Introducción a la Computación Gráfica

Ing. Gabriel Ávila, MSc.

Quiz

Discusión

Que es SVG?

Qué otros formatos?

Y en Processing?

Formato SVG
Scalable Vector Graphics

Especificación abierta que define un formato de imágenes vectoriales, estáticas y/o dinámicas.

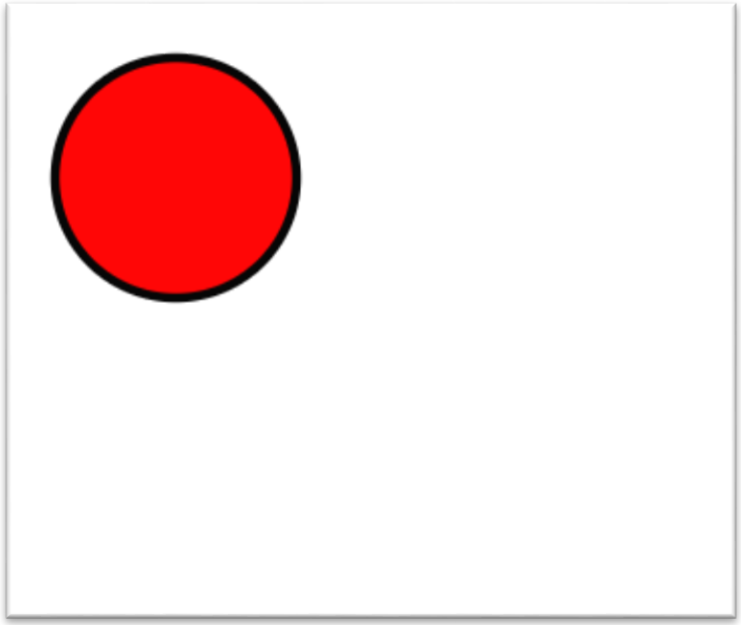
Funciona en diferentes navegadores de forma nativa.

La especificación permite hacer uso de:

- Elementos vectoriales.
- Mapas de bits.
- Texto.

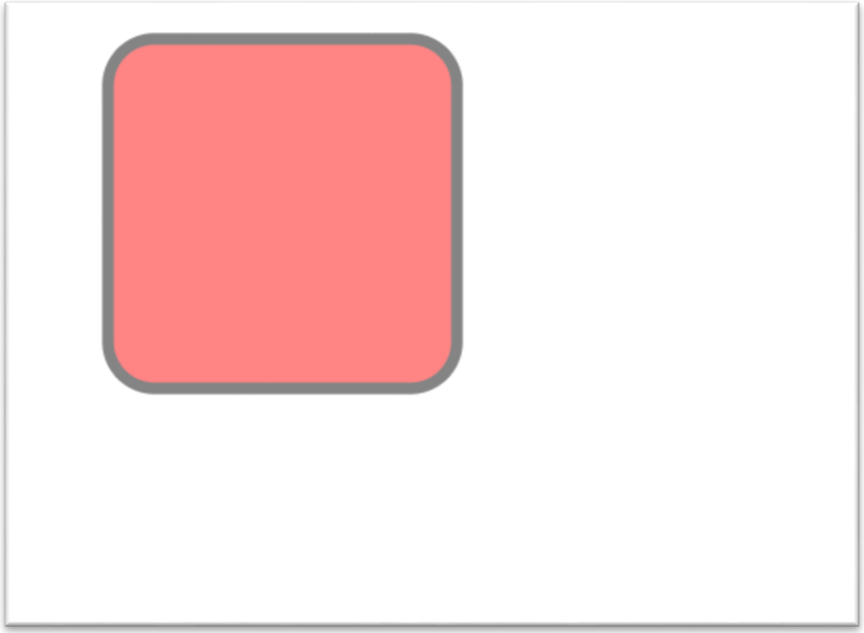
Ejemplos SVG

```
<!DOCTYPE html>
<html>
<body>

<svg height="100" width="100">
  <circle cx="50" cy="50" r="40"
stroke="black" stroke-width="3" fill="red"
/>
  Sorry, your browser does not support
inline SVG.
</svg>

</body>
</html>
```



Ejemplos SVG

```
<!DOCTYPE html>
<html>
<body>

<svg width="400" height="180">
  <rect x="50" y="20" rx="20" ry="20"
width="150" height="150"
style="fill:red;stroke:black;stroke-
width:5;opacity:0.5" />
  Sorry, your browser does not support
inline SVG.
</svg>

</body>
</html>
```



