APUNTES PROCESSING

Contenido

INTRODUCCIÓN

Processing es un lenguaje de programación, derivado de java, dirigido a la creación de gráficos, compatible con los principales sistemas operativos. Es de código abierto, y desde la página https://processing.org/ podemos descargar el entorno de desarrollo que nos permitirá escribir el código, ejecutar y visualizar los programas creados.

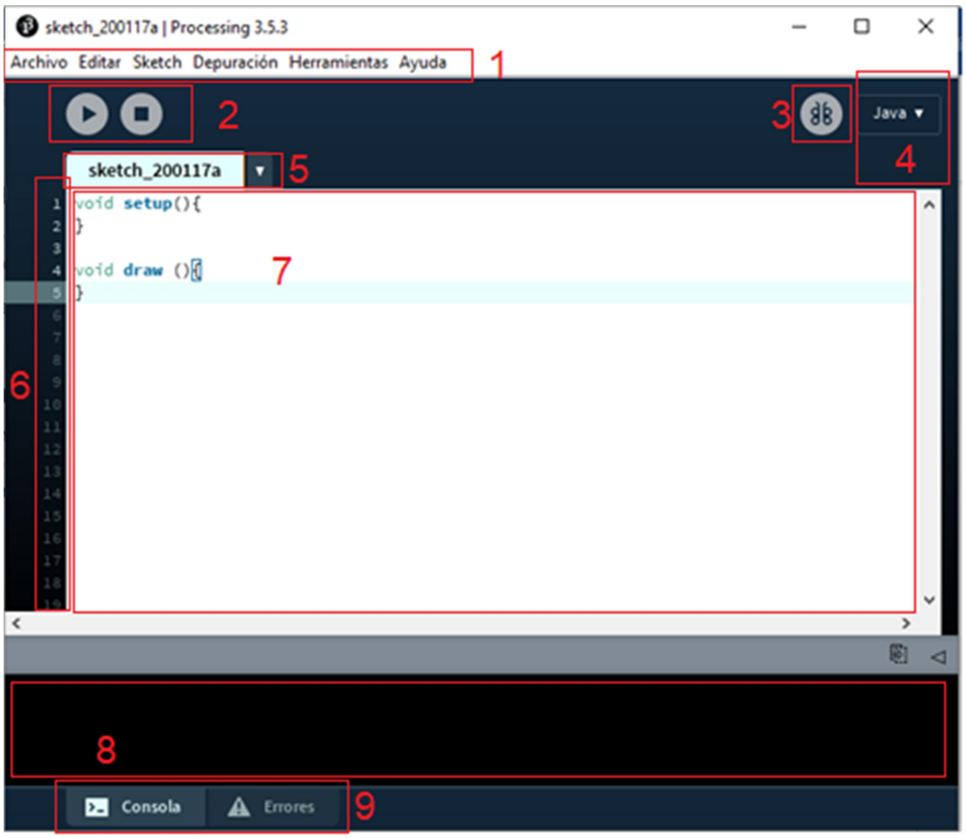
Es un lenguaje fácil de empezar a usar, pero muy potente, permite usar distintos tipos de programación como sentencia a sentencia, estructurada y orientada a objetos. Se puede combinar fácilmente con arduino y html.

NOCIONES BÁSICAS

Para empezar a programar debemos conocer como es el entorno de desarrollo, cómo se incluyen comentarios, cómo se usan las coordenadas, cuales son las funciones, órdenes, datos, y variables principales.

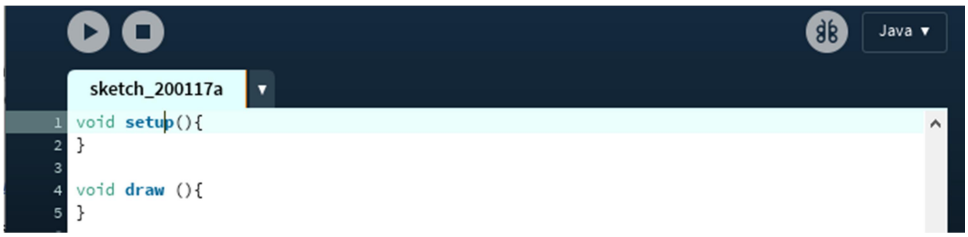
Partes del entorno de desarrollo (IDE)

1. Menú.
2. Botones para ejecutar y parar el programa.
3. Control de variables.
4. Modo (Java, R, Python...).
5. Nombre del archivo.
6. Números de líneas.
7. Zona de código.
8. Visor de mensajes.
9. Botones de consola y errores.



Funciones principales:

Un programa de processing va estar formado por dos funciones principales: **void setup()** donde se incluirán las sentencias que se quieran ejecutar una sola vez, y **void draw ()** que se ejecutará en bucle indefinidamente.



Comentarios

Para incluir comentarios en el código tenemos 2 formas según ocupen una o varias líneas.

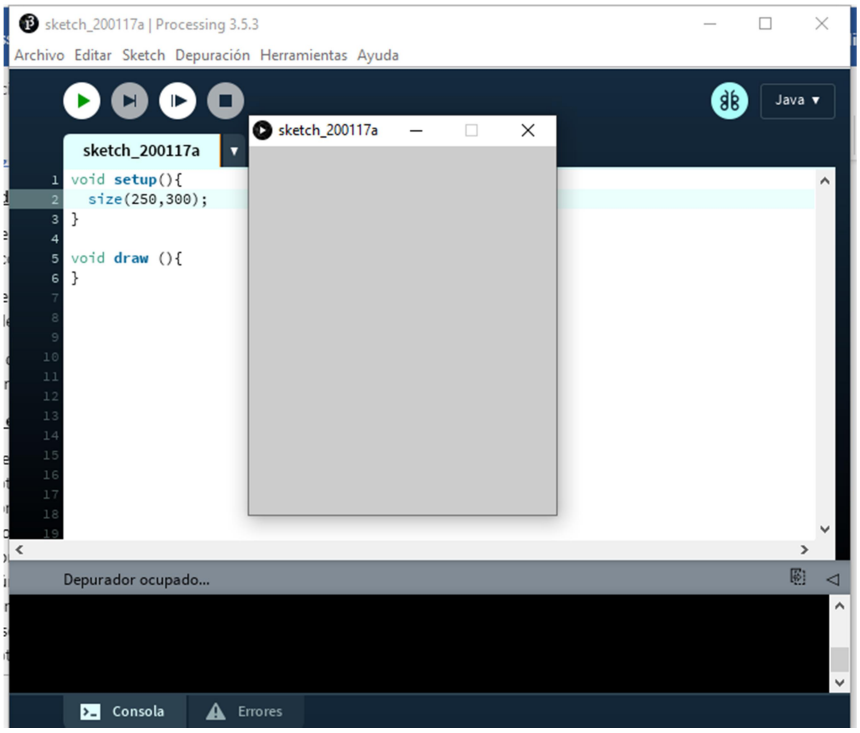
- // 1 línea.
- /* */ varias líneas.

Coordenadas:

Muchas de las sentencias que usemos para crear nuestros diseños, utilizarán coordenadas. Lo primero que debemos conocer es que el punto de origen es la esquina superior izquierda de la ventana.

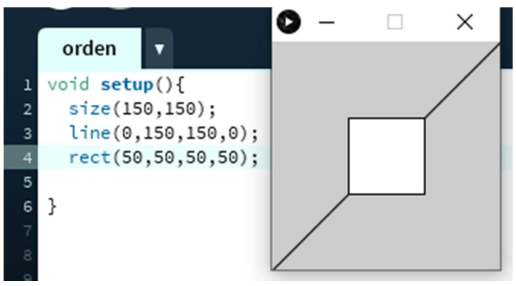
Las coordenadas indicarán la posición de un punto, indicando su distancia en pixeles desde el eje x y la distancia desde el eje y separados por una coma.

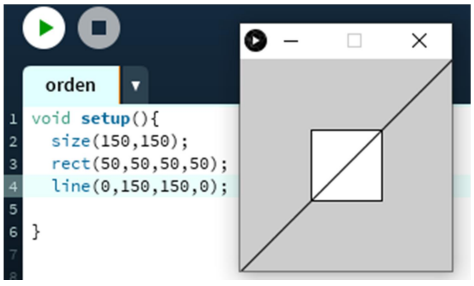
El tamaño del lienzo que vayamos a programar se indicará con la orden **size(x,y)**, donde "x" serán los pixeles que ocupará a lo ancho, e "y" los ocupados a lo alto. Este tamaño no limita los elementos, si no la parte visible, si un elemento es más grande que el lienzo se saldrá por los lados o incluso ni se verá, si su origen está en los bordes.



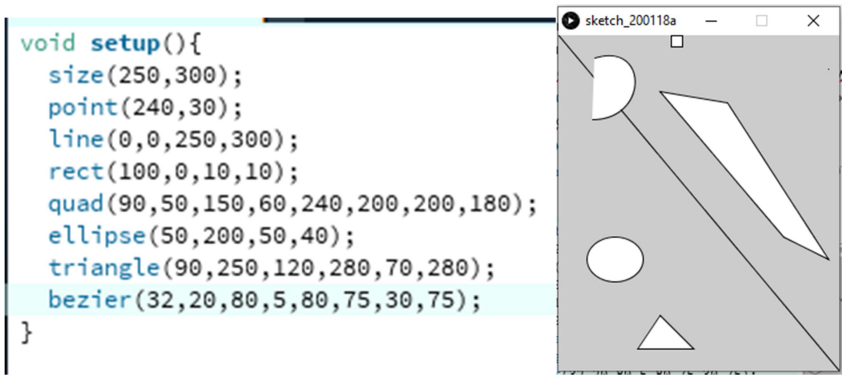
Orden:

Las sentencias se ejecutan en orden descendente. Este orden también marca la posición de los elementos en el lienzo, dibujándose unos sobre otros, quedando encima el último dibujado.

