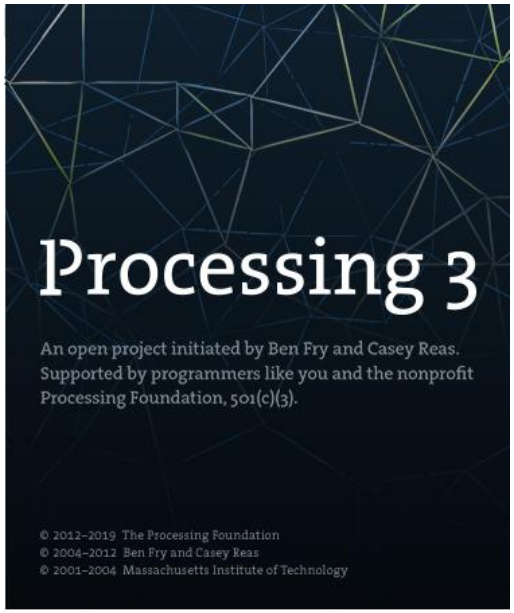
PROCESSING

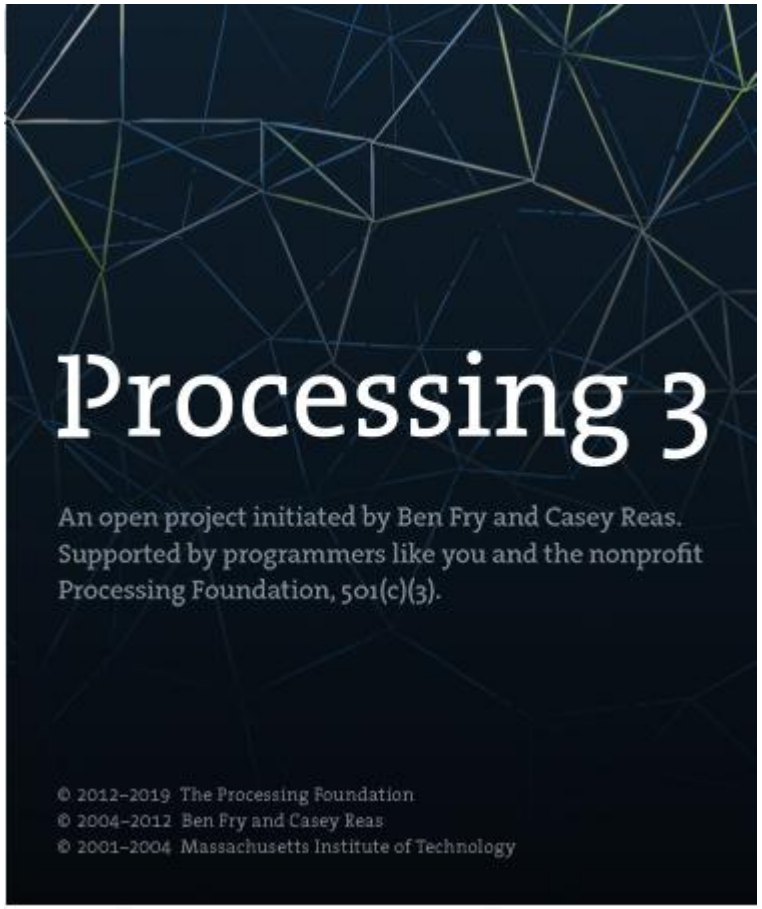
Anexo 1



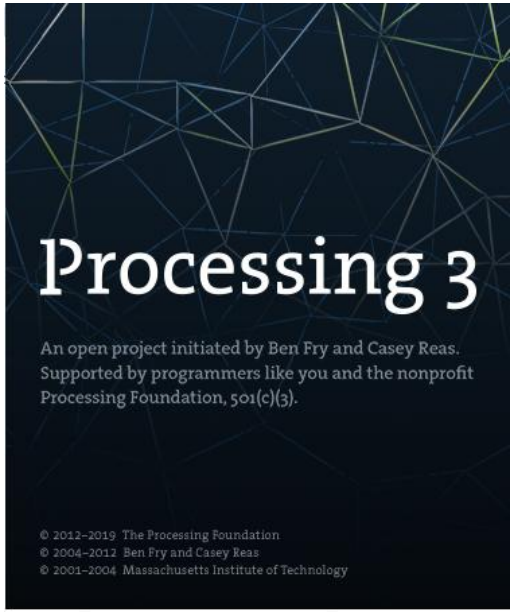
Processing

PROCESSING

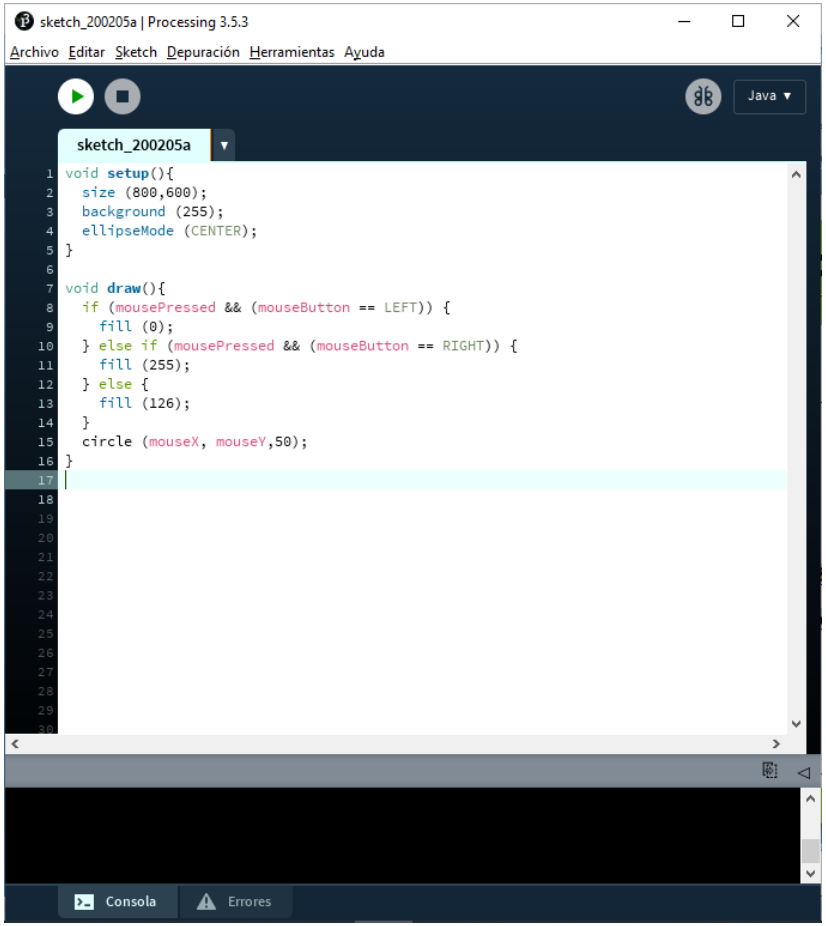
Processing



PROCESSING



Processing



PROCESSING

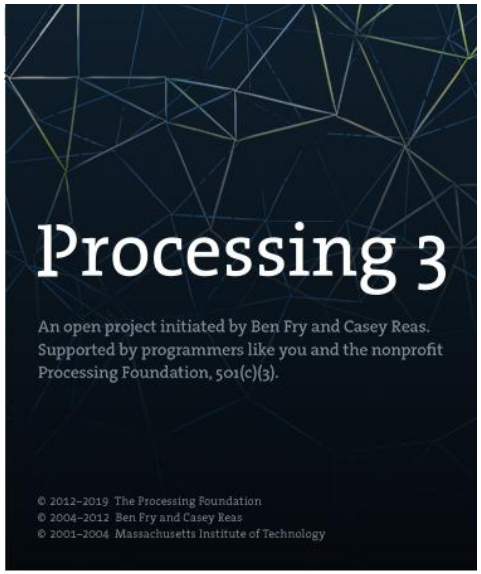
Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital. https://processing.org/

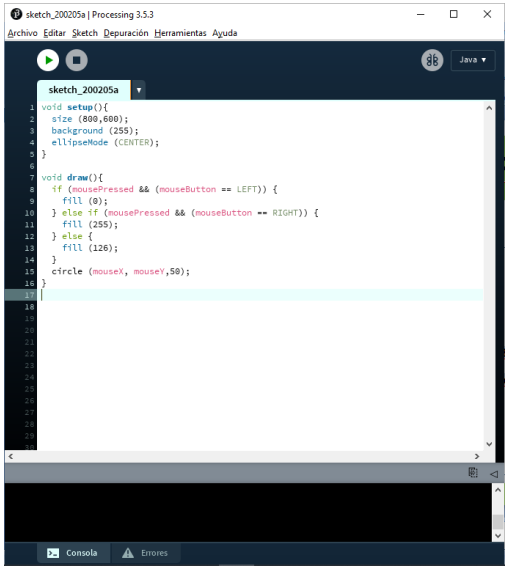
Uno de los objetivos declarados de Processing es el de actuar como herramienta para que artistas, diseñadores visuales y miembros de otras comunidades ajenos al lenguaje de la programación, aprendieran las bases de la misma a través de una muestra gráfica instantánea y visual de la información.