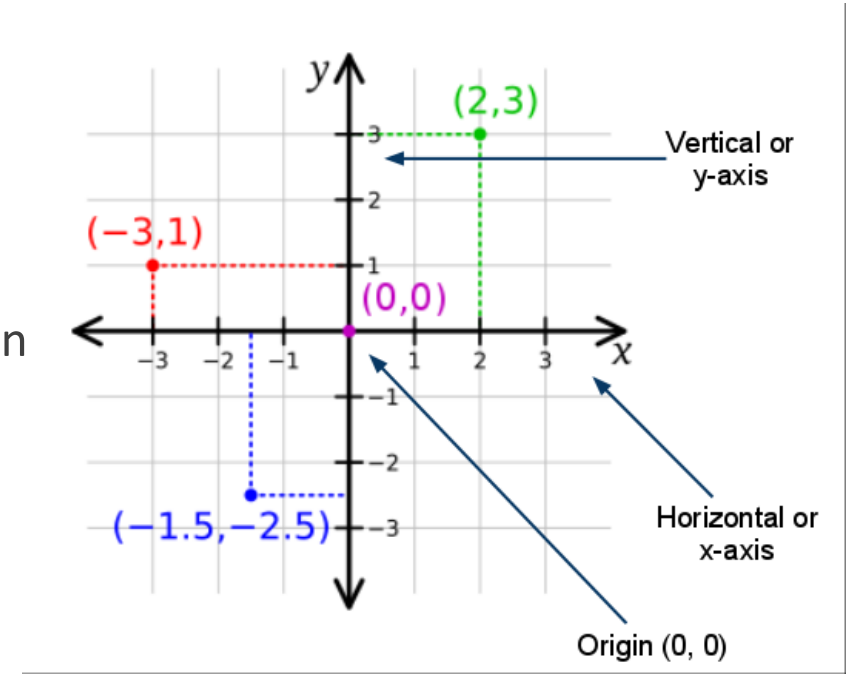
Sistemas de coordenadas

Coordenadas cartesianas 2D

También conocidas como coordenadas rectangulares.

Elementos importantes:

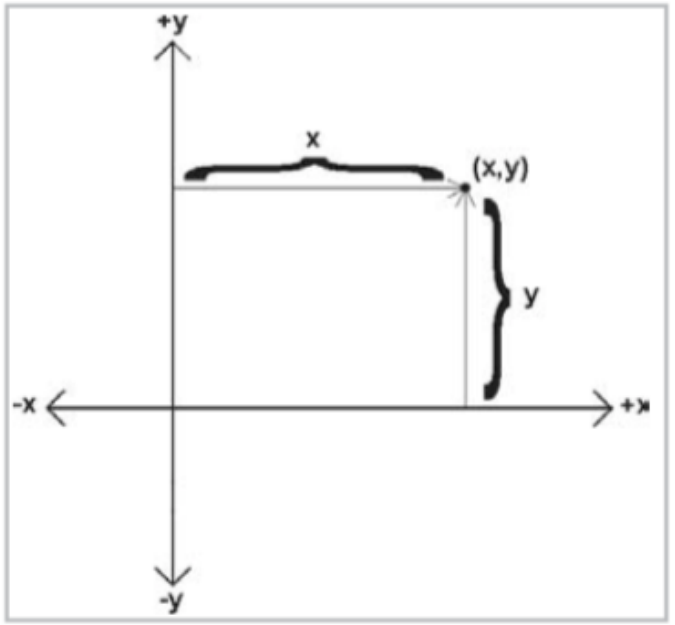
- ***Origen***.
- Dos ***ejes*** que pasan por el origen y son perpendiculares entre sí.



Coordenadas cartesianas 2D

Elementos importantes:

- 2 números (x, y) permiten obtener la ubicación de un punto en particular.
- Cada número representa la distancia a uno de los ejes, medida de forma paralela al otro eje.
- El signo determina la ubicación, positiva o negativa en el plano.



Coordenadas polares

Sistema bidimensional de coordenadas, en el que un punto puede ser representado mediante una distancia, tomada desde el origen, y un ángulo.