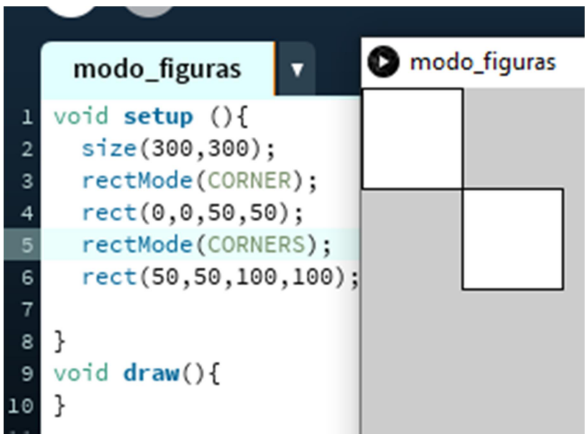




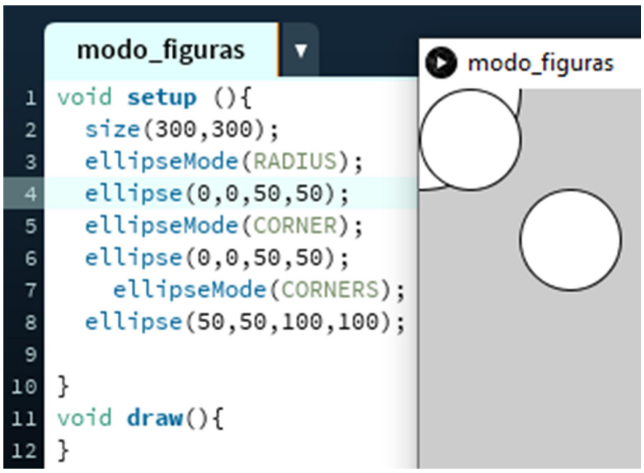
Formas básicas:



- **Punto:** **point(x,y);**posición en el eje x y posición en el eje y.
- **Línea**: **line(x_1,y_1,x_2,y_2);** Punto de inicio y punto de final.
- **Rectángulo**: o cuadradado **rect(x,y,ancho,alto);** punto de inicio (x,y) ancho y alto.
    - **rectMode();** admite los parámetros CORNER, para tomar como punto de origen la esquina superior izquierda en vez del centro que toma por defecto o con CENTER, y CORNERS, que convierte los valores de ancho y alto en el punto de la esquina opuesta origen (esquina inferior derecha)



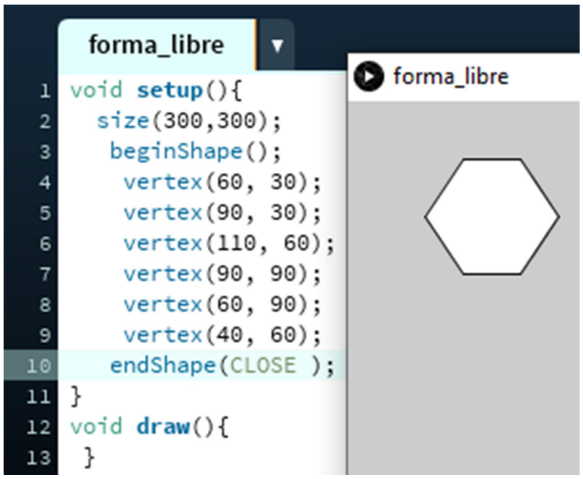
- **Cuadrilátero** **quad(x_1,y_1,x_2,y_2,x_3,y_3, x_4,y_4)** se indican los cuatro puntos que limitan el polígono.
- **Círculo**: o elipse, **ellipse**(x,y,ancho,alto);
    - **ellipseMode();** RADIUS toma origen el centro de la elipse pero los valores 3 y 4 indican sus radios no sus diámetros como CENTER o por defecto. Con CORNER cambia el origen de la elipse del centro a la esquina superior izquierda del rectángulo que la contendría, CORNERS toma el origen como CORNER y los valores 3 y 4 como la esquina opuesta.



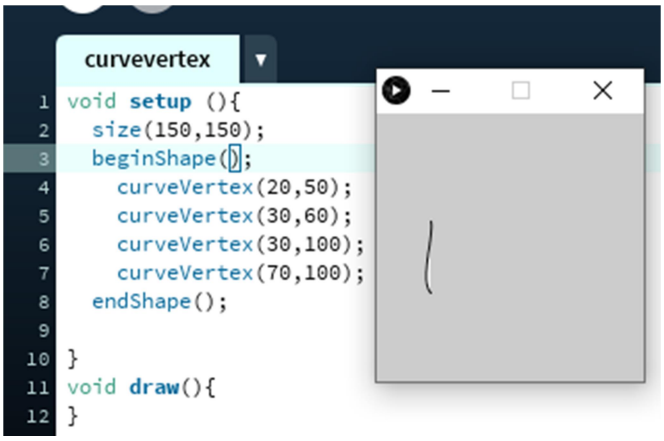
- **Triángulo**: **triangle**(x_1,y_1,x_2,y_2,x_3,y_3)
- **Curva Bezier**: **bezier**(x_1,y_1, cx_1,cy_1,cx_2,cy_2, x_2,y_2,) definida por los puntos de inicio y fin, y los puntos de control.

Forma libre:

Processing permite crear figuras a partir de sus vértices. Para ello usaremos el método **beginShape()** **endShape()**, entre los que indicaremos los vertices de la figura **vertex(x,y);.** Para que se muestre la forma contendia en los vertices usamos el parámetro CLOSE en **endShape();**

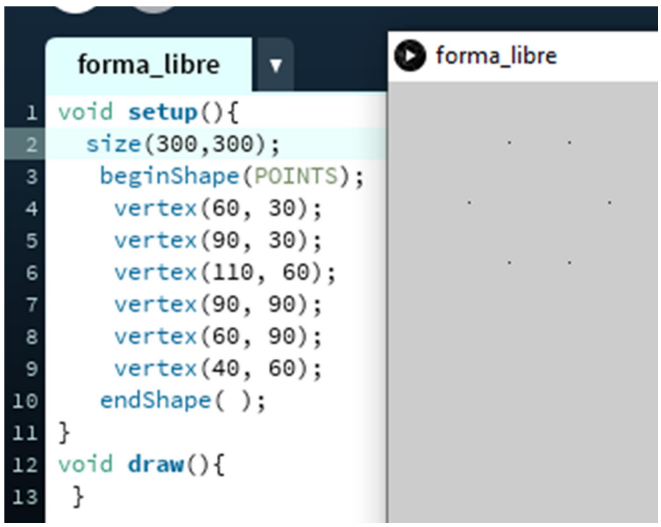


Si usamos **curveVertex();,** en vez de vertex, usaremos líneas curvas para limitar la figura creada. Tambien podemos usar la opción **bezierVertex();** para usar curvas tipo Bezier .

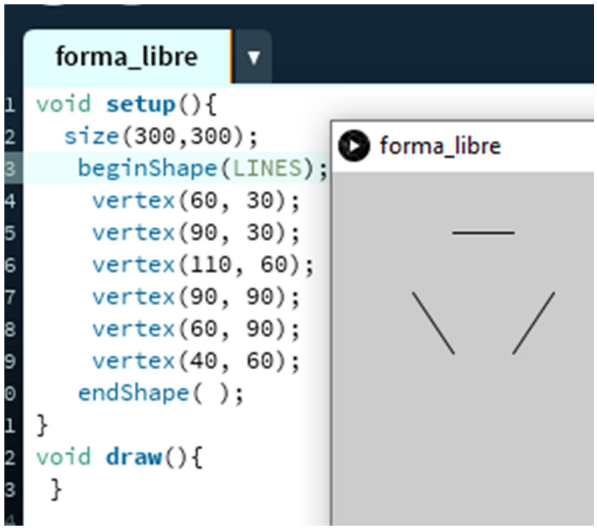


Este método beginShape() admite como paramétros:

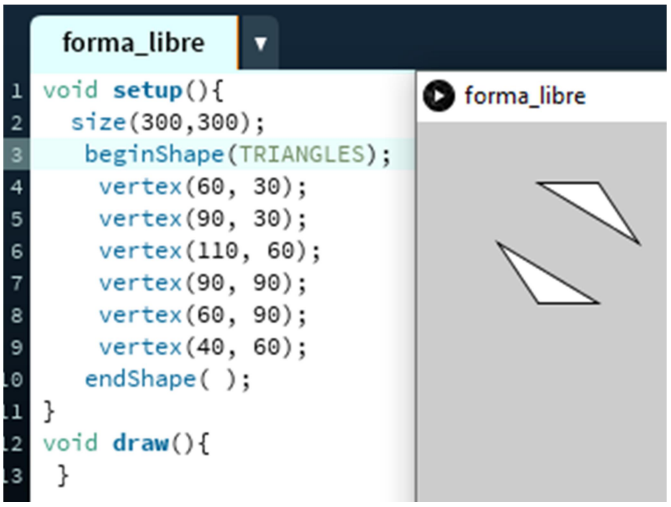
- POINTS, la forma está limitada por puntos.



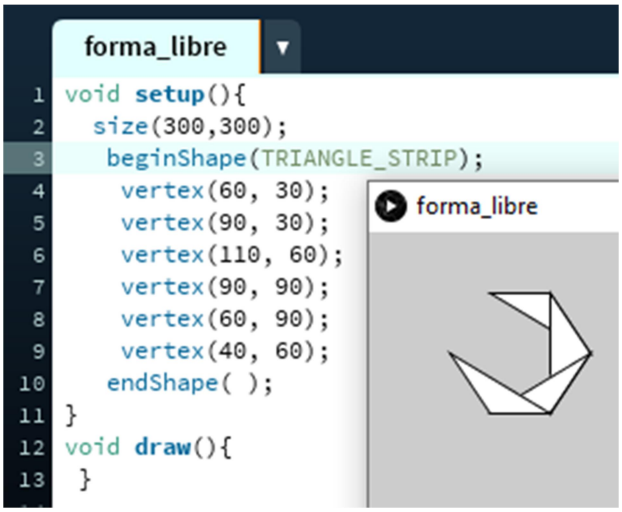
- LINES, la forma está límitada por líneas independientes (no comparten vertice).



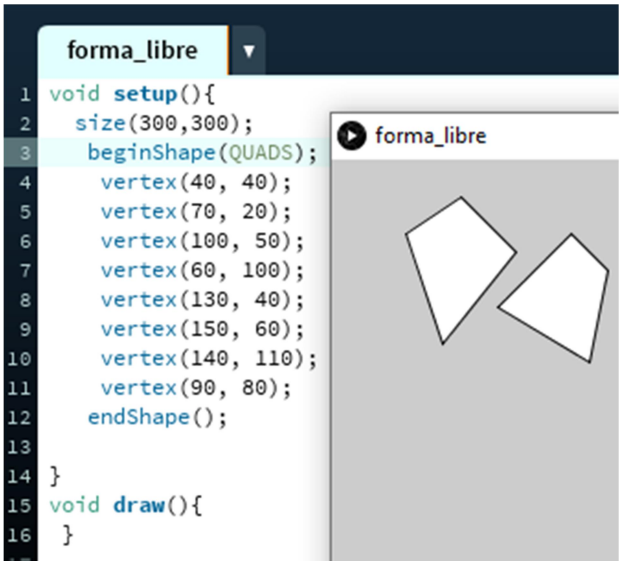
- TRIANGLES, los vertices se unen mediante triángulos independoientes.