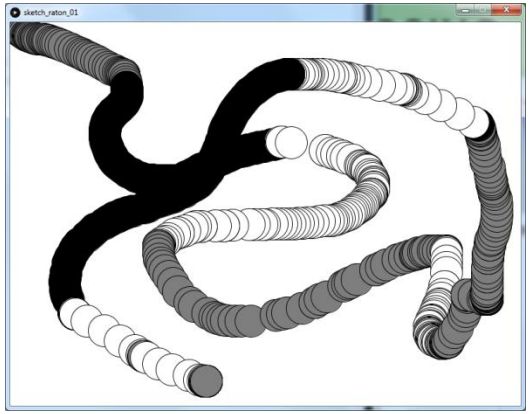
https://es.wikipedia.org/wiki/Processing

PROCESSING

Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital.



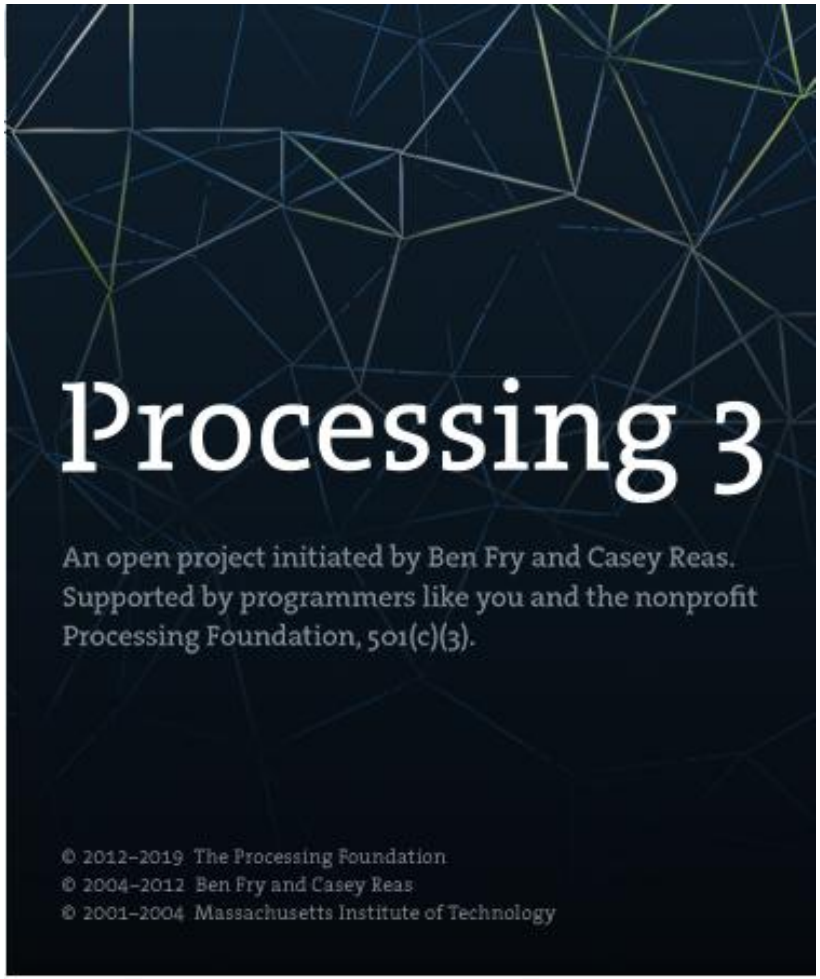




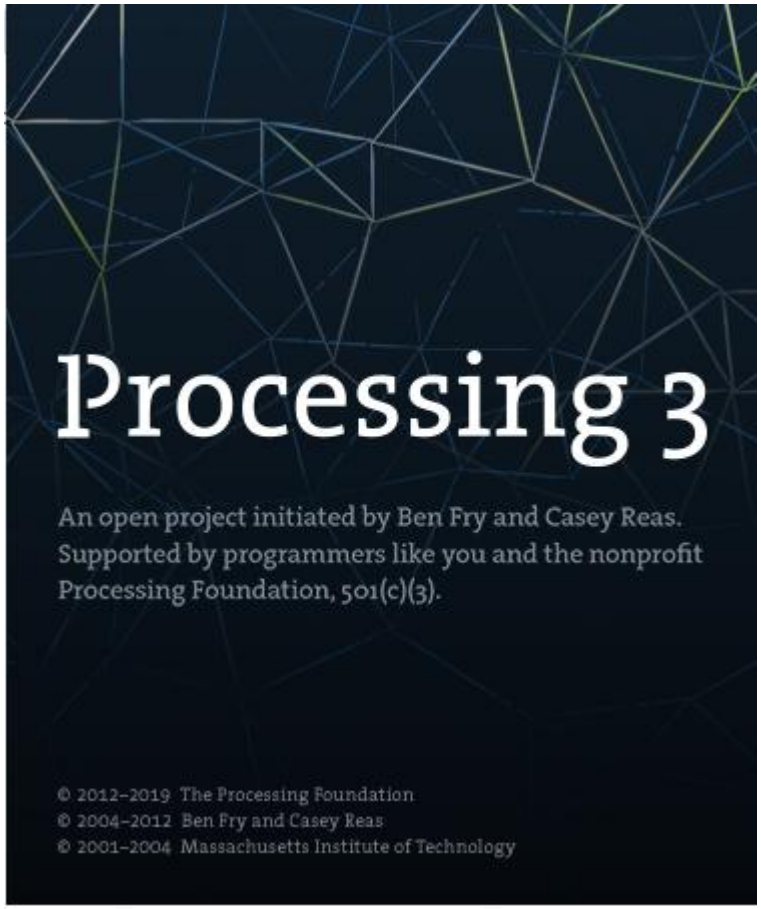
https://processing.org/

PROCESSING

Uno de los objetivos declarados de Processing es el de actuar como herramienta para que artistas, diseñadores visuales y miembros de otras comunidades ajenos al lenguaje de la programación, aprendieran las bases de la misma a través de una muestra gráfica instantánea y visual de la información.
https://es.wikipedia.org/wiki/Processing



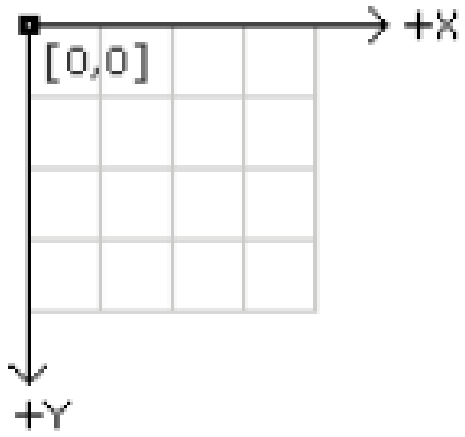
PROCESSING



La pantalla

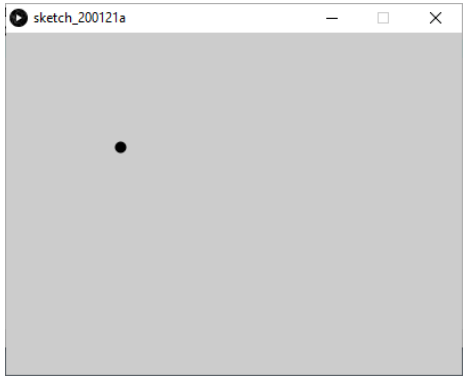
PROCESSING

Pantalla:



La pantalla es una rejilla de píxeles que se identifican por sus coordenadas (X,Y).
El origen (0,0) está en la esquina superior izquierda de la pantalla.
Las x aumentan hacia la derecha y las y aumentan hacia abajo.

```
size (400,300);
strokeWeight(10);
point(100,100);
```



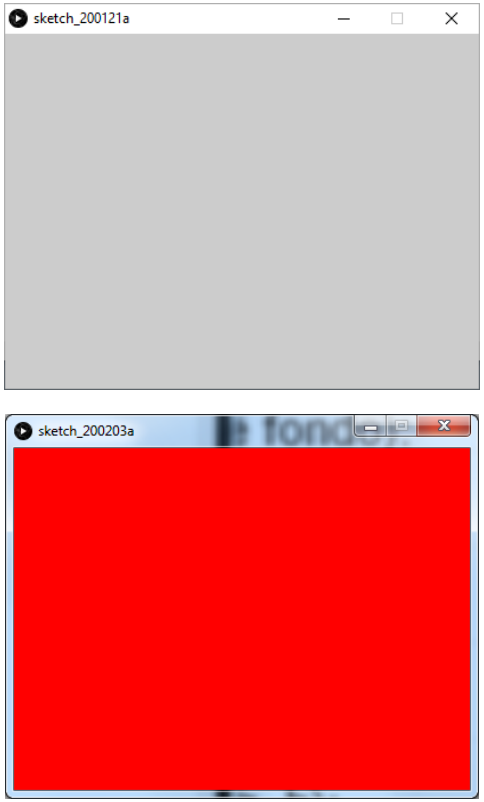
https://processing.org/reference/

PROCESSING

Pantalla (color de fondo):