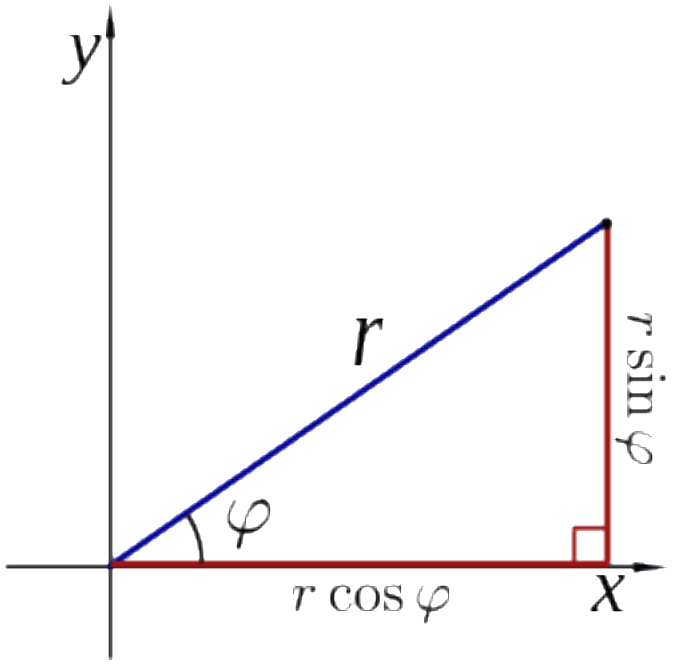
La relación entre los sistemas de coordenadas está dada por:

Cartesiano a polar:

$$\boldsymbol{r = \sqrt{x^2 + y^2}} \quad , \quad \boldsymbol{\theta = \tan^{-1}\left(\frac{y}{x}\right)}$$

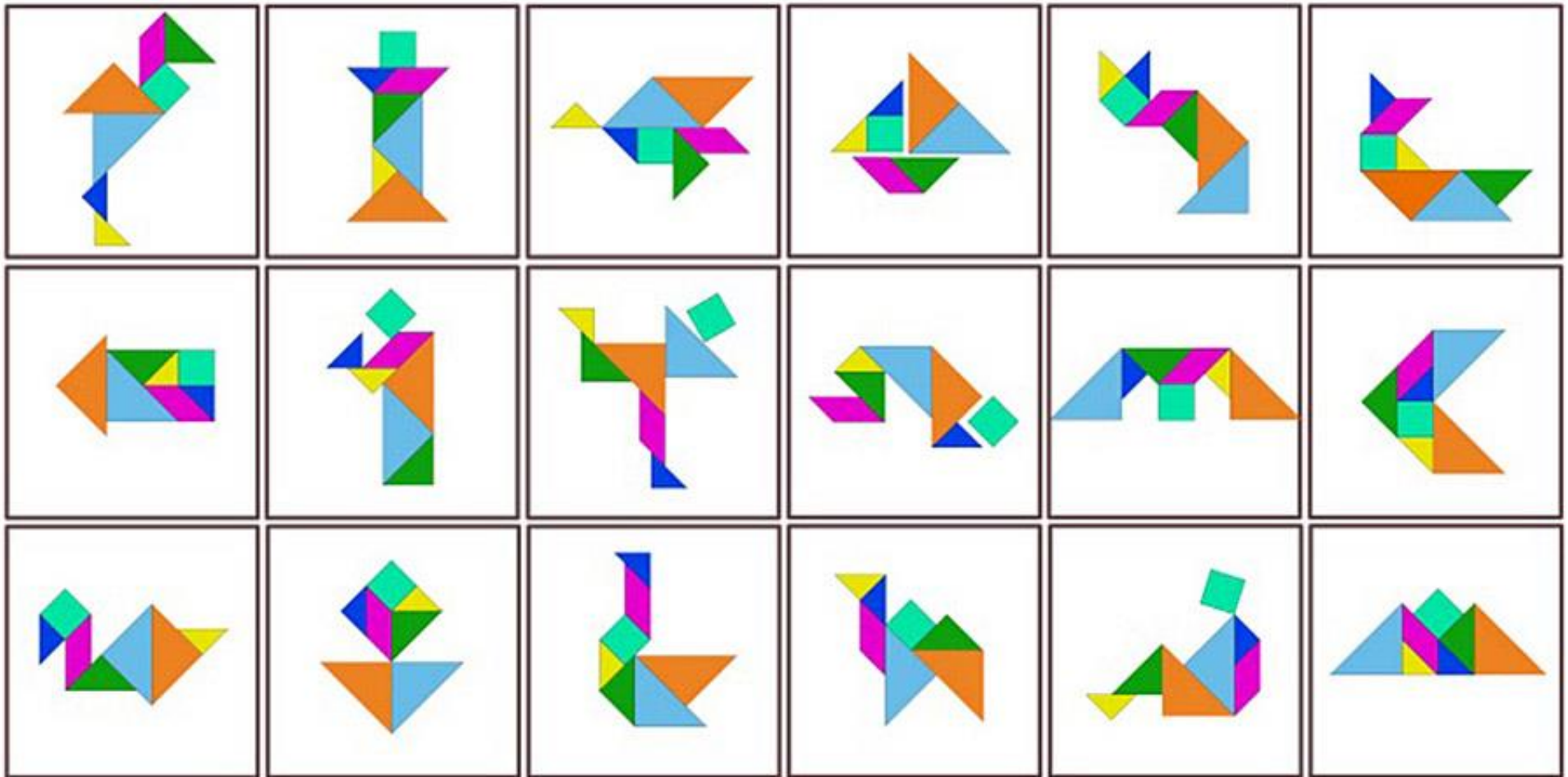
Polar a cartesiano:

$$\boldsymbol{x = r\cos\theta} \quad , \quad \boldsymbol{y = r\sin\theta}$$



Ejercicio

Usando los ejemplos sobre SVG en W3Schools, escoger y replicar:

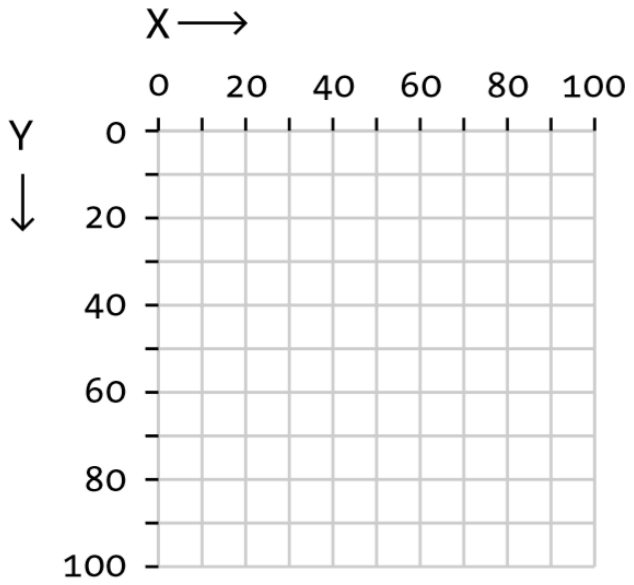


Processing

DE LA TEORÍA A LA PRÁCTICA

Coordenadas 2D en pantalla

La pantalla de visualización usualmente maneja un sistema de coordenadas cuyo origen se ubica en la esquina superior izquierda.



El tamaño de la pantalla se define con el comando:

```
size(width, height)
```

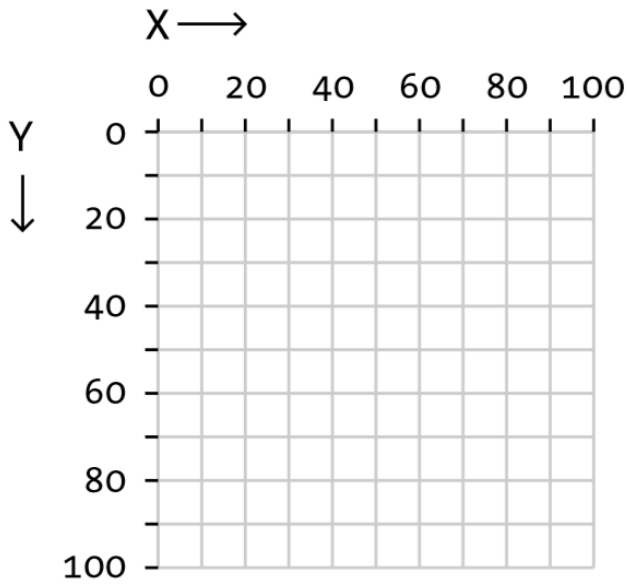
Las variables globales width y height almacenan los valores usados en la función size().