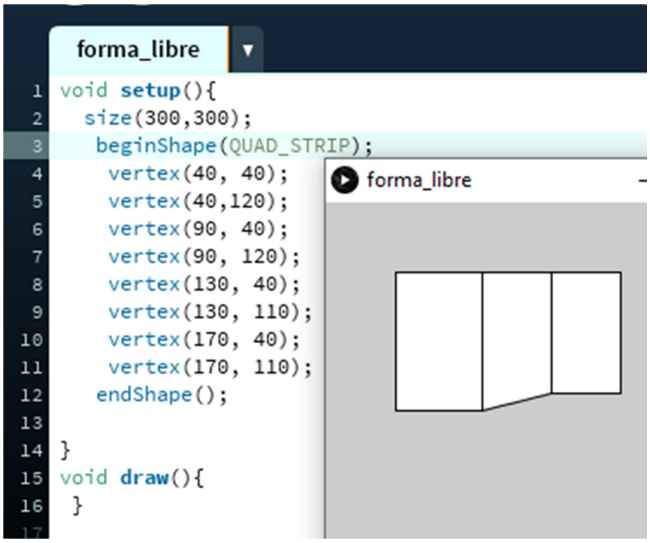
- TRIANGLE_STRIP, los vertices se unen mediante uan secuencia de triánguos unidos.



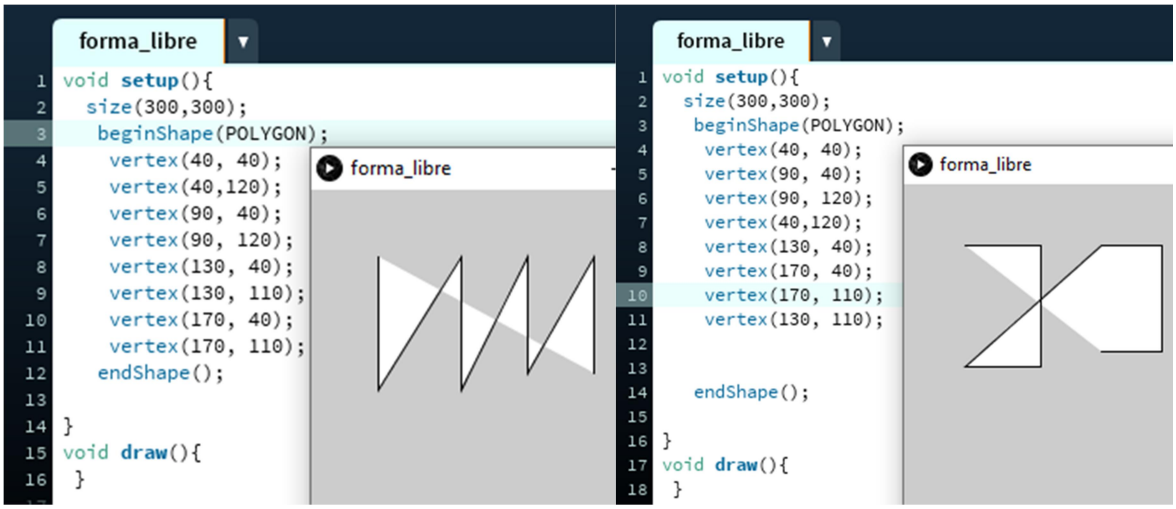
- QUADS, agrupa cada cuaterna de vertices de forma independienrte.



- QUAD_STRIP, Limita la forma mediantes rectandulos contiguos, aquí el el orden de los puntos varian, no los indicamos por como los dibujariamos con líneas sin saltar ningún punto. Los ordenamos de menor a mayor, primero por el valor de x y después por el de y.

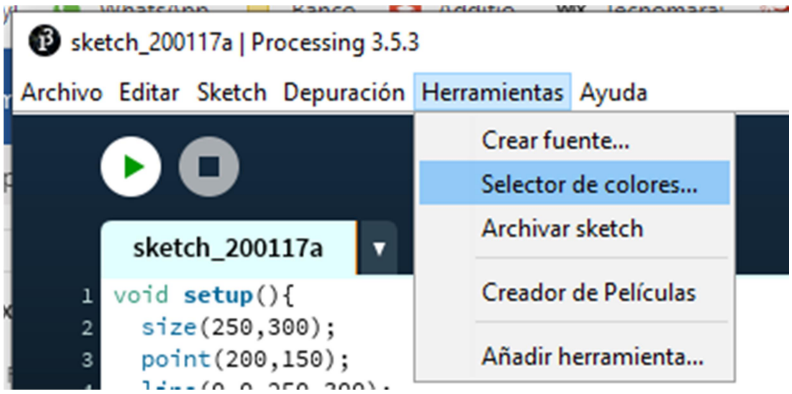


- POLYGON



Colores y acabados:

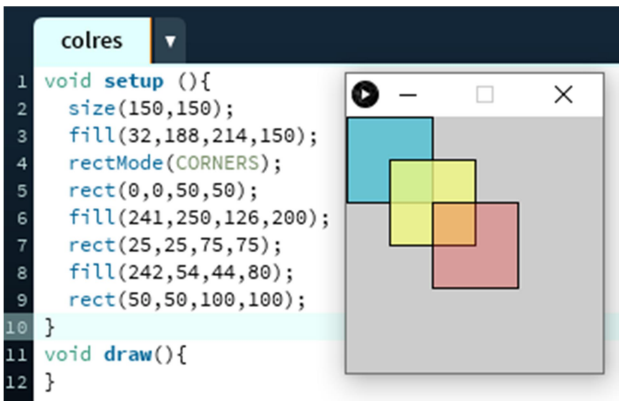
El color de los distintos componentes se puede indicar usando principalmente los códigos RGB (con y sin transparencia) y hexadecimal. El entorno de desarrollo incluye la herramienta "selector de color."



Dependiendo de a qué queramos darle color usaremos las siguientes órdenes.

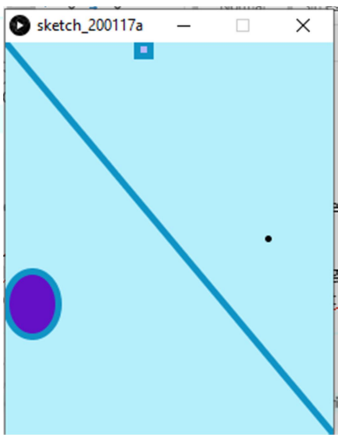
- **Fondo**: **background(código_color);**
- **Relleno de formas**: **fill (código_color);** antes de la forma o formas elegidas. **Nofill()** sin relleno.

Si superponemos figuras no opacas de diferentes colores podemos crear colores nuevos en las intersecciones.

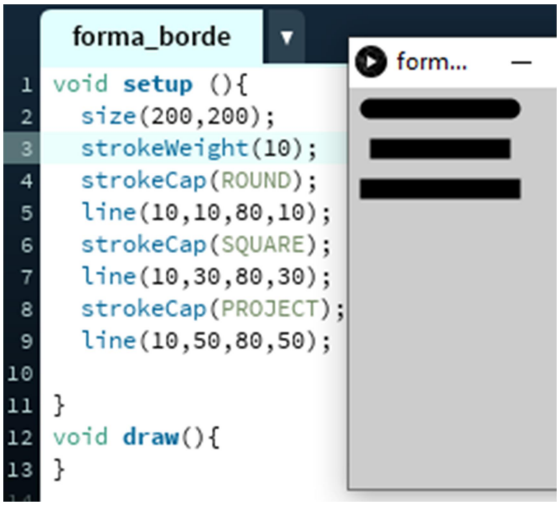


- **Borde**: **strokeWeight(grosor);** antes de la formas o formas elegidas. **Stroke(código_color); noStroke();** sin contorno.

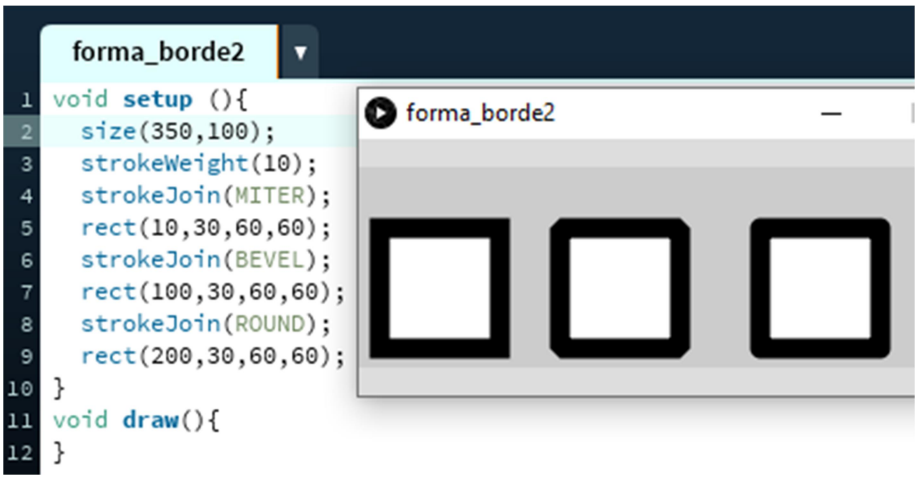
```
void setup(){
  size(250,300);//tamaño del lienzo
  background(#B5EFFC);//color del fondo
  strokeWeight(5);//borde de 5 px del punto, línea, cuadrado y elipse.
  point(200,150);//punto
  stroke(#1094C6);//color borde línea, cuadrado y elipse.
  line(0,0,250,300);//línea diagonal
  fill(#B5B8FC);//color del cuadrado
  rect(100,0,10,10);//cuadrado de 10x10
  fill(#6410C6);//color de la elipse
  ellipse(20,200,40,50);//elipse de 40 de ancho y 50 de alto
}
void draw(){
}
```