- size(ancho, alto);

- background(x);
- background(x, t);
- background(r, g, b);
- background(r, g, b, t);

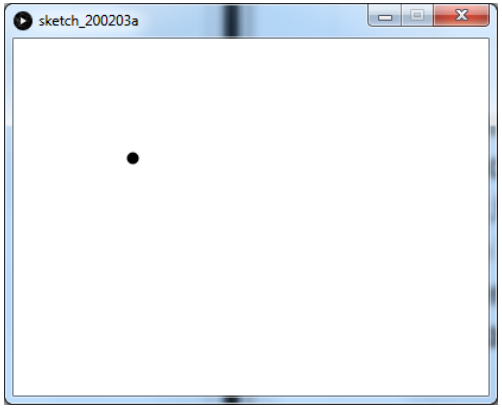


https://processing.org/reference/

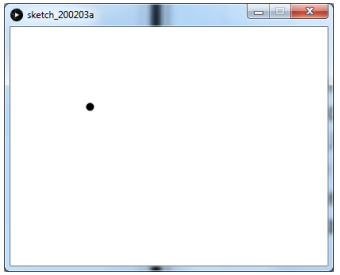
PROCESSING

Pantalla (color de fondo):

```
size (400,300);
background(255);
strokeWeight(10);
point(100,100);
```



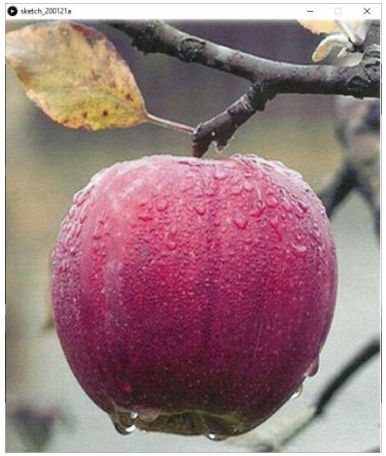

```
size (400,300);
background(255, 255, 255);
strokeWeight(10);
point(100,100);
```




```
size (400,300);
background(255, 0, 0);
strokeWeight(10);
point(100,100);
```

PROCESSING

Pantalla (imagen de fondo):



size (ancho, alto);

PImage imagen;
imagen=loadImage(“ruta_del_archivo”);
background(imagen);

https://processing.org/reference/

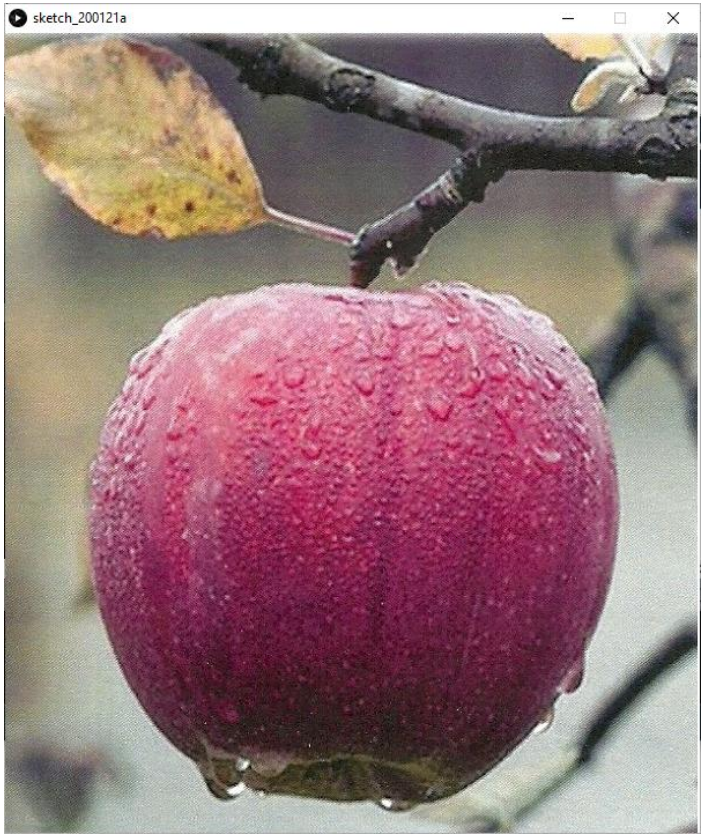
Importante: la imagen de fondo ha de tener las mismas dimensiones que la pantalla.

PROCESSING

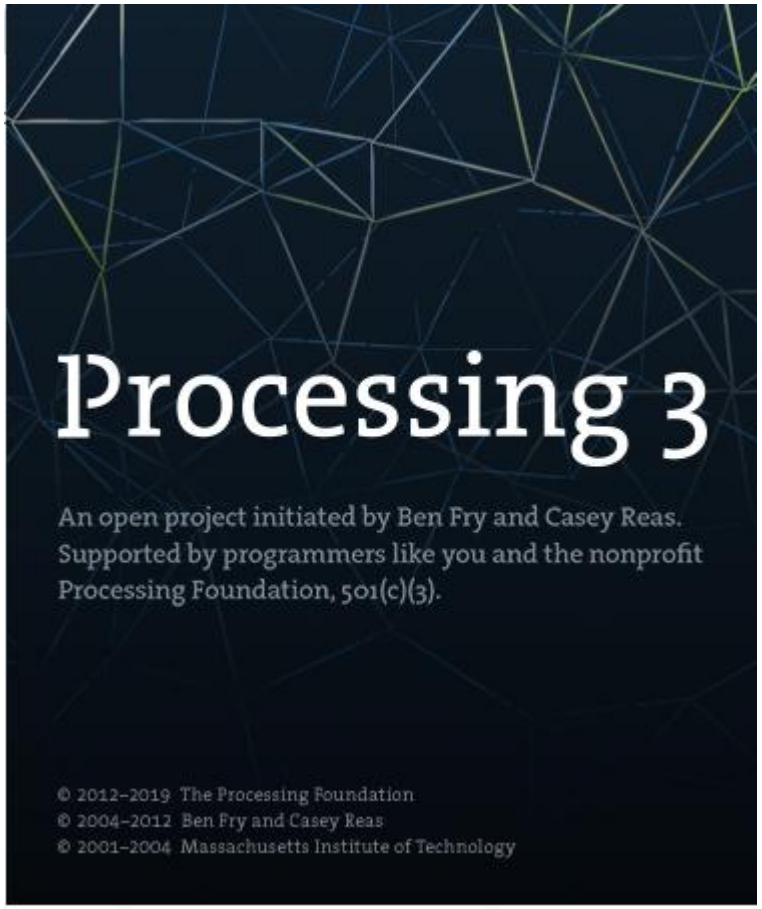
Cargar imagen de fondo:

```
size(601,692);
PImage imagen;
imagen=loadImage("manzana.jpg");
background(imagen);
```

Importante: Para poder utilizar una determinada imagen, antes hay que importarla a la carpeta data del sketch. Para ello hay que utilizar la opción de menú **Sketch -> Añadir archivo** para copiar el archivo de imagen a la carpeta data.



PROCESSING



Primitivas 2D

PROCESSING

Primitivas 2D:

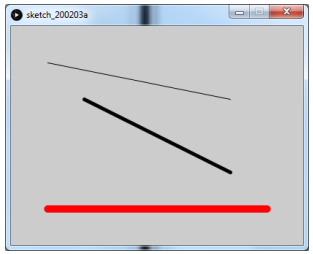
- arc()
- circle()
- ellipse()
- line()
- point()
- quad()
- rect()
- square()
- triangle()

Color:

- clear()
- colorMode()
- fill()
- noFill()
- noStroke()
- stroke()

Atributos:

- ellipseMode()
- rectMode()
- strokeCap()
- strokeJoin()
- strokeWeight()



https://processing.org/reference/

PROCESSING

Primitivas 2D:

- arc()
- circle()
- ellipse()
- line()
- point()
- quad()
- rect()
- square()
- triangle()

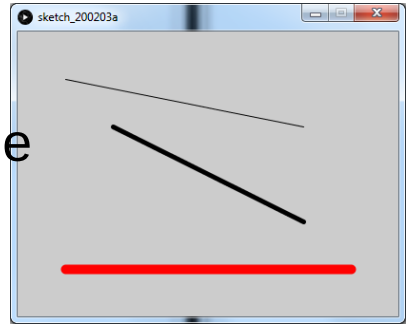
Color:

- fill() //Color de relleno
- noFill() //Sin relleno
- stroke() //Color del borde
- noStroke() //Sin borde

Atributos:

- strokeWeight() //Grosor del borde

```
size (400,300);
line(50,50,300,100);
strokeWeight(5);
line(100,100,300,200);
strokeWeight(10);
stroke(255, 0,0);
line(50,250,350,250);
```



https://processing.org/reference/

PROCESSING

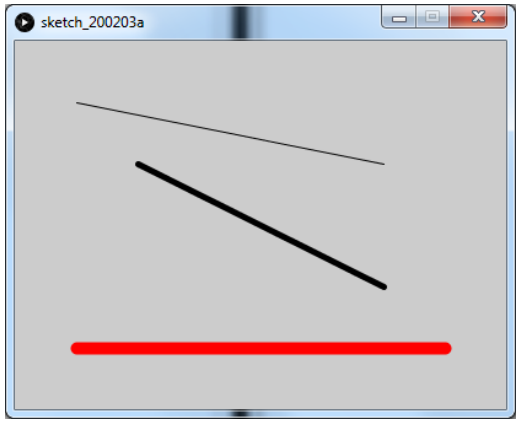
Color:

- fill() //Color de relleno
- noFill() //Sin relleno
- stroke() //Color del borde
- noStroke() //Sin borde

Atributos

- strokeWeight() //Grosor del borde

```
size (400,300);
line(50,50,300,100);
strokeWeight(5);
line(100,100,300,200);
strokeWeight(10);
stroke(255, 0,0);
line(50,250,350,250);
```



https://processing.org/reference/

PROCESSING

Primitivas 2D:

- point()
- line()
- triangle()
- quad()
- arc()

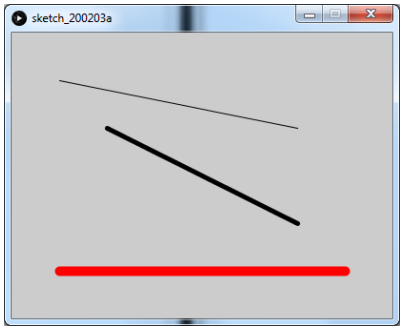
Color:

- stroke()
- noStroke()

Atributos:

- strokeWeight()

```
size (400,300);
line(50,50,300,100);
strokeWeight(5);
line(100,100,300,200);
strokeWeight(10);
stroke(255, 0,0);
line(50,250,350,250);
```



Atributos:

- strokeCap()
  - ROUND, SQUARE, PROJECT

- strokeJoin()
  - BEVEL, MITTER, ROUND

https://processing.org/reference/

PROCESSING

Primitivas 2D:

- circle()
- ellipse()
- square()
- rect()

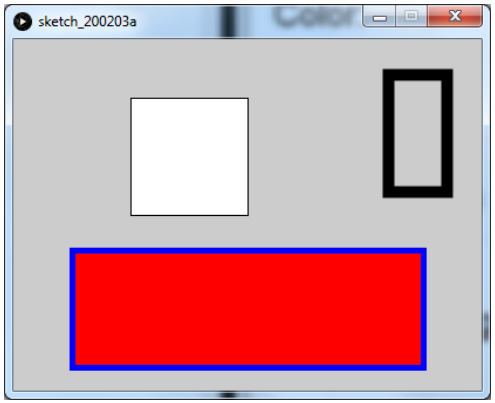
Color:

- fill()
- stroke()
- noFill()
- noStroke()

Atributos:

- strokeWeight()

```
size (400,300);
rect(100,50,100,100);
noFill();
strokeWeight(10);
rect(320,30,50,100);
strokeWeight(5);
stroke(0, 0, 255);
fill(255,0,0);
rect(50,180,300,100);
```



Atributos:

- ellipseMode()
  - CENTER, RADIUS, CORNER, CORNERS
- rectMode()
  - CORNER, CORNERS, CENTER

https://processing.org/reference/

PROCESSING

Atributos:

https://processing.org/reference/

- ellipseMode()
  - CENTER, RADIUS, CORNER, CORNERS
- rectMode()
  - CORNER, CORNERS, CENTER
- strokeCap()
  - ROUND, SQUARE, PROJECT

- strokeJoin()
  - BEVEL, MITTER, ROUND

- strokeWeight()

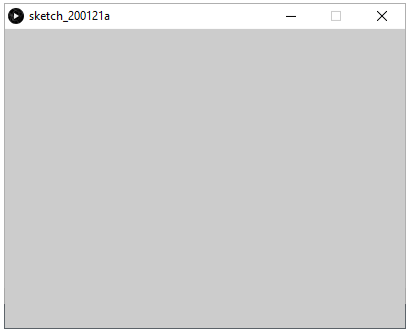
PROCESSING

Antialiasing:

- smooth()
- noSmooth()

https://processing-spain.blogspot.com/2015/09/311-propiedades-de-la-forma-smooth-y.html

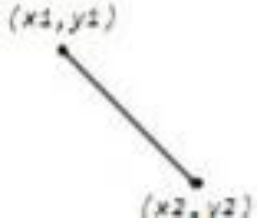
```
smooth(0); // Nada de suavizado
smooth(2); // Por defecto en P2D y P3D
smooth(3); // Por defecto en JAVA2D
smooth(4);
smooth(8); // Máximo suavizado

noSmooth(); = smooth(0);
```

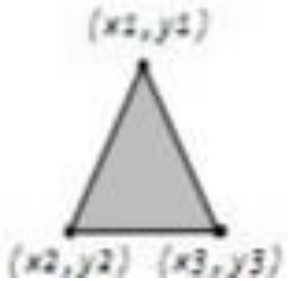


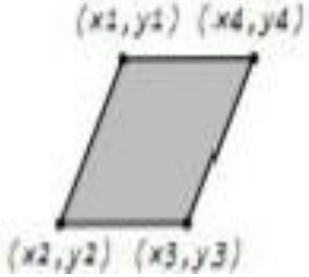
https://processing.org/reference/

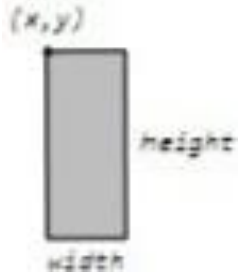
PROCESSING

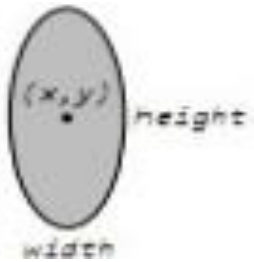
Primitivas 2D:

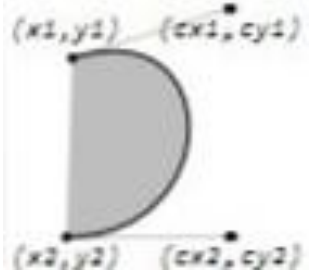








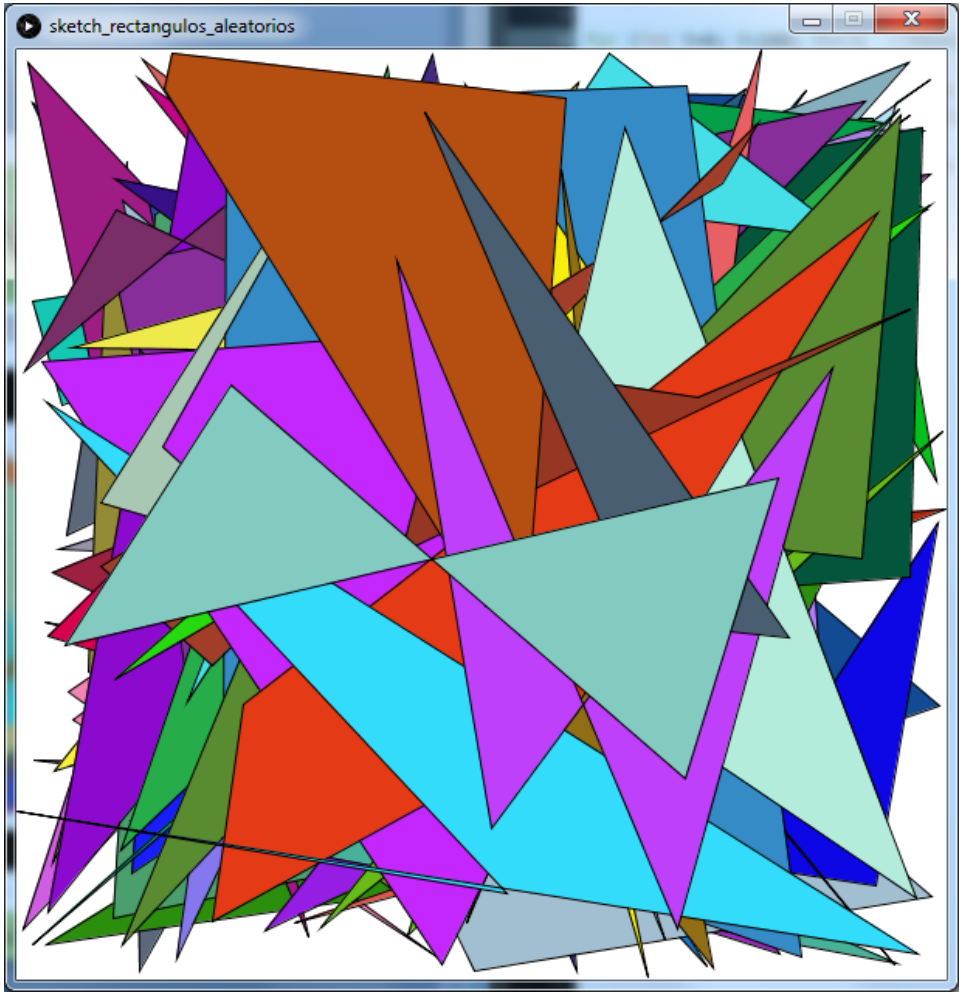




PROCESSING

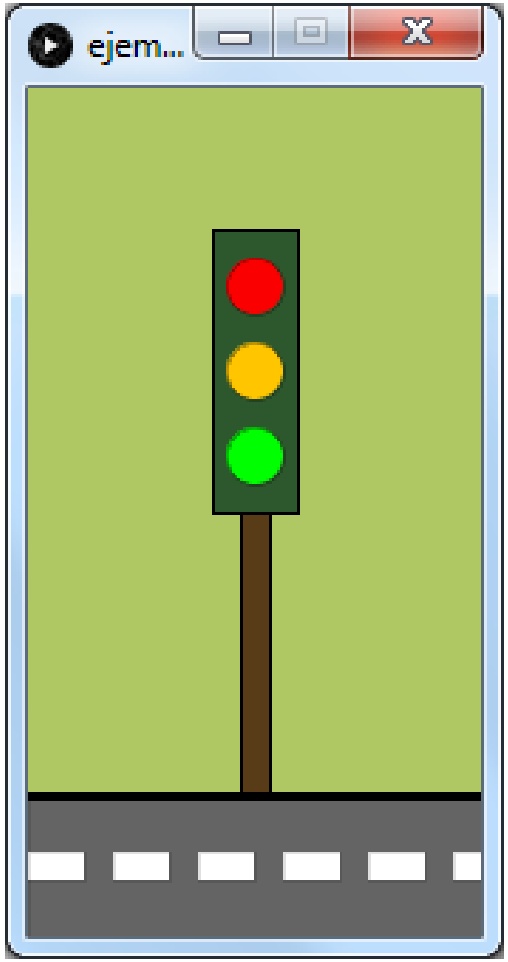
ejemplo_03

```
1
2  //Programa que dibuja cuadriláteros aleatoriamente
3  int x1,y1,x2,y2,x3,y3,x4,y4;
4
5  size(600,600);
6  background(255);
7
8  for (int i=0; i<100; i++){  //Repetir 100 veces
9
10   //Establecer el color de relleno
11   fill(random(0,255),random(0,255),random(0,255));
12
13   //Generar las coordenadas de los vértices
14   x1=(int)random(0, width);
15   y1=(int)random(0, height);
16   x2=(int)random(0, width);
17   y2=(int)random(0, height);
18   x3=(int)random(0, width);
19   y3=(int)random(0, height);
20   x4=(int)random(0, width);
21   y4=(int)random(0, height);
22
23   //Dibujar el cuadrilatero
24   quad (x1,y1,x2,y2,x3,y3,x4,y4);
25
26 }
```

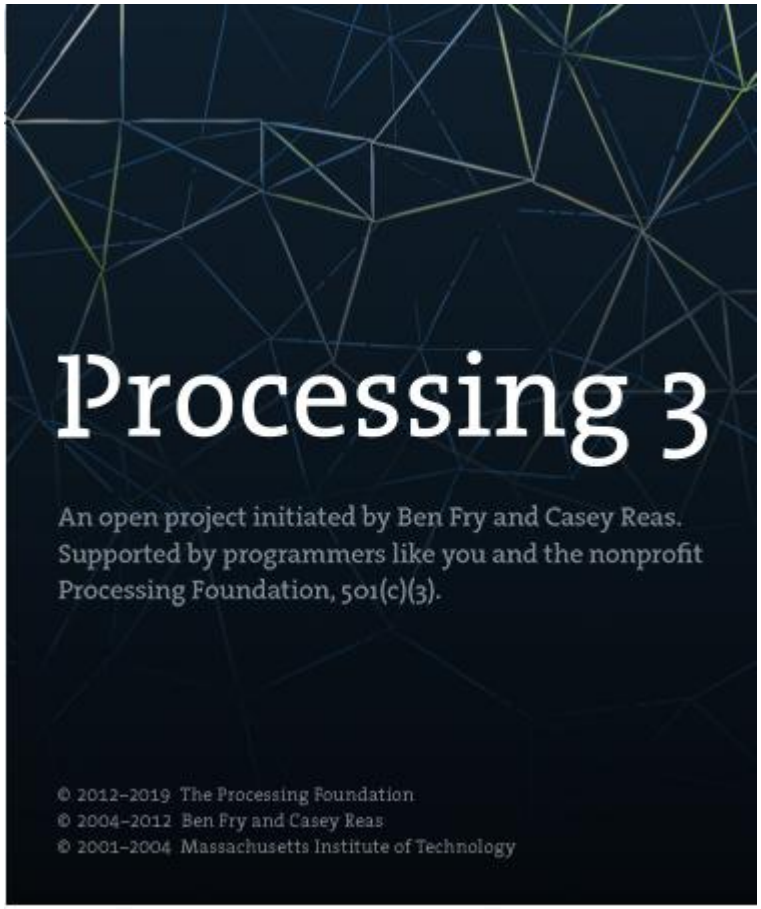


PROCESSING

```
size(160, 300);
background(175, 200, 100);
smooth();
fill(88, 60, 23);
strokeWeight(1);
rect(75, 150, 10, 100);
fill(45, 87, 44);
rect(65, 50, 30, 100);
fill(250, 0, 0);
strokeWeight(0);
ellipse(80, 70, 20, 20);
fill(255, 198, 0);
ellipse(80, 100, 20, 20);
fill(0, 255, 0);
ellipse(80, 130, 20, 20);
fill(100);
rect(0, 250, 160, 50);
strokeWeight(3);
line(0, 250, 160, 250);
fill(255);
strokeWeight(0);
rect(0, 270, 20, 10);
rect(30, 270, 20, 10);
rect(60, 270, 20, 10);
rect(90, 270, 20, 10);
rect(120, 270, 20, 10);
rect(150, 270, 20, 10);
```



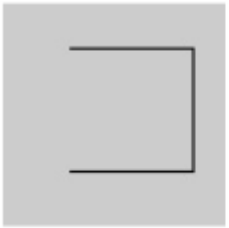
PROCESSING



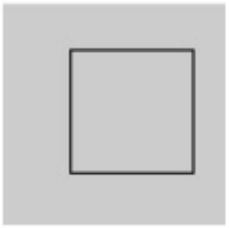
Formas

PROCESSING

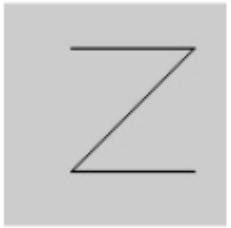
Formas:



```
noFill();
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape();
```



```
noFill();
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape(CLOSE);
```



```
noFill();
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(30, 75);
vertex(85, 75);
endShape();
```

https://processing.org/reference/

PROCESSING

Formas:

```
fill(0);
noStroke();
smooth();
beginShape();
vertex(10, 0);
vertex(100, 30);
vertex(90, 70);
vertex(100, 70);
vertex(10, 90);
vertex(50, 40);
endShape();
```

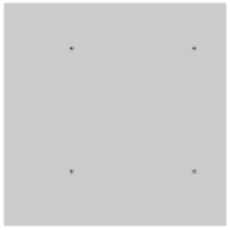
```
noFill();
smooth();
strokeWeight(20);
beginShape();
vertex(52, 29);
vertex(74, 35);
vertex(60, 52);
vertex(61, 75);
vertex(40, 69);
vertex(19, 75);
endShape();
```

```
noStroke();
fill(0);
beginShape();
vertex(40, 10);
for (int i = 20; i <= 100; i += 5) {
      vertex(20, i);
      vertex(30, i);
}
vertex(40, 100);
endShape();
```

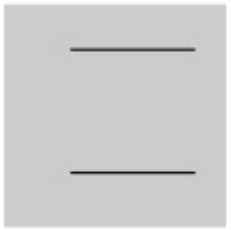
https://processing.org/reference/

PROCESSING

Formas:



```
//Dibuja un punto en cada vértice
beginShape(POINTS);
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape();
```