Si no se especifica size, por defecto width y height valen 100.

Ejercicio

Ubique los siguientes puntos en el sistema de coordenadas de pantalla:

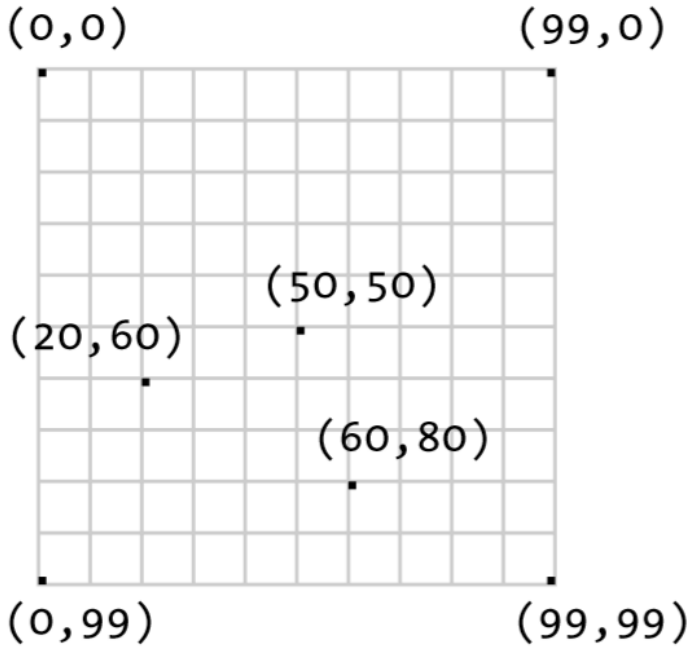
1. (0, 0)
2. (99, 0)
3. (20, 60)
4. (50, 50)
5. (60, 80)
6. (0, 99)
7. (99, 99)



Ejercicio

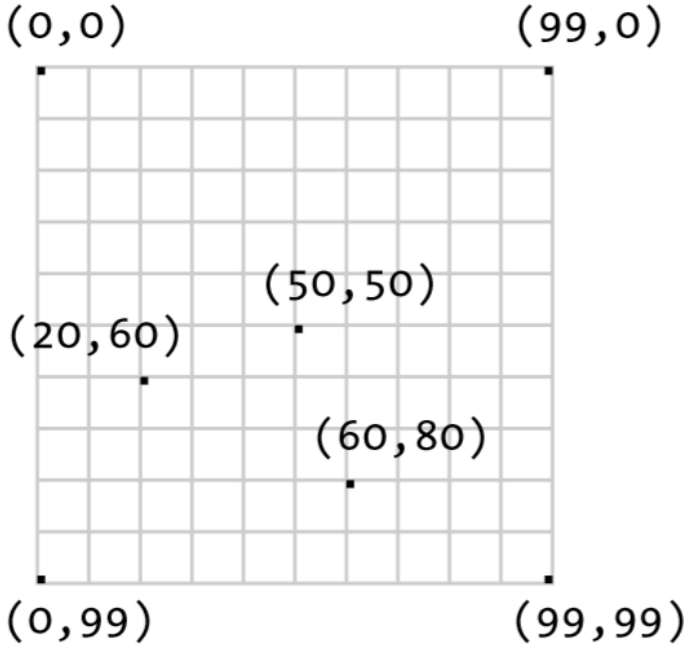
Ubique los siguientes puntos en el sistema de coordenadas de pantalla:

1. (0, 0)
2. (99, 0)
3. (20, 60)
4. (50, 50)
5. (60, 80)
6. (0, 99)
7. (99, 99)



Ejercicio

Convierta estas coordenadas a formato polar. Verifique el resultado convirtiendo a rectangular nuevamente.



Primitivas:

Point: Es la mas sencilla de las primitivas. Si no se especifica, su tamaño es un pixel.

```
point(posX, posY)
```

Line: Funciona indicando un punto de origen y uno de final.

```
line(posX1, posY1, posX2, posY2)
```

Triangle: Utiliza los parámetros x, y, de tres puntos

```
triangle(x1, y1, x2, y2, x3, y3)
```

Otras primitivas:

Quad: Utiliza los parámetros x, y, de cuatro puntos para graficar un cuadrilátero:

```
quad(x1, y1, x2, y2, x3, y3, x4, y4)
```