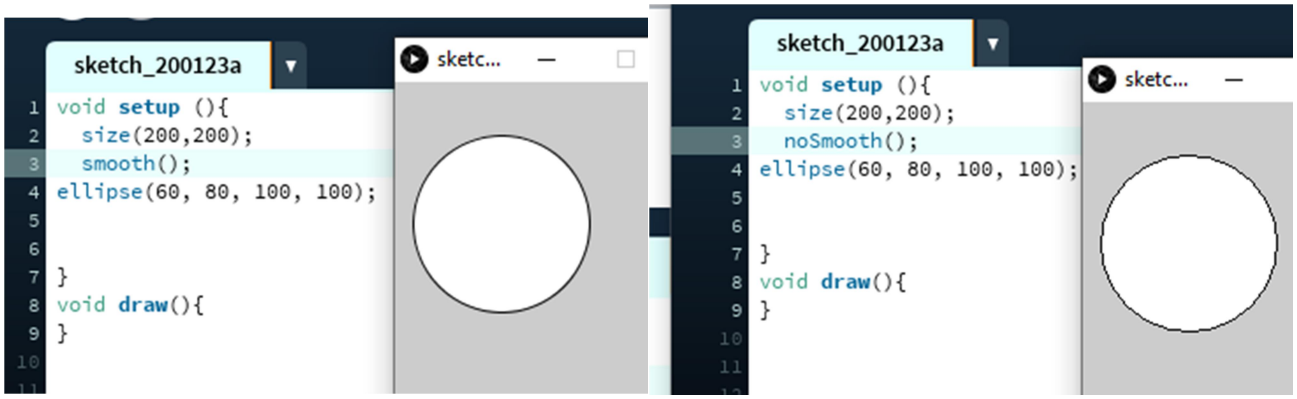
StrokeCap(); Acabado de los bordes, forma cuadrada (SQUARE), redondeada(ROUND) y termino medio (PROJECT)



strokeJoin(); configura el acabado de los vertices de una figura para que aparezcan en esquina (MITER, forma por defecto), corta recta la esquina (BEVEL), o esquina redondeada (ROUND)



Suavizar: **noSmooth**, muestra el contorno redondeado de las figuras como líneas y puntos, mientras que smooth, opción por defecto, no hace falta especificarlo, suaviza los contornos, mostrando una sola línea curva.



Datos:

En programación se usan distintos tipos de datos, que permiten ser usados en diferentes sentencias y operaciones. Los principales tipos de datos son:

- **Entero**: int
- **Reales**: float
- **Carácter:** char
- **Cadena de carácter:** string
- **Lógicos**: boolean

Operadores:

- **Asignación** =
- **Igualdad ==** distinto: **!=**
- **Mayor y menos que** >, <
- **Mayor o igual, menor o igual** >=, <=
- **Suma +** Incrementar en 1 ++
- **Resta –** reducir en 1 --
- **Multiplicación ***
- **División** cociente :/ resto: %
- **Lógicos** y: || o: && no: !

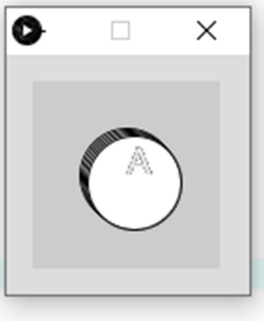
Variables

Processing permite utilizar variables locales, creadas dentro de una función que solo puede usarse en ella, o globales, creada fuera y que puede ser usada por todas las funciones.

Las variables se crean indicando el tipo de dato y el nombre de la variable. En el mismo momento se pueden iniciar asignándoles un valor.

```
float n=50; //Variable número real global inicializada
void setup(){
  int i; //variable número entero local sin inicializar
  for (i=0;i<5;i++){
    ellipse(n+i,n+i,50,50);
  }
  }
void draw(){
  fill(random(255));
  char j='A';//Variable caracter local con valor A
  textSize(20);
  text(j,n,n);

}
```

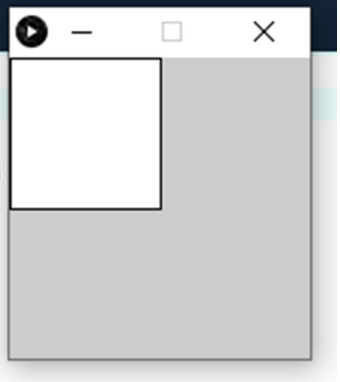


Variables de sistema:

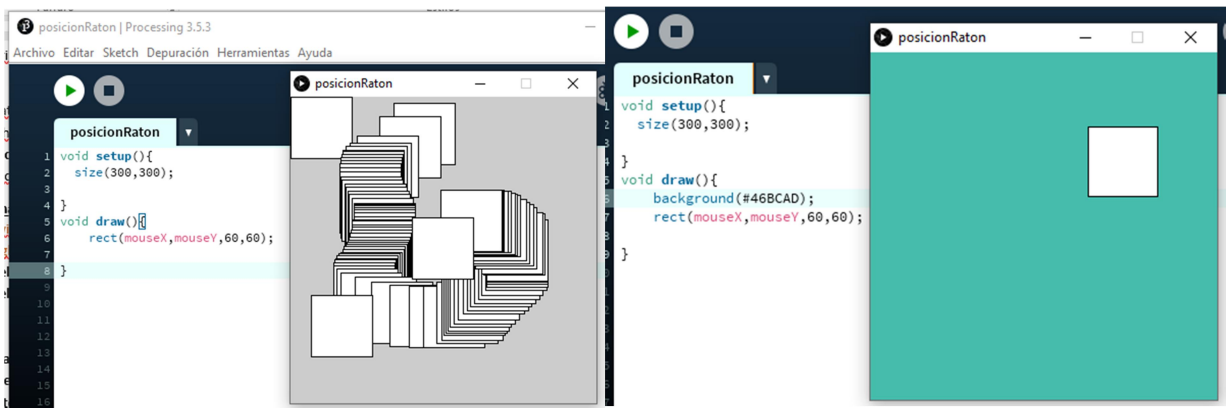
- **Anchura**: width
- **Altura**: height

```
void setup (){
  size(150,150);
  rectMode(CORNER);
  rect(0,0,width*0.5,height*0.5);

}
void draw(){
}
```



- **Posición del ratón en el eje x**: mouseX
- **Posición del ratón en el eje y:** mouseY



De forma infinita se dibujará un cuadrado de 60x60 en la posición del ratón. En el segundo caso solo se ve el ultimo dibujado, en cada repetición se pinta el fondo de Nuevo tapando los cuadrados anteriores.

- **Última letra pulsada**: key
- **Código de letra:** KeyCode

```
void setup(){
  }
void draw(){
    if(keyPressed)
  {
     println("El código de la tecla pulsada es:" + keyCode);
    }
}
```

```
El código de la tecla pulsada es:35
El código de la tecla pulsada es:35
El código de la tecla pulsada es:35
El código de la tecla pulsada es:35
```

Para conocer el código de las distintas teclas especiales (las letras y el espacio tienen código 0)

Código de algunas teclas

- Mayúscula: 16
- Control: 17
- Alt:18
- Bloque mayúsculas: 20
- Fin: 35
- Inicio: 36
- Flecha izquierda:37
- Flecha arriba:38
- Flecha derecha:39
- Flecha abajo:40

- **Presionar tecla**: KeyPressed (bolean)
- **Soltar una tecl**a: keyReleased (bolean)
- **Presionar el ratón:** mousePressed (bolean)
- **Botón del ratón**: mouseButton (bolean)
- **Soltar el ratón:** mouseReleased (bolean)

Texto:

- **Mostrar texto en la consola**: **print("texto");** o **println("texto");** inserta un salto de línea después del texto.

```
print("Hola mundo");print("Hola mundo");
println("¿Qué tal?");println("¿Qué tal?");
```

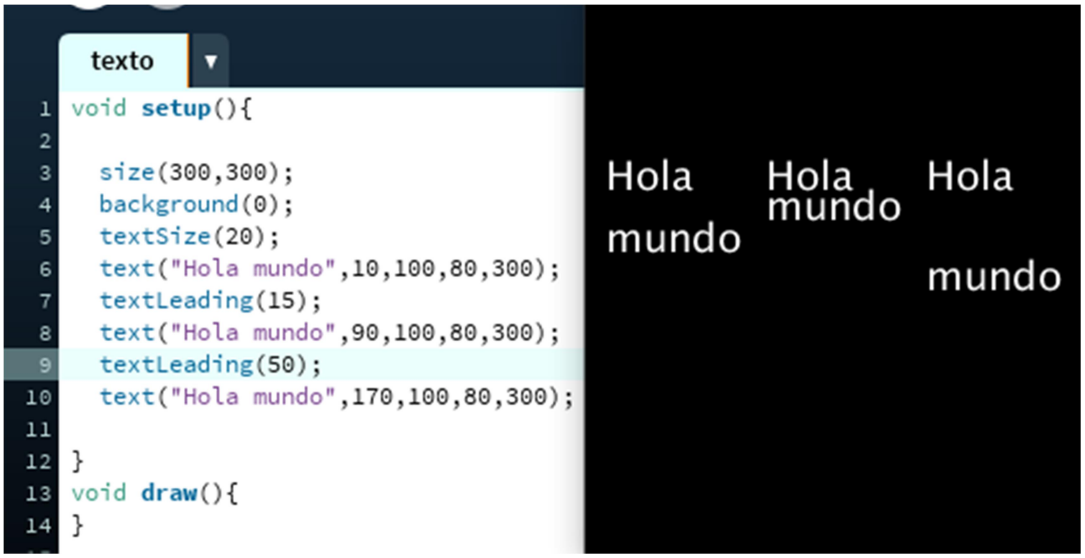
```
Hola mundoHola mundo¿Qué tal?
¿Qué tal?
```

- **Mostrar texto en la ventana**: **text("texto", x,y); textSize(pixeles);** antes del texto al que se le quiere cambiar el tamaño.

```
textSize(22);
text("Hola mundo",50,75);
```

Además del punto de origen del texto puedo la anchura y altura del cuadro que contendrá el texto.

Con **textLeading();** puede configurar la distancia entre las líneas de un texto.



Con **textAlign(**LEFT, CENTER O RIGHT); Podemos alinear el texto dentro del cuadro de texto. Por defecto se alinea a la izquierda.

Para el color, al texto le afecta la orden **fill**, usando el código RGB además del color podemos configurar la transparencia con el último parámetro, desde trasparente (0), hasta opaco (255). También podemos cambiar la fuente con la herramienta crear fuente, en el código crearé una variable **Pfont** y usaremos el nombre del archivo.vlw con la ordenes variable = **loadFont**(“nombrefuente.vlw”) y **textFont**(variable);

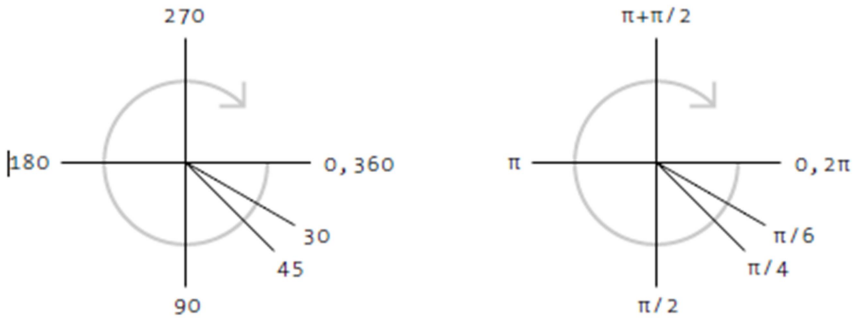
Se ha escrito el mismo texto con el mismo color, cada vez 10pixeles más abajo y con una transparencia menor.

