

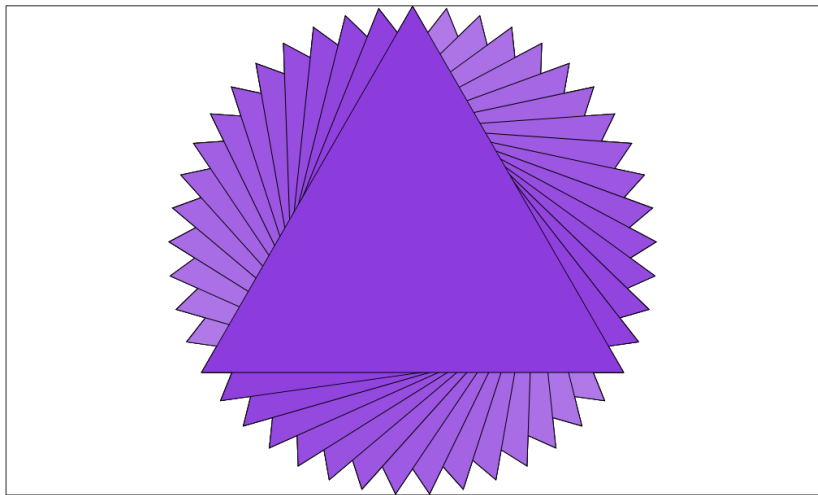


[http://ortuno.es/rotacion\\_degradado\\_01.html](http://ortuno.es/rotacion_degradado_01.html)

```
let radio = 100;
let lados = 3;
let color_borde = 'black';
let ancho_borde = 1;
let rotacion = 0;
let incremento_rotacion = 15;
let from_center = false;

let hue = 210; // Lo mantendremos constante
let saturation = 100; // Lo mantendremos constante
let lightness = 50; // Lo iremos incrementando
let incremento_lightness = 3;

for (let y=max_y; y >= 0; y -= radio*2)
  for (let x=0; x <= max_x; x += radio*2){
    color = vjcanvas.hsl_to_color(hue, saturation, lightness);
    vjcanvas.polygon(x, y, lados, radio , color, from_center,
      color_borde, ancho_borde, rotacion);
    rotacion += incremento_rotacion;
    lightness += incremento_lightness;
  }
```



[http://ortuno.es/rotacion\\_degradado\\_02.html](http://ortuno.es/rotacion_degradado_02.html)

```
let radio = 300;
let lados = 3;
let color_borde = 'black';
let ancho_borde = 1;
let from_center = true;
let rotacion = 0;
let incremento_rotacion = 4;

let hue = 270;
let saturation = 70;
let lightness = 100;
let incremento_lightness = -1.0;

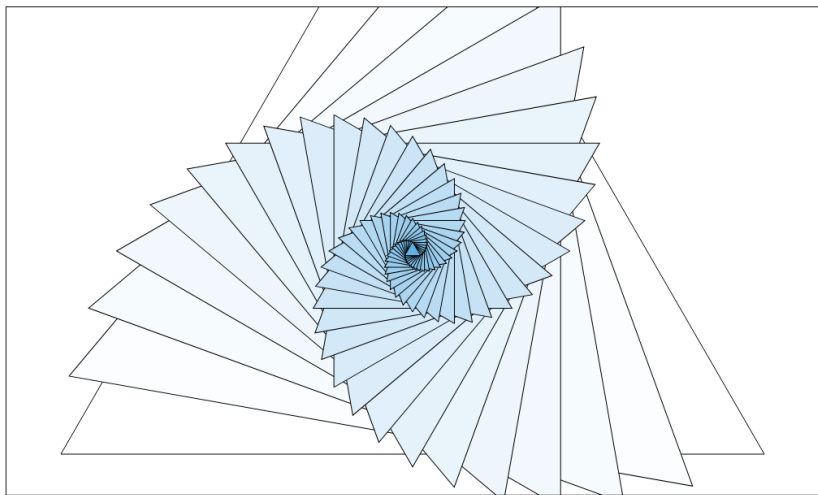
x = 500;
y = 300;

for (let rotacion=0; rotacion <= 360; rotacion += incremento_rotacion){
  color = vjcanvas.hsl_to_color(hue, saturation, lightness)

  vjcanvas.polygon(x, y, lados, radio , color, from_center,
    color_borde, ancho_borde, rotacion);

  lightness += incremento_lightness;
  rotacion += incremento_rotacion;
}
```