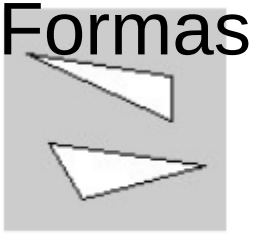


```
//Dibuja una línea ente cada par de vértices
beginShape(LINES);
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape();
```

https://processing.org/reference/

PROCESSING

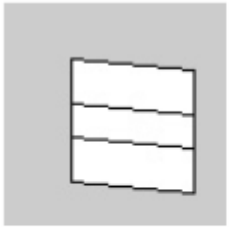
Formas:



```
//Conecta cada grupo de tres vértices
beginShape(TRIANGLES);
vertex(75, 30);
vertex(10, 20);
vertex(75, 50);
vertex(20, 60);
vertex(90, 70);
vertex(35, 85);
endShape();
```

```
beginShape(QUADS);
vertex(30, 25);
vertex(85, 30);
vertex(85, 50);
vertex(30, 45);
vertex(30, 60);
vertex(85, 65);
```

```
//Comienza con el tercer vértice, conecta todos los
//vértices subsecuentes a los dos primeros
beginShape(TRIANGLE_STRIP);
vertex(75, 30);
vertex(10, 20);
vertex(75, 50);
vertex(20, 60);
vertex(90, 70);
vertex(35, 85);
endShape();
```



```
beginShape(QUAD_STRIP);
vertex(30, 25);
vertex(85, 30);
vertex(30, 45);
vertex(85, 50);
vertex(30, 60);
vertex(85, 65);
vertex(30, 80);
vertex(85, 85);
endShape();
```

```
beginShape(TRIANGLE_FAN);
vertex(10, 20);
vertex(75, 30);
vertex(75, 50);
vertex(90, 70);
vertex(10, 20);
endShape();
```

https://processing.org/reference/

PROCESSING

Formas:

https://processing.org/reference/

```
smooth();
noFill();
beginShape();
curveVertex(20, 80);  //C1
curveVertex(20, 40);  //V1
curveVertex(30, 30);  //V2
curveVertex(40, 80);  //V3
curveVertex(80, 80);  //C2
endShape();
```

```
noFill();
beginShape();
vertex(32, 20);                          //V1
bezierVertex(80, 5,  80, 75,  30, 75);   //C1, C2, V2
endShape();
```

```
smooth();
noFill();
beginShape();
vertex(15, 30);                          //V1
bezierVertex(20, -5, 70, 5, 40, 35);     //C1, C2, V2
bezierVertex(5, 70, 45, 105, 70, 70);    //C3, C4, V3
endShape();
```

PROCESSING

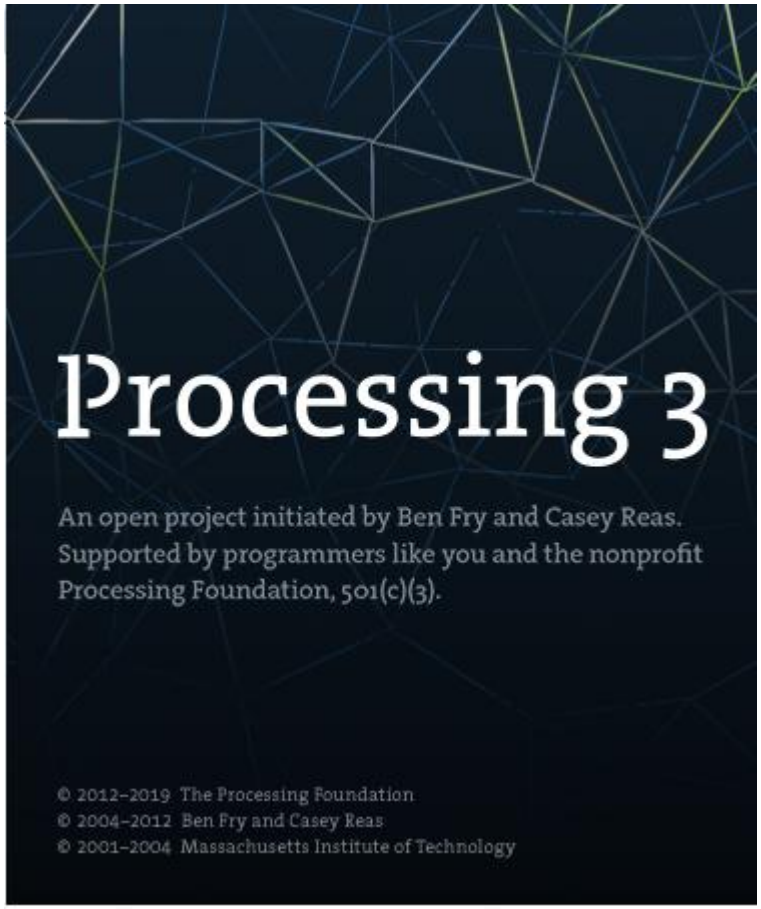
Formas:

```
smooth();
noStroke();
beginShape();
vertex(90, 39);                          //V1
bezierVertex(90, 39, 54, 17, 26, 83);    //C1, C2, V2
bezierVertex(26, 83, 90, 107, 90, 39);   //C3, C4, V3
endShape();
```

```
smooth();
noFill();
beginShape();
vertex(15, 40);                          //V1
bezierVertex(5, 0, 80, 0, 50, 55);       //C1, C2, V2
vertex(30, 45);                          //V3
vertex(25, 75);                          //V4
bezierVertex(50, 70, 75, 90, 80, 70);    //C3, C4, V5
endShape();
```

https://processing.org/reference/

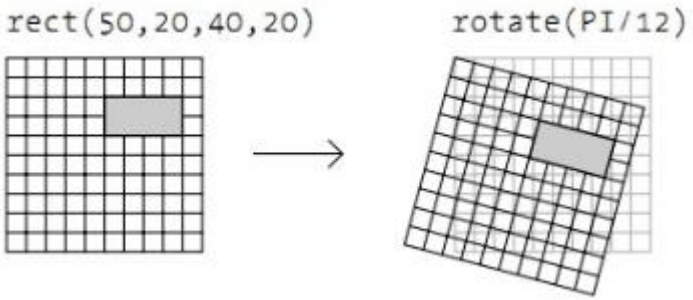
PROCESSING



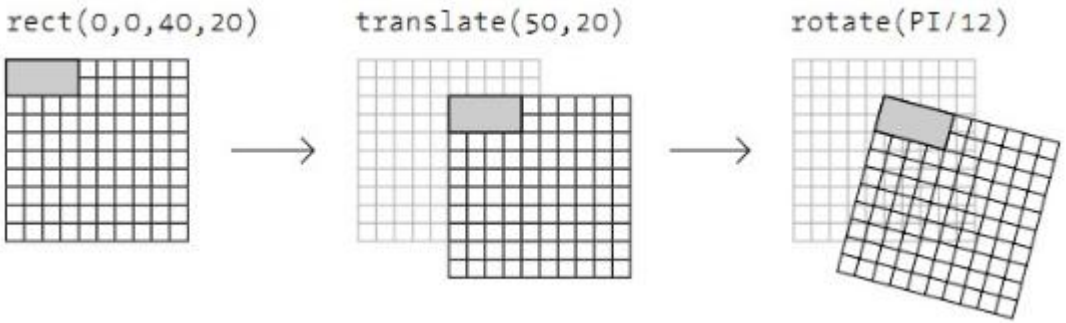
Transformaciones

PROCESSING

Traslaciones y rotaciones:



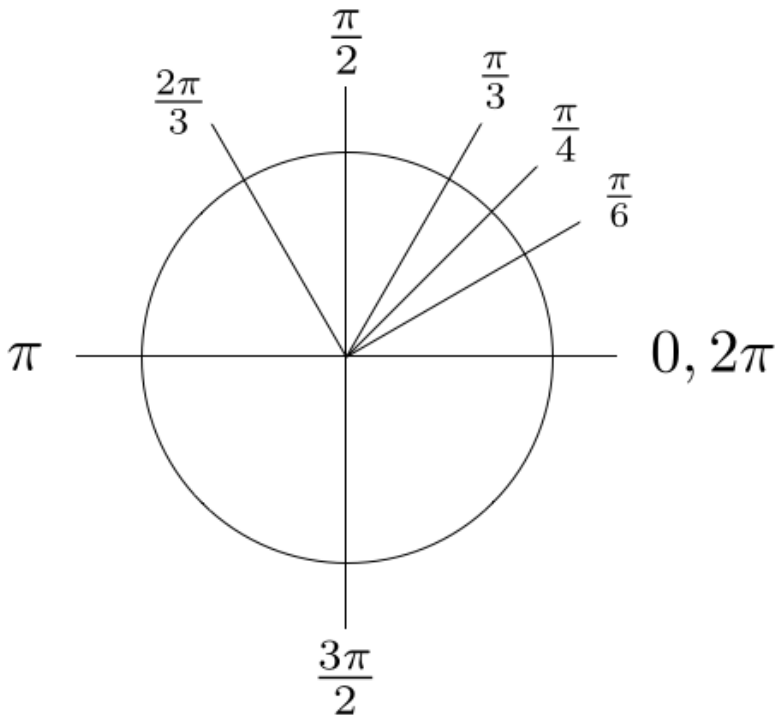
translate cambia el origen de coordenadas a una nueva posición.



rotate gira el sistema de referencia respecto al origen de coordenadas actual.