FUNCIONES PROPIAS

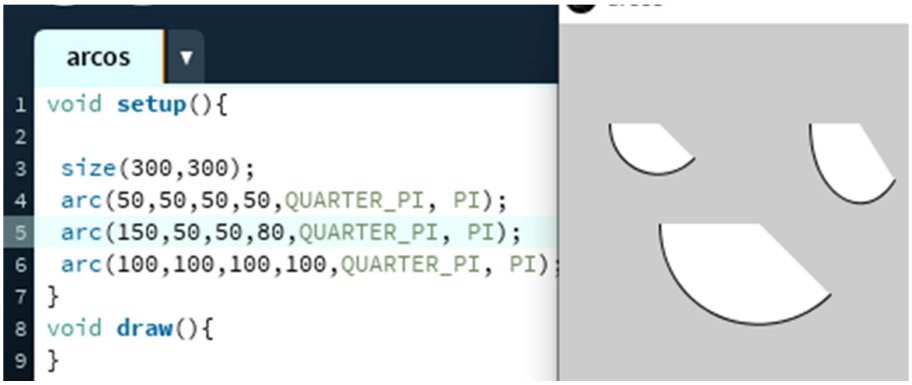
Matemáticas:

- Cuadrado **sq();,** devuelve el cuadrado de un número.
- Raíz cuadrada **sqrt();,** devuelve el cuadrado de un número.
- Exponente **pow(x,n);,** devuelva la potencia i del número x.
- Número pi: **PI**, sus variantes (**QUARTER_PI, HALF_PI, TWO_PI**)
- Trigonometría: seno: **sen(),** coseno: **cos(),**

- Paso de grados radianes:(**radians()** y de radianes a grados **degree();**

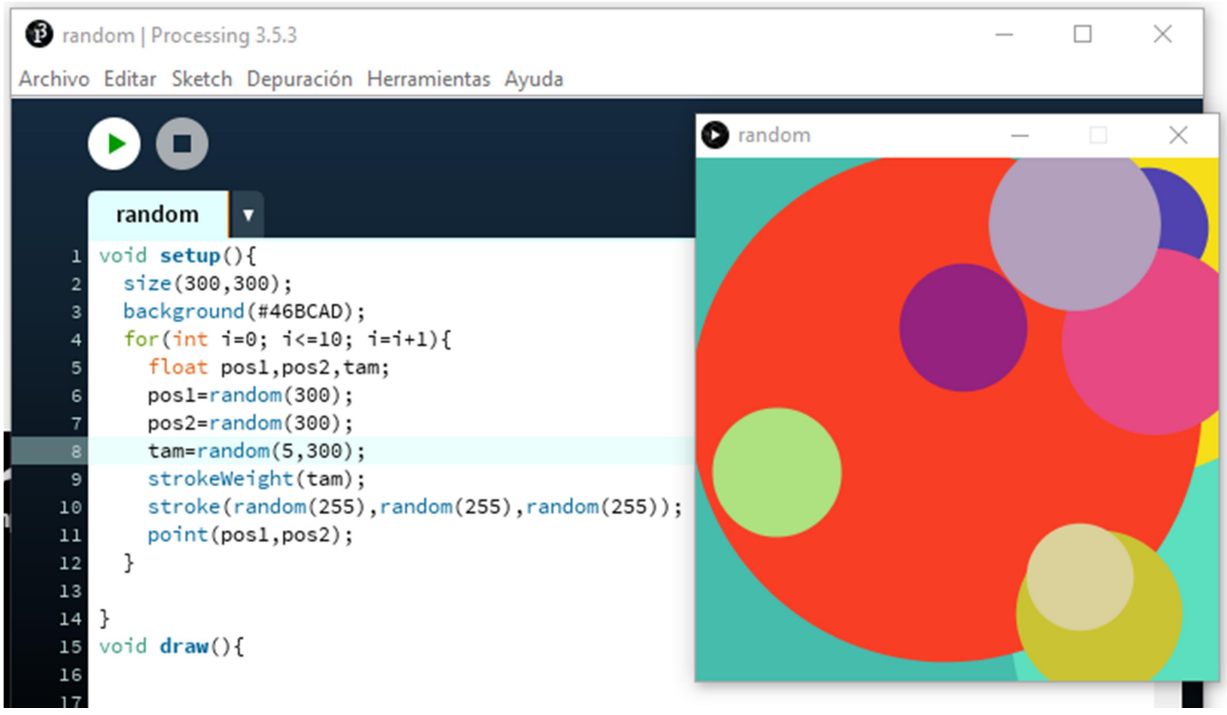


- Convertir valor a otro rango: **map** cambia un valor(valor 1) de un rango indicado(valores 2 y 3) a otro (valores 4 y 5)
- Arcos: **arc();** tiene como valores, el centro (x,y), ancho, alto, ángulo inicial y ángulo final.



Aleatorio:

La función **random** permite crear número aleatorio dentro del rango marcado por los parámetros random(n_1, n_2) o entre 0 y el parámetro marcado random(n);

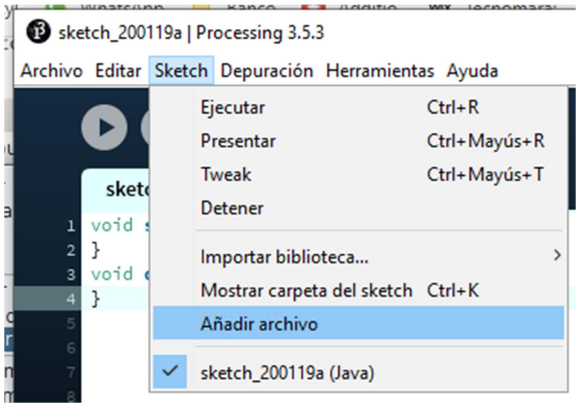


Si en vez de escribir el bucle for, para repetir varias veces la creación de puntos con color, posición y tamaño aleatorio, lo incluimos en la función draw(), crearemos una animación donde irán apareciendo puntos de diversos colores, tamaños, y en distintas posiciones dentro de la pantalla.

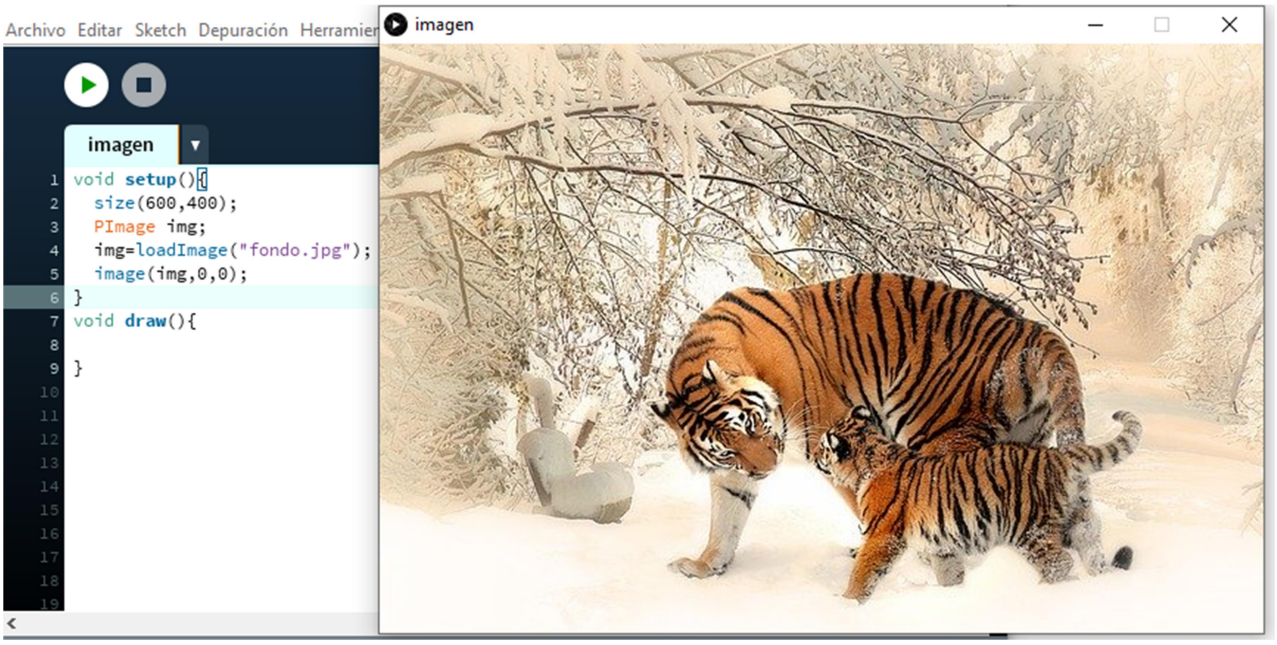
Esta función tiene como variante **randomSeed();** que repetirá en el mismo orden los números aleatorios generados cada vez que se ejecute el programa.