Rect: Utiliza un parámetro x, y, como origen además el ancho y alto del rectángulo:

```
rect(x, y, widht, height)
```

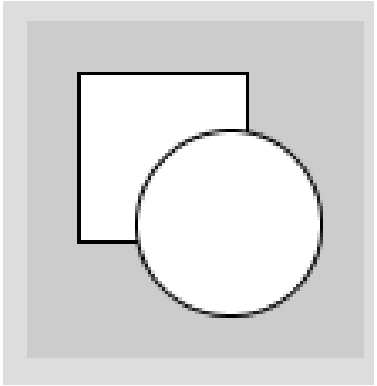
Ellipse: Utiliza los parámetros x, y, de cuatro puntos para graficar un cuadrilátero:

```
quad(x, y, widht, height)
```

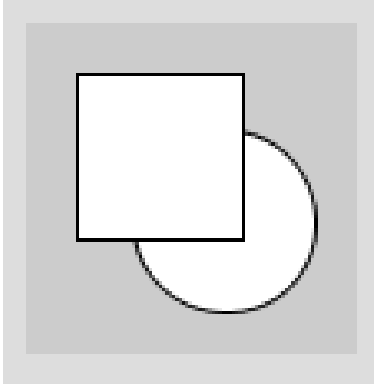
Orden de dibujado

El dibujo en pantalla se realiza secuencialmente, por lo tanto si un elemento se encuentra primero en el código, este quedará por detrás de otros elementos más recientes:

```
rect(15, 15, 50, 50);     // Bottom
ellipse(60, 60, 55, 55);  // Top
```



```
ellipse(60, 60, 55, 55);  // Bottom
rect(15, 15, 50, 50);     // Top
```



Fondo, relleno y bordes

Por defecto, el fondo en Processing es gris, las líneas negras y los rellenos blancos. Para modificar esto se puede usar:

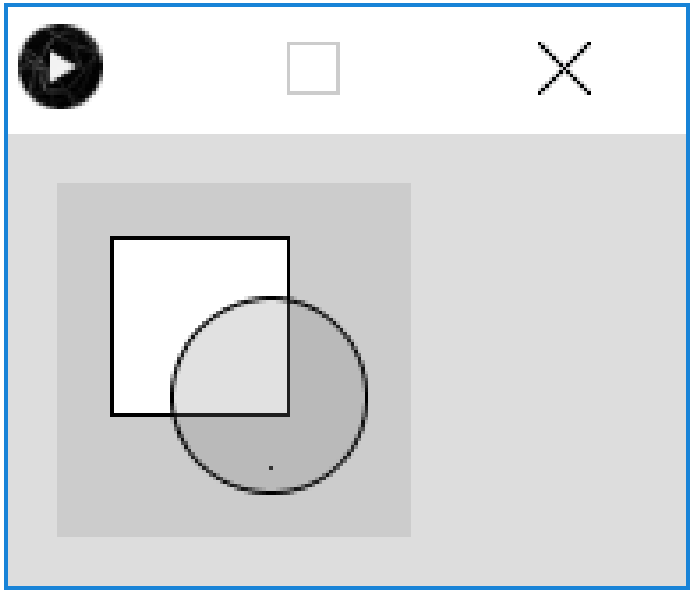
```
background(n)        //Cambia el fondo por un color
                     entre negro (0) y blanco (255).

fill(n)              //Cambia el relleno por un color
                     entre negro (0) y blanco (255).

stroke(n)            //Cambia el fondo por un color
                     entre negro (0) y blanco (255).
```

Una vez aplicado un cambio, todas las figuras tomarán esos valores para relleno y borde. Si no se desea tener relleno o línea de borde, se pueden usar las funciones `nofill()` y `nostroke()`.

Opacidad (Transparencia)

La función fill() puede tener dos parámetros, donde el primero define color y el segundo un nivel de transparencia: opaco (255) y transparente (0).

```
rect(15, 15, 50, 50);
fill(127, 60);
ellipse(60, 60, 55, 55);
```