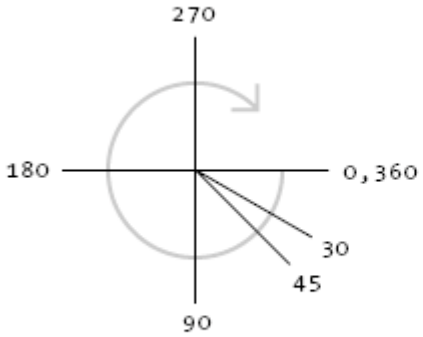
https://processing.org/reference/

PROCESSING

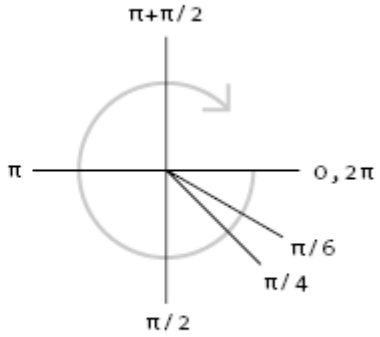


rotate (ángulo en radianes);

rotate (PI/2);
rotate (radians(90));



Degree values

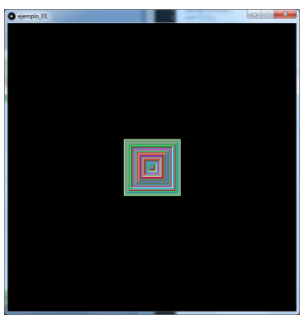


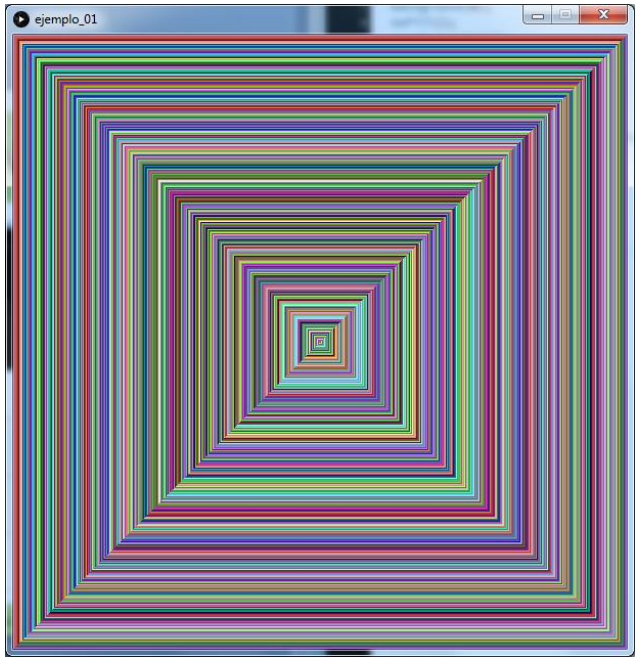
Radian values

PROCESSING

ejemplo_01

```
//Programa que dibuja cuadrados aumentando el lado

int lado=0;

void setup(){
  size (600,600);
  background(0);
  noFill();
  rectMode(CENTER);
}

void draw(){

  translate(width/2,height/2);
  stroke(random(255),random(255),random(255));
  rect(0,0,lado,lado);
  lado++;
  if (lado>width){
    lado=0;
  }

}
```

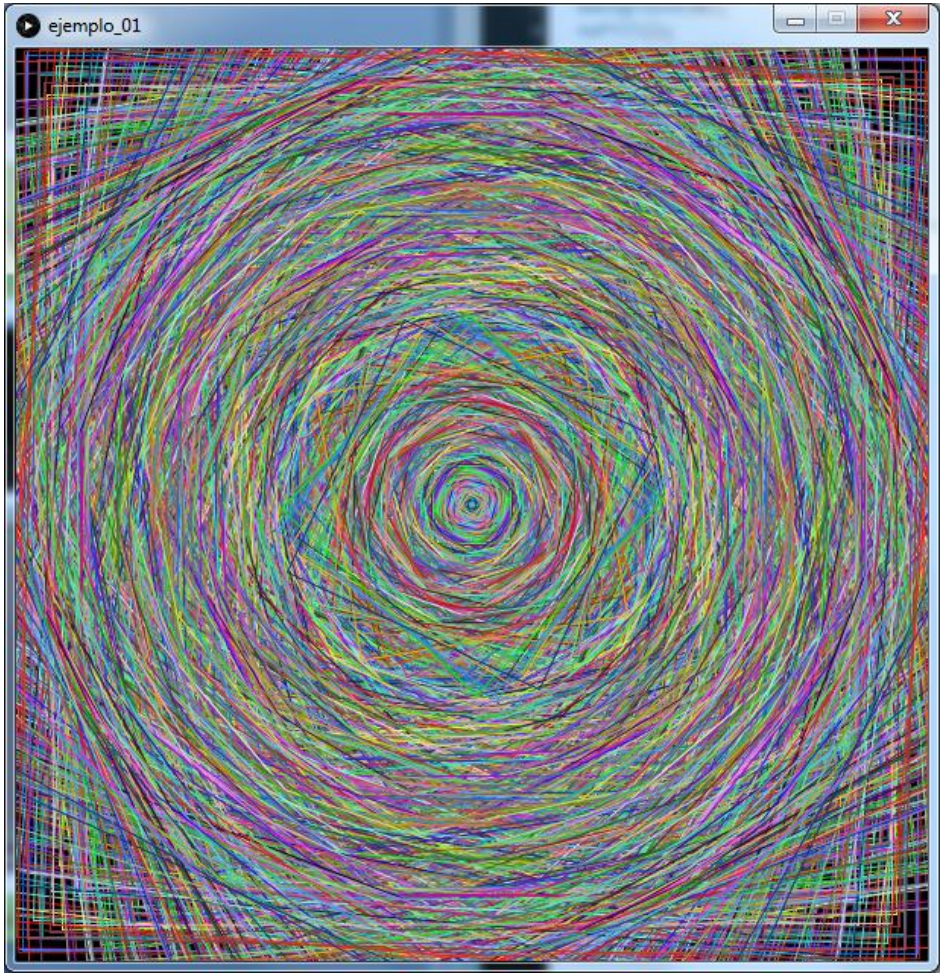




PROCESSING

ejemplo_02

```
1  //Programa que dibuja cuadrados aumentando el lado
2  //y rotándolos aleatoriamente
3
4  int lado=0;
5
6  void setup(){
7    size (600,600);
8    background(0);
9    noFill();
10   rectMode(CENTER);
11 }
12
13 void draw(){
14
15   translate(width/2,height/2);
16   rotate(random(2*PI));
17   stroke(random(255),random(255),random(255));
18   rect(0,0,lado,lado);
19   lado++;
20   if (lado>width){
21     lado=0;
22   }
23
24 }
```

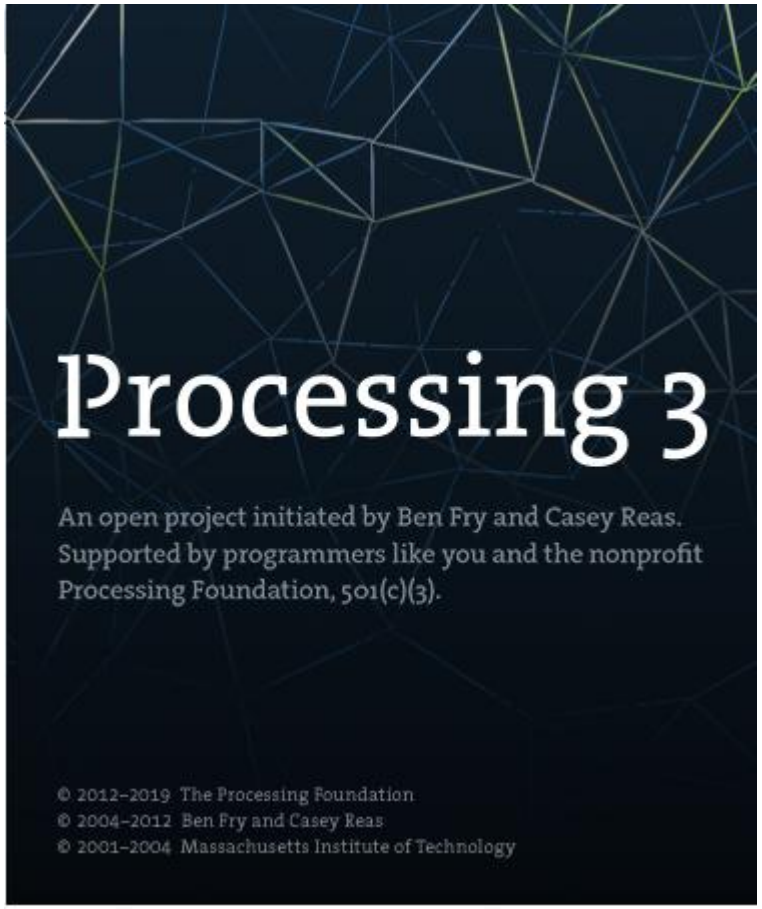


PROCESSING

ejemplo_04

```
1  //Programa que escribe un texto
2  //rotándolo aleatoriamente
3
4  void setup(){
5    size (600,600);
6    background(0);
7  }
8
9  void draw(){
10
11   rotate(random(2*PI));
12   fill(random(255),random(255),random(255));
13   textSize(30);
14   textAlign(CENTER);
15   text("José Emilio", mouseX, mouseY);
16
17 }
```