[http://ortuno.es/rotacion\\_degradado\\_03.html](http://ortuno.es/rotacion_degradado_03.html)

```
x = 500;
y = 300;
let radio = 500;
let lados = 3;
let color_borde = 'black';
let ancho_borde = 1;
let from_center = true;

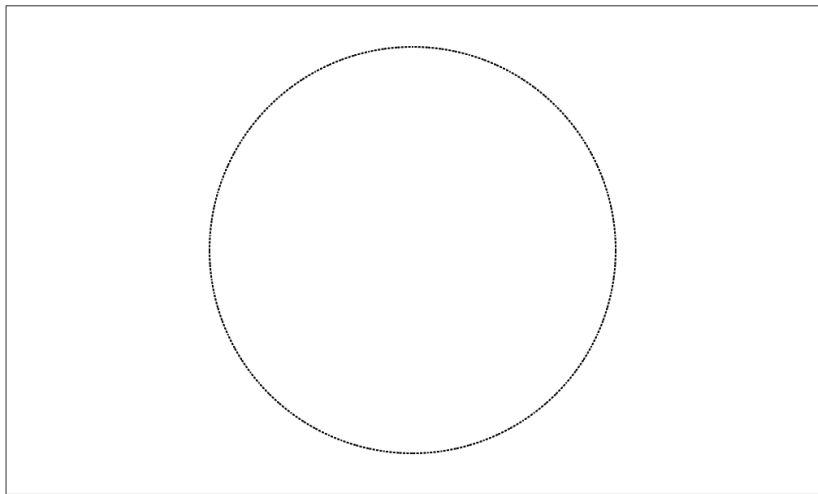
let rotacion = 0;
let incremento_rotacion = 5;
let hue = 204;
let saturation = 70;
let lightness = 100;
let incremento_lightness = -1.0;
let incremento_radio = .9;

for (let rotacion=0; rotacion <= 360; rotacion += incremento_rotacion){
  color = vjcanvas.hsl_to_color(hue, saturation, lightness)

  vjcanvas.polygon(x, y, lados, radio , color, from_center,
    color_borde, ancho_borde, rotacion);

  lightness += incremento_lightness;
  rotacion += incremento_rotacion;
  radio *= incremento_radio;
}
```

# Construcción de un círculo



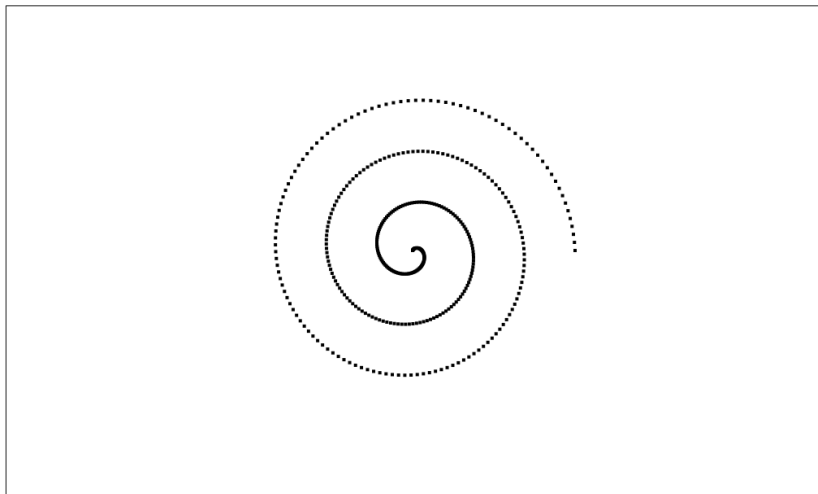
[http://ortuno.es/construccion\\_circulo.html](http://ortuno.es/construccion_circulo.html)

```
let min_x = -1;
let max_x = 1;
let min_y = -.6;
vjcanvas.set_coords(min_x, max_x, min_y);

let x, y, grosor, alto, radio, color;
color = "black"
radio = max_x / 2;
let incremento = .02;
grosor = 2;

for (let angulo = 0; angulo <= 2*Math.PI ; angulo = angulo + incremento){
  y = Math.sin(angulo) * radio;
  x = Math.cos(angulo) * radio;
  vjcanvas.dot(x, y, color, grosor);
};
```

# Espiral



<http://ortuno.es/espiral.html>

```
let incremento_angulo = .05;
let incremento_radio = .001;
let grosor = 4;
let vueltas = 5;
let x0 = .5;
let y0 = .5;

for (let angulo = 0; angulo <= 2*Math.PI * vueltas ;
    angulo = angulo + incremento_angulo){
  y = Math.sin(angulo) * radio;
  x = Math.cos(angulo) * radio;
  vjcanvas.dot(x, y, color, grosor) ;
  if (radio > 0){
    radio -= incremento_radio;
  };
};
```