Imagen:

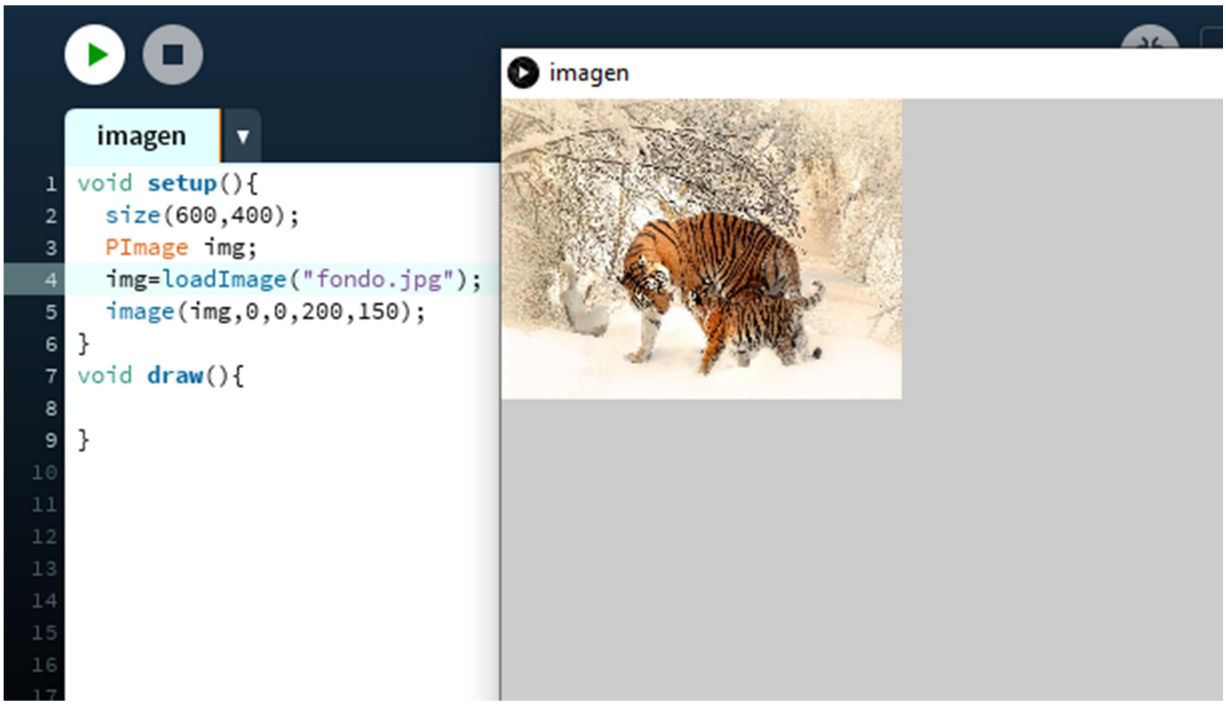
Processing permite incluir imágenes. Para ello deben estar guardadas en la carpeta DATA del proyecto, y la seleccionaremos desde el menú "Sketch", opción "Añadir archivo".



En el código debemos iniciar una variable de tipo PImage, en esta variable cargaremos la imagen seleccionada con **loadImage("imagen.jpg");** y después con la función **img(variable,x,y**); colocaremos al imagen en el lienzo.



Podemos cambiar el tamaño de la imagen, añadiendo los valores al anchura y altura la función img();



Con la orden **tint();** podemos tintar la imagen usando tonos de gris (un valor entre 0 y 255), gris con trasparencia (valor de gris y valor de transparencia), con código RGB con y sin transparencia o con el código hexadecimal. **noTint();** devuelve el color original a la imagen.

Efectos de imágenes

En processing existen varias sentencias que aplican efectos a las imágenes.

Filter(MODO, NIVEL); aplica los siguientes modos en los que se podrá elegir con qué porcentaje se aplica.

- THRESHOLD, convierte a blanco y negro según el nivel.

```
PImage img;
img=loadImage("fondo.jpg");
image(img,0,0,200,150);
filter(THRESHOLD,0.5);
```

- GRAY, convierte en escala de grises.

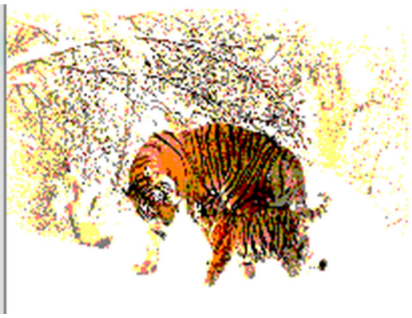
```
PImage img;
img=loadImage("fondo.jpg");
image(img,0,0,200,150);
filter(GRAY);
```

- INVERT, invierte los colores de la imagen.

```
PImage img;
img=loadImage("fondo.jpg");
image(img,0,0,200,150);
filter(INVERT);
```

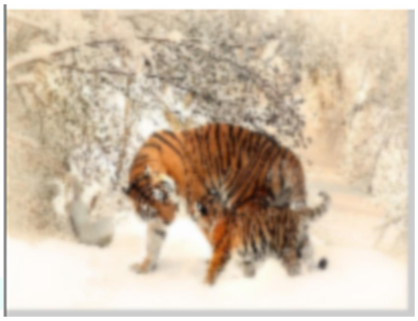
- POSTERIZE, convierte la imagen al número de colores que especifique el nivel.

```
PImage img;
img=loadImage("fondo.jpg");
image(img,0,0,200,150);
filter(POSTERIZE,3);
```



- BLUR, difumina la imagen con la intensidad que indique el nivel.

```
PImage img;
img=loadImage("fondo.jpg");
image(img,0,0,200,150);
filter(BLUR,1);
```



- ERODE, reduce la luz.

```
PImage img;
img=loadImage("fondo.jpg");
image(img,0,0,200,150);
filter(ERODE);
```

- DILATE, incremente la luz.

```
PImage img;
img=loadImage("fondo.jpg");
image(img,0,0,200,150);
filter(DILATE);
```

Blend(); mezcla varias imágenes.

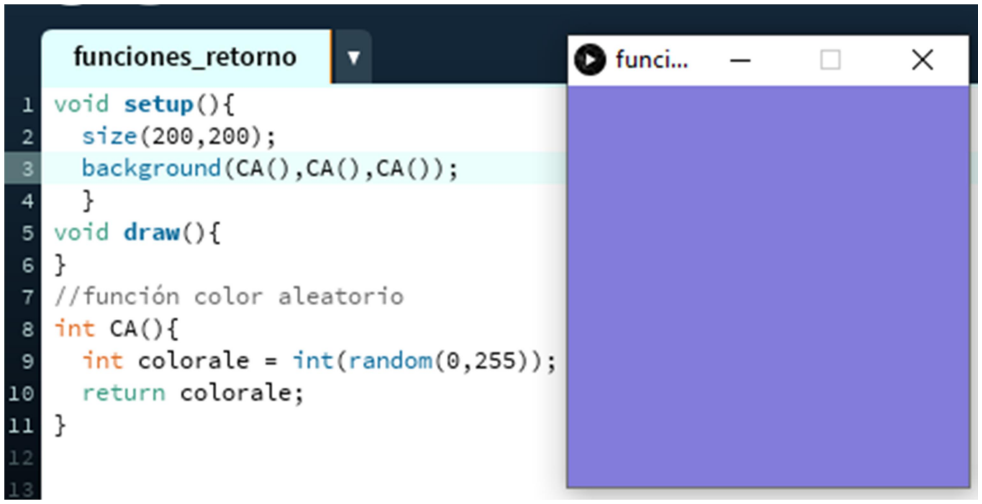
CREAR FUNCIONES

Además de las funciones principales void setup(); y void draw();, podemos crear funciones propias que después llamaremos desde las funciones principales. Esto permitirá organizar las ordenes de forma más sencilla de entender al reducir el número de líneas de código dentro de las funciones principales, y a reducir las órdenes necesarias al reutilizar aquellas sentencias que suelan repetirse.

Para crear una función usaremos void y el nombre de la función. A las funciones podemos pasarle parámetros cuyos valores indicaremos entre los paréntesis de las funciones, o no pasar nada como ocurre con setup o void por lo que los paréntesis aparecen vacíos. Si las funciones van necesitar parámetros se crearán dentro del paréntesis de la creación. El código de la función aparecerá limitado por llaves {}.

Para llamar a la función simplemente se escribirá en las funciones principales el nombre de la función seguido de los paréntesis vacíos o con los valores de los parámetros.

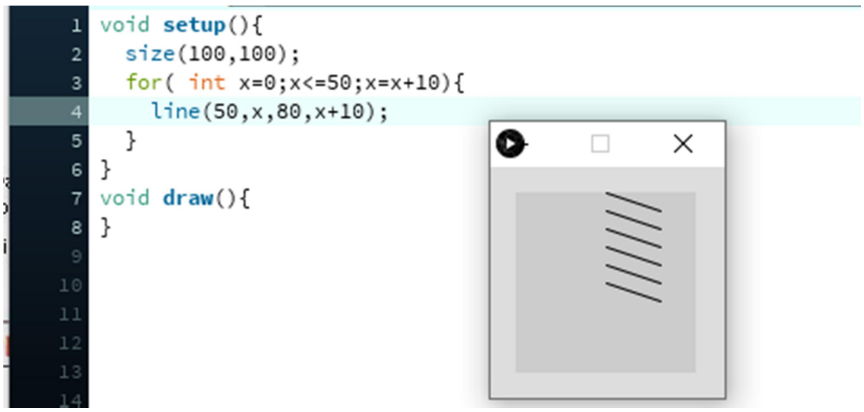
Processing también permite crear funciones que devuelvan valores, En este caso la función no se creará como void, si no del tipo de dato que vaya a devolver (float, int, char, string…). Puede pedir parámetros y la última línea de la función será return y el valor a devolver.



ITERACIONES

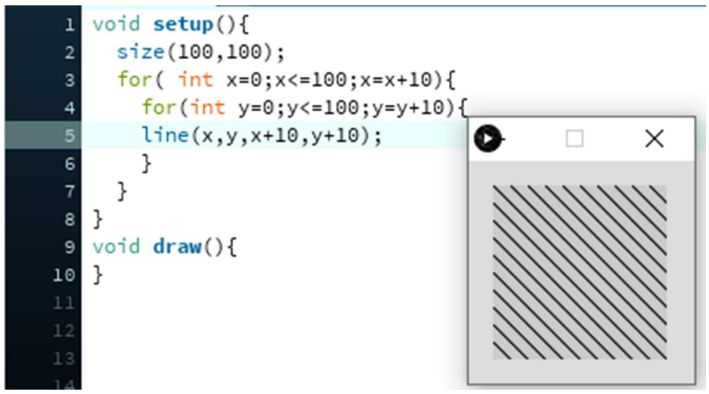
For:

El bucle **for**, permite repetir una serie de órdenes o funciones en función de unas condiciones relacionadas con una variable. Al bucle for debemos indicarle desde y hasta que valores de la variable va a ejecutarse, así como el incremento que sufrirá la variable en cada ciclo.



Hemos repetido la orden línea desde x= hasta x=50 incrementando en 10 el valor de x en cada iteración, por lo que se van a dibujar 5 líneas. Estas líneas tendrán como valor x de inicio 50, y de final 80, mientras que la posición del eje y se va desplazando en 10 cada iteración, por lo que las líneas resultantes serán paralelas y distanciadas por 10 pixeles.