Pensando en objetos

OBJECT ORIENTED PROGRAMMING

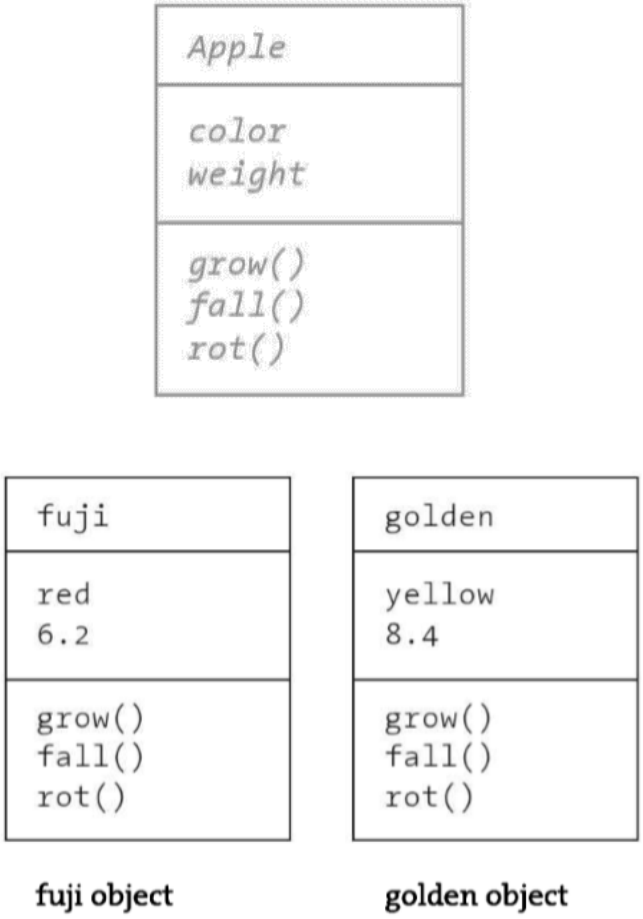
Recordando… Programación Orientada a Objetos

Técnica que permite formalizar las relaciones entre variables y funciones interconectadas.

Una **clase** define un grupo de estados (variables) e interfaces (métodos).

Un **objeto** es una instancia particular de una clase.

El paso inicial es el diseño del Tipo Abstracto de Dato (TAD), incluyendo su diagrama, así como los estados e interfaces necesarios.

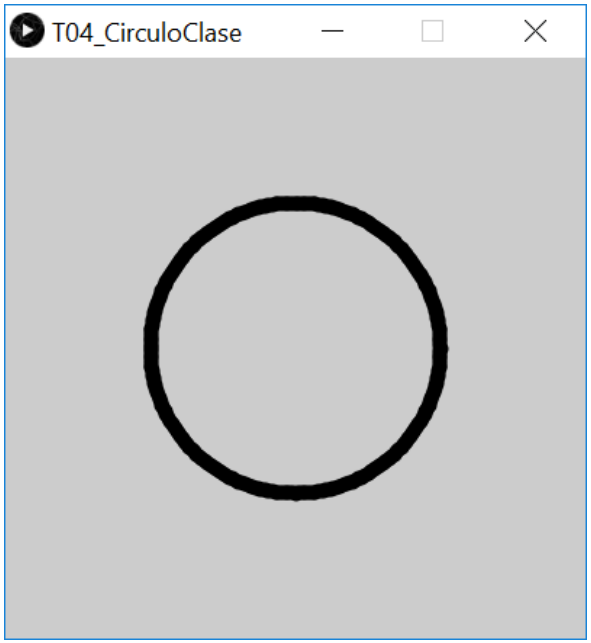


Clase Circulo

```
class Circulo{
    float cX; //Centro en X
    float cY; //Centro en Y
    float rX; //Radio en X
    float rY; //Radio en Y

    Circulo(float x, float y, float r){
        cX = x;
        cY = y;
        rX = rY = r;
    }

    void dibujar() {
        strokeWeight(10);
        for(int i=0; i<360; i++){
            point(cX+rX*cos(radians(i)), cY+rY*sin(radians(i)));
        }
        strokeWeight(1);
    }
}
```

Objeto: Instancia de la clase

```
void setup(){
  size(400, 400);
  float x = width/2.0;
  float y = height/2.0;
  float r = width/4.0;
  Circulo miCirculo = new Circulo(x, y, r);
  miCirculo.dibujar();
}
```



Ejercicio (Para entregar)

Realizar el ejercicio disponible en el Aula Virtual.

Referencias

[1] Tangram form. Tomado de: http://www.didatticarte.it/public/tangram_forme.jpg

[2] Reas, C. y Fry, B. (2007). Processing: A Programming handbook for visual designers. TheMIT Press. 1st Ed.