# Espiral de color



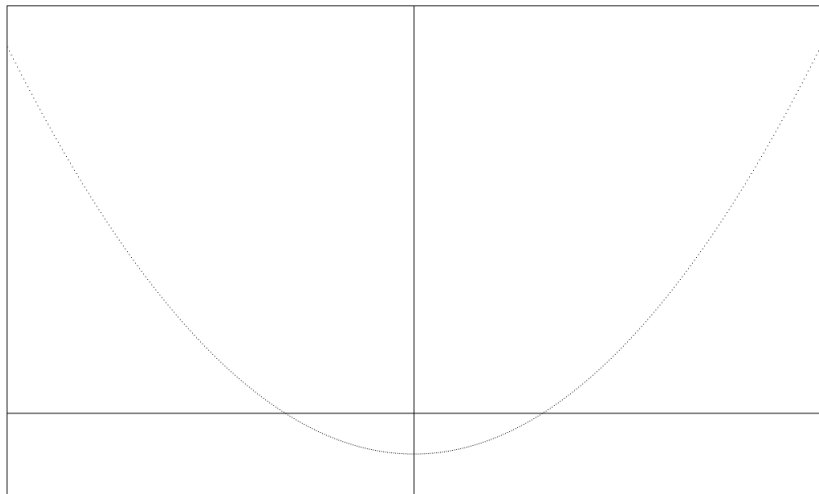
[http://ortuno.es/espiral\\_color.html](http://ortuno.es/espiral_color.html)

```
let radio = .05; // De la espiral
let radio_maximo = 1;
let color ;
let diametro_circulo = .3;
let x, y;
let color_borde = 'black';
let grosor_borde = 0;
let angulo = 0;
let incremento_angulo = 0.05;
let incremento_radio = 0.001;
let tamano = 90;
let vueltas = 12;

for (let angulo = 0; angulo <= 2*Math.PI * vueltas ;
    angulo = angulo + incremento_angulo){
  y = Math.sin(-1 * angulo) * radio;
  x = Math.cos(-1 * angulo) * radio;
  color = vjcanvas.random_color();
  vjcanvas.circle( x, y, diametro_circulo, color,
    color_borde, grosor_borde );
  if (radio < radio_maximo ){
    radio += incremento_radio;
  };
};
```



# Funciones matemáticas



<http://ortuno.es/parabola.html>

Para estos ejemplos, ya no es conveniente ajustar el ratio de aspecto de los gráficos. No importa que las proporciones gráficas sean distintas a las lógicas

Añadimos al método `set_coords` el parámetro `correct_ratio` con el valor `false`

```
let vc = {}; // virtual coordinates. Objeto global
vc.min_x = -10;
vc.max_x = 10;
vc.min_y = -20;
vc.max_y = 100;

function dibuja_ejes(){
  let color = "black";
  let grosor = 1;
  let puntos;
  puntos = [ [vc.min_x,0], [vc.max_x, 0] ];
  vjcanvas.line(puntos, color, grosor);

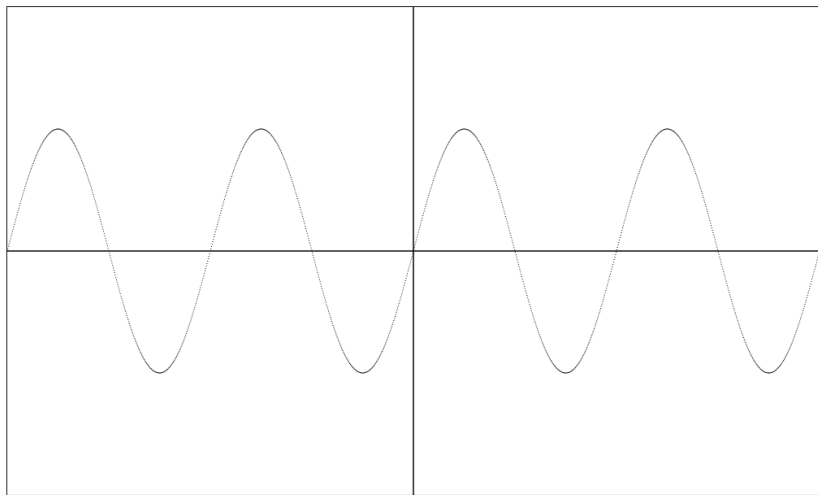
  puntos = [ [0,vc.min_y], [0, vc.max_y] ] ;
  vjcanvas.line(puntos, color, grosor);
  return;
}
```

```
function main(){
  let corrige_ratio = false;
  vjcanvas.set_coords(vc.min_x,vc.max_x, vc.min_y, vc.max_y, corrige_ratio);

  let color = "black"
  let grosor = 1;
  let y;
  let incremento_x = 0.05;
  let alpha = 1;

  dibuja_ejes();

  for(let x = vc.min_x; x<= vc.max_x; x += incremento_x){
    y = alpha * x**2 - 10;
    vjcanvas.dot(x, y, color, grosor);
  }
};
```



<http://ortuno.es/seno.html>

```
function main(){
  let corrige_ratio = false;
  vjcanvas.set_coords(vc.min_x,vc.max_x, vc.min_y, vc.max_y, corrige_ratio);

  let color = "black"
  let grosor = 1;
  let y;
  let incremento_x = 0.02;

  dibuja_ejes();

  for(let x = vc.min_x; x<= vc.max_x; x += incremento_x){
    y = Math.sin(x) ;
    vjcanvas.dot(x, y ,color, grosor);
  }
};
```

# Depuración

Por la naturaleza de estos programas, no tiene sentido probarlos en node.js, es necesario desarrollar directamente sobre el navegador.

Para depurar:

- Presta atención a los mensajes que pueda mostrar el motor de JavaScript en la consola
  - Atajo en Chrome: `Ctrl Shift I`
- Puedes trazar los distintos parámetros para comprobar que tengan valores *razonables*