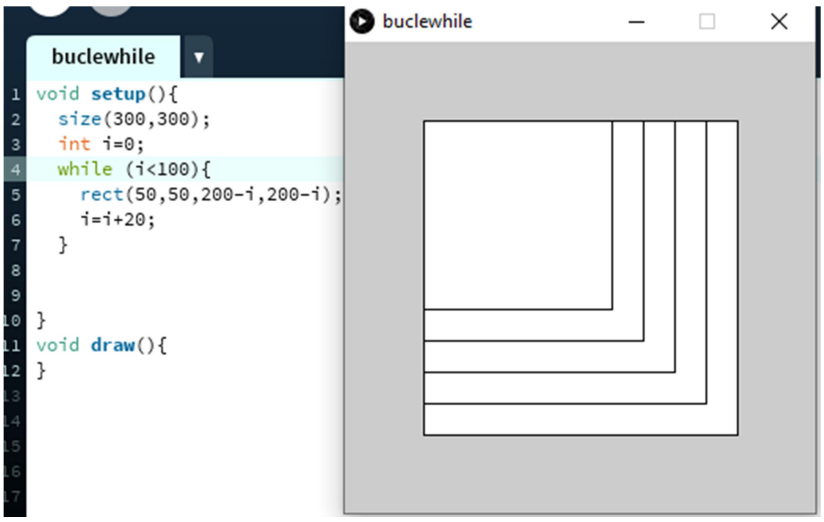
Los bucles for se pueden anidar, y forma común de anidarlos, es crear un bucle for para la posición x y dentro otro para controlar la posición y.



El primer bucle controla desde el origen del eje x (x=0) hasta la anchura máxima de la pantalla (x=100), incrementando cada vez el valor de x en 10 pixeles, a su vez se ejecute el mismo bucle para y (desde el origen del eje y hasta la altura máxima), dibujando en cada iteración una línea diagonal. Llenando así todo el lienzo de líneas diagonales paralelas distanciadas 10 pixeles.

While:

El bucle **while** se repite mientras se cumplen una condición. A diferencia del for, la variable contador a usar se declara fuera del bucle, y el incremento se realiza dentro, como parámetro del while solo se incluye la condición.



CONDICIONALES

Si:

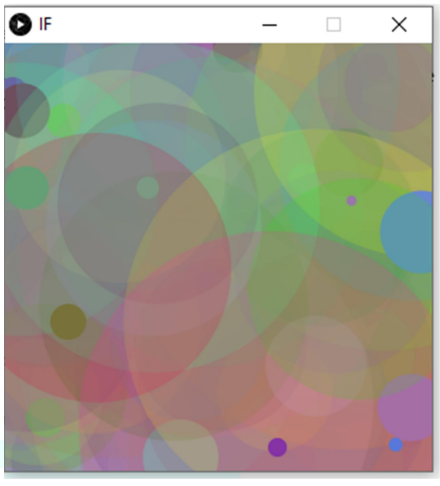
La función **if** permite que se ejecute una parte del código si se cumple una condición.

```
void setup(){
  size(300,300);
  background(#46BCAD);
}
void draw(){
if(mousePressed==true){
  background(random(255),random(255),random(255));
  }
}
```

Si se presiona el ratón el color del fondo cambia de forma aleatoria.

La función **if** puede tener una segunda parte, **else** , de forma que si se cumple una condición se ejecute un código, y si no se ejecute otro.

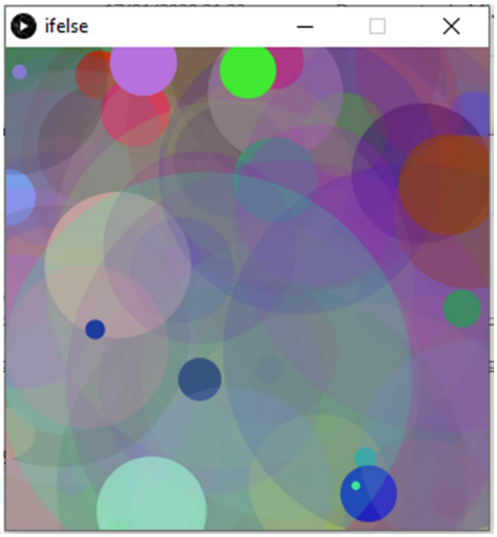
```
void setup(){
  size(300,300);
  background(#46BCAD);
 }
void draw(){
     float pos1,pos2,tam;
    pos1=random(300);
    pos2=random(300);
    tam=random(5,300);
    strokeWeight(tam);
    if (tam<60){
    stroke(random(255),random(255),random(255));
    point(pos1,pos2);
    }else {
       stroke(random(255),random(255),random(255),50);
    point(pos1,pos2);
    }
}
```



Al código de crear puntos con tamaño, color y en sitio aleatorio, se incluimos la condición de que si el tamaño es menor de 60 el punto sea opaco, y si no tenga transparencia de 50.

Los condicionales pueden anidarse.

```
void setup(){
  size(300,300);
  background(#46BCAD);
 }
void draw(){
     float pos1,pos2,tam;
    pos1=random(300);
    pos2=random(300);
    tam=random(5,300);
    strokeWeight(tam);
    if (tam<60){
      stroke(random(255),random(255),random(255));
      point(pos1,pos2);
    }else if(tam<100) {
       stroke(random(255),random(255),random(255),150);
       point(pos1,pos2);
    }else{
      stroke(random(255),random(255),random(255),50);
      point(pos1,pos2);
    }
}
```

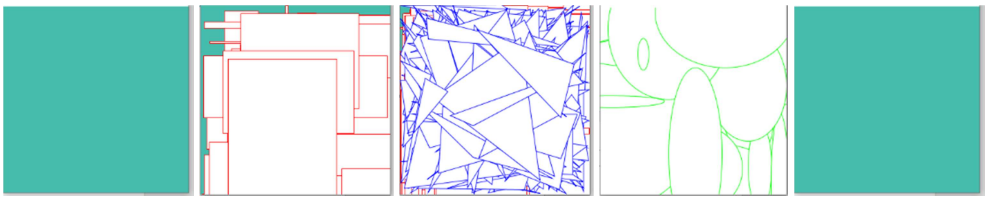


Ahora si los puntos son menores de 60 serán opacos, si miden entre 60 y 100 tendrán una transparencia de 150, y si son mayores de 100 de 50.

En caso de:

Si una variable puede tomar varios valores podemos usar la función **Switch**. Necesitaremos una variable, en función del valor de esa variable tendremos varios casos, **case**, que harán que se ejecute un determinado código delimitado por un **break**. La función switch cuenta con la opción **default**, que se ejecutará si no la variable no coincide con el valor de ningún case. La variable switch debe ser un entero o carácter, por lo que su uso está limitado.

```
void setup(){
  size(300,300);
  background(#46BCAD);
 }
void draw(){
        switch(key){
    case 'r':
      stroke(255,0,0);
      rect(random(300),random(300),random(300),random(300));
    break;
    case 'e':
      stroke(0,255,0);
      ellipse(random(300),random(300),random(300),random(300));
    break;
    case 't':
      stroke(0,0,255);
      triangle(random(300),random(300),random(300),random(300),random(300),random(300));
      break;
    default:
      background(#46BCAD);
    }
}
```



Al pulsar la r, se dibujan cuadrados con borde rojo de tamaño y posición aleatoria, con la t se dibujan triángulo de borde azul, con la e elipses con borde verde, y con cualquier otra tecla se vuelve al fondo inicial.

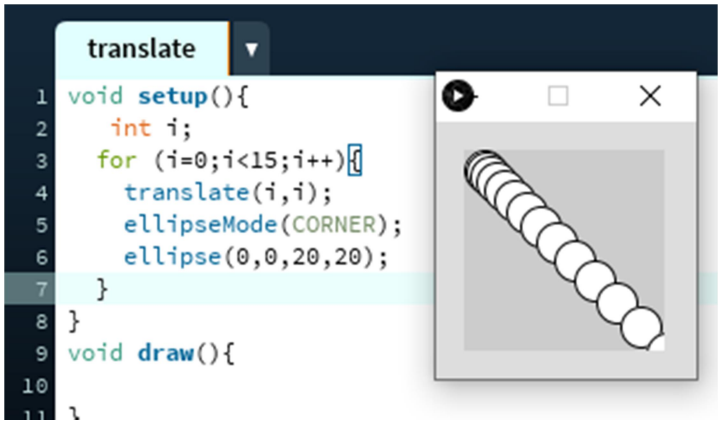
EFECTOS

Escalar:

Scale(n); escala la forma a un tamaño n veces el original. Afecta a las formas dibujadas después.

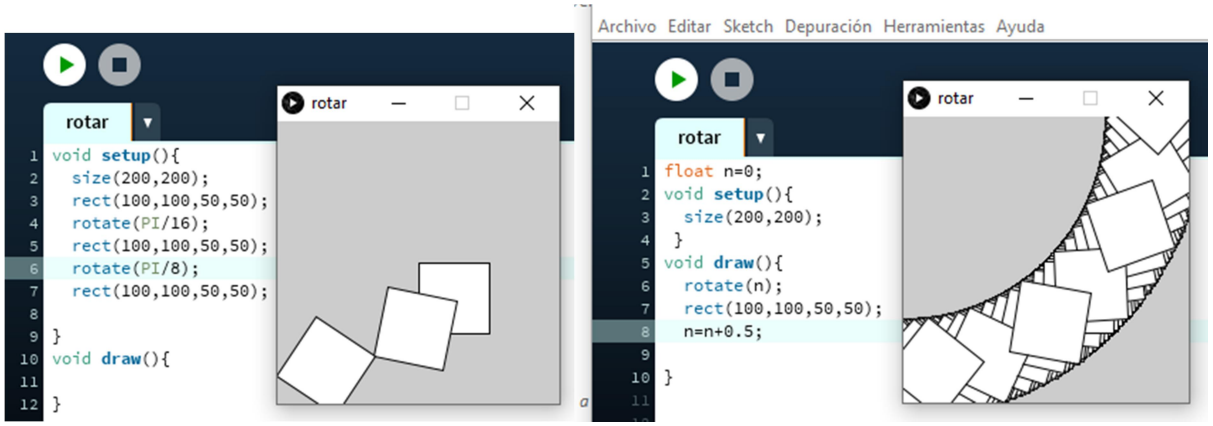
Trasladar:

Translate(x,y); Traslada la forma dibujada después al punto indicado. Si al inicio del programa usamos translate, estamos cambiando el origen del eje de coordeandas.

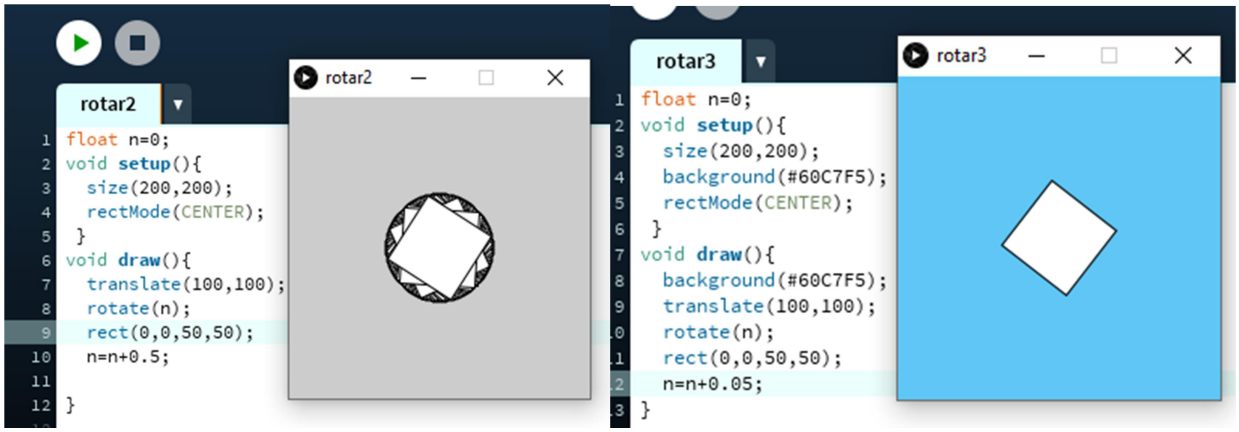


Rotación:

Con processing se pueden girar las formas con la orden **rotate(n);** donde n es el ángulo de giro, El giro se produce respecto al eje de coordenadas. En el segundo ejemplo al ejecutar la rotación en el draw, se repetirá creando una animación.

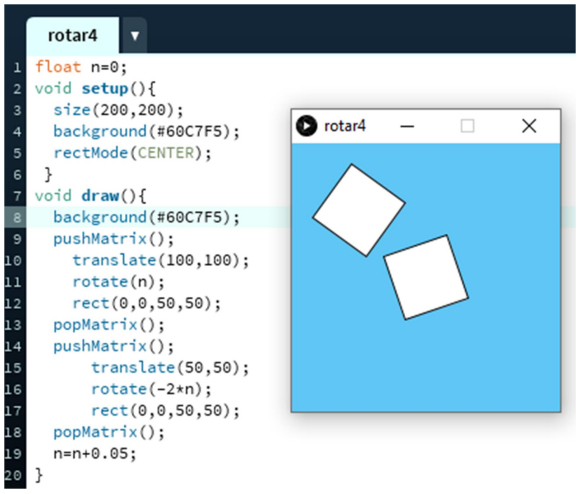


Para que un objeto gire sobre su centro, tendremos que cambiar el modo de la figura y trasladar el objeto al centro del lienzo. Si no queremos que se vea la estela que dejan los cuadrados al sobrescribirse, podeos cambiar el color del fondo en cada ciclo.



Reiniciar Efectos

Para que los efectos anteriores no se apliquen a todas las formas dibujadas después podemos usar **pushMatrix();** y **popMatrix();** para limitar a quién afecta cada opción, o **resetMatrix();** para resetear los efectos aplicados antes.



TIEMPO Y VELOCIDAD

Processing permite modificar la velocidad a la que se muestran los frames, cada uno de los ciclos. Indicando cuando se muestran por segundo, por defecto la velocidad es de 60 por segundo. La opción **frameRate();** permite modificar esta velocidad.

```
void draw(){
    frameRate(10);
    float pos1,pos2,tam;
    pos1=random(300);
    pos2=random(300);
    tam=random(5,300);
    strokeWeight(tam);
    stroke(random(255),random(255),random(255));
    point(pos1,pos2);
```

Cuando modificamos a velocidad de ejecución, podemos usar la orden **saveFrame()** para guardar cada uno de los ciclos como imagen en la carpeta del proyecto. Pare ello, incluiremos la orden al final del código y ejecutaremos el proyecto durante unos segundos. Al parar el proyecto en la carpeta tendremos guardados los "frames", una imagen por cada ciclo ejecutado.

```
random
void draw(){
    frameRate(10);
    float pos1,pos2,tam;
    pos1=random(300);
    pos2=random(300);
    tam=random(5,300);
    strokeWeight(tam);
    stroke(random(255),random(255),random(255));
    point(pos1,pos2);
    saveFrame();
}
```

La función millis(); almacena el tiempo en milisegundos desde que empezó la ejecución del programa. Esta función permite controlar en qué momento se ejecuta una orden.

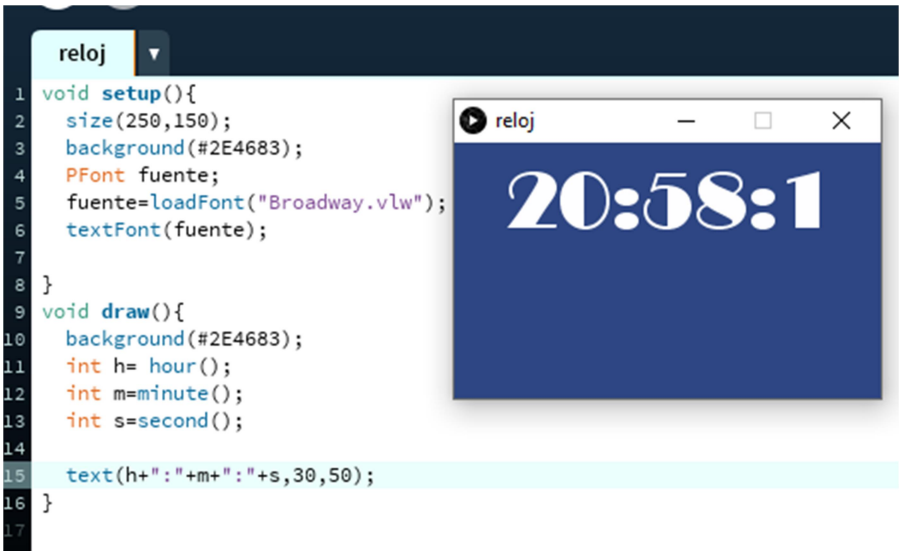
```
reloj
int x=0;
void setup(){
  size(200,200);
  background(#C84CFA);
}
void draw(){
if (millis()>4000){
  x++;
}
ellipse(100,100,x,x);
}
```

En el ejemplo, a los 4 segundos de empezar se dibujaría una circunferencia cuyo radio va creciendo.

FECHA Y HORA

En processing podemos incluir información sobre la fecha y hora actual con las siguientes funciones.

- Día: **day();**
- Mes: **month();**
- Año: **year();**
- Hora: **hour();**
- Minuto: **minute();**
- Segundo: **second();**



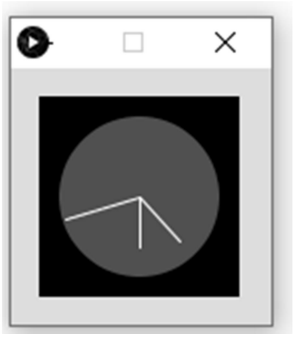
Guardando en variables el resultado de las funciones de hora, podemos crear un reloj digital, mostrando como texto el contenido de las variables, al que podemos dar distinto formato.

Para que el valor de las horas, minutos, o segundos siempre se muestre en 2 dígitos, podemos usar **nf(variable,nºdígitos);**

Usando el seno y el coseno de las funciones de hora podemos dibujar un reloj analógico. (código obtenido de programaciónyrobotica.com).

reloj_analogico

```
void setup() {
size(100, 100);
stroke(255);
}
void draw() {
background(0);
fill(80);
noStroke();
// Los ángulos para el senio y el coseno, empiezan a las 3, es necesario restar PI/2 para que empieze al principio
ellipse(50, 50, 80, 80);
float s = map(second(), 0, 60, 0, TWO_PI) - HALF_PI;
float m = map(minute(), 0, 60, 0, TWO_PI) - HALF_PI;
float h = map(hour() % 12, 0, 12, 0, TWO_PI) - HALF_PI;
stroke(255);
line(50, 50, cos(s) * 38 + 50, sin(s) * 38 + 50);
line(50, 50, cos(m) * 30 + 50, sin(m) * 30 + 50);
line(50, 50, cos(h) * 25 + 50, sin(h) * 25 + 50);
}
```



ARRAY

Para guardar una serie de datos, existe un tipo de dato complejo, el array. Nos va a permitir guardar y leer varios datos de una misma variable.

Para poder asignar y recuperar los datos usaremos **nombre_array[n]**, donde n es la posición del array. Para asignar todos los valores podemos usar {} que incluirá la lista de valores separados por comas. Los array pueden contener valores de distintos tipos de datos, a la hora de crearlo deberemos indicar el tipo de dato que contendrá. La orden **nombre_array.lenght** indica el tamaño total del array, el número de valores que almacena.

array

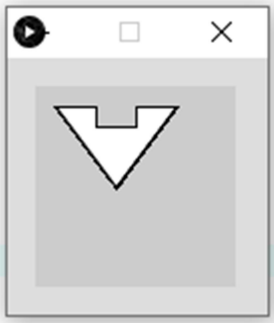
```
1  int[] data={10,20,30,40,50,60,70,80,90};
2  void setup() {
3  size(100, 100);
4  }
5  void draw() {
6    stroke(random(255),random(255),random(255));
7    strokeWeight(5);
8    for (int i=0; i<data.length; i++){
9       point(data[i],data[i]);
10    }
11 }
```

En el ejemplo se dibujan puntos de color aleatorio en la diagonal en las posiciones 10,10 al 90,90, posiciones marcadas por los valores del array.

MATRICES

Las matrices son arrays de dos filas, como valores se guardan parejas de datos (posición 1 de las filas 0 y 1) .

```
int[][] puntos={{10,10},{30,10},{30,20},{50,20},{50,10},{70,10},{40,50}};
void setup() {
size(100, 100);
}
void draw() {
  beginShape();
   for (int i=0; i<puntos.length; i++){
     vertex(puntos[i][0],puntos[i][1]);
    }
    endShape(CLOSE);
}
```



En el ejemplo hemos guardado en una matriz los puntos para crear una forma. Dentro de beginShape hemos recorrido la matriz para que cada columna nos indique las coordenadas de cada vértice de la figura.

FUENTES CONSULTADAS

http://www.programacionyrobotica.com/

https://programarfacil.com/

http://processingjs.org/

https://processing-spain.blogspot.com/