PROCESSING



Primitivas 3D

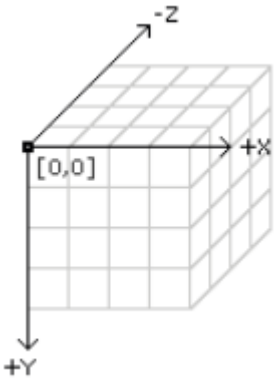
PROCESSING

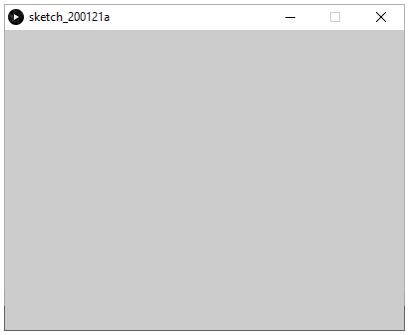
Primitivas 3D:

- box()
- sphere()
- sphereDetail()

Pantalla:

- size (ancho, alto, P3D)

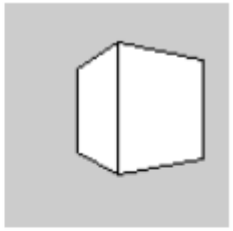




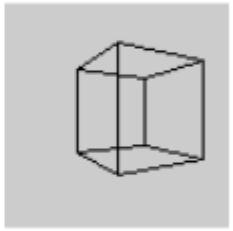
https://processing.org/reference/

PROCESSING

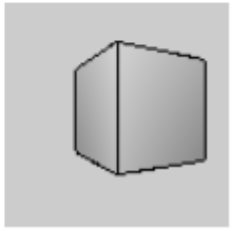
```
size(100,100,P3D);
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



```
size(100,100,P3D);
noFill( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



```
size(100,100,P3D);
lights( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



```
size(100,100,P3D);
noStroke( );
lights( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```

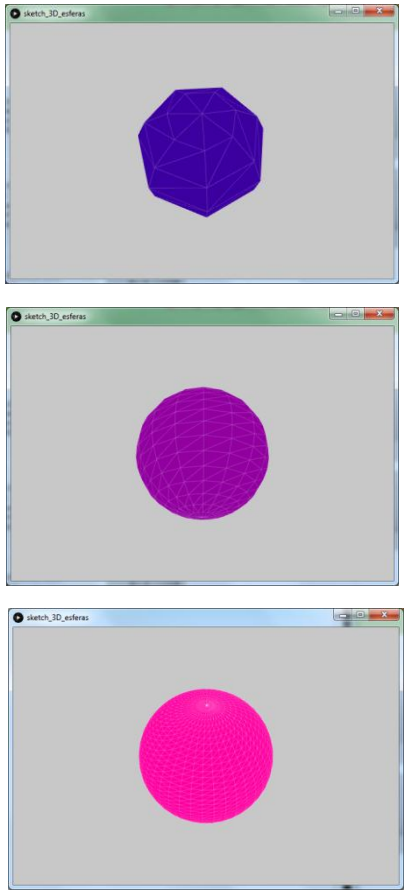
```
size(100,100,P3D);
noStroke( );
lights( );
translate(58, 48, 0);
sphere(28);
```

https://processing.org/reference/

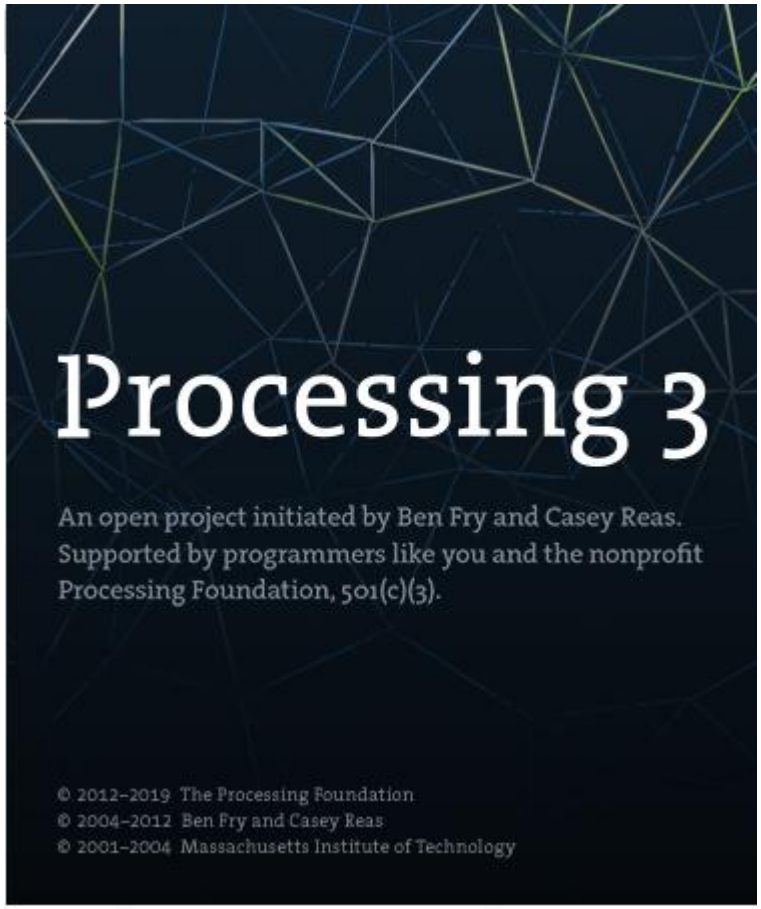
PROCESSING

Primitivas 3D (Ejemplo):

```
void setup() {
  size(600, 400, P3D);
}

void draw() {
  background(200);
  stroke(255, 50);
  translate(width/2, height/2, 0);
  rotateX(mouseY * 0.05);
  rotateY(mouseX * 0.05);
  fill(mouseX * 2, 0, 160);
  sphereDetail(mouseX / 4);
  sphere(100);
}
```



PROCESSING



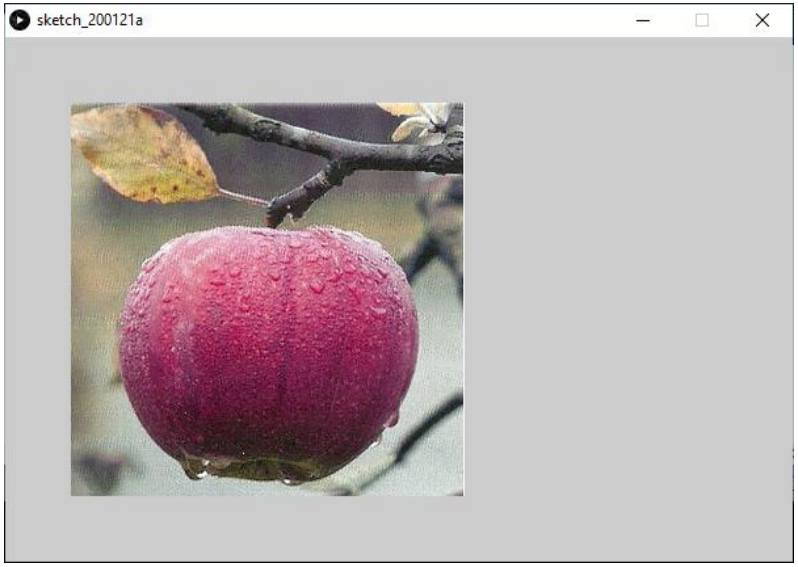
Insertar imágenes

PROCESSING

Cargar imágenes:

https://processing.org/reference/

```
size(600,400);
PImage imagen;
imagen=loadImage("manzana.jpg");
image(imagen,50,50,300,300);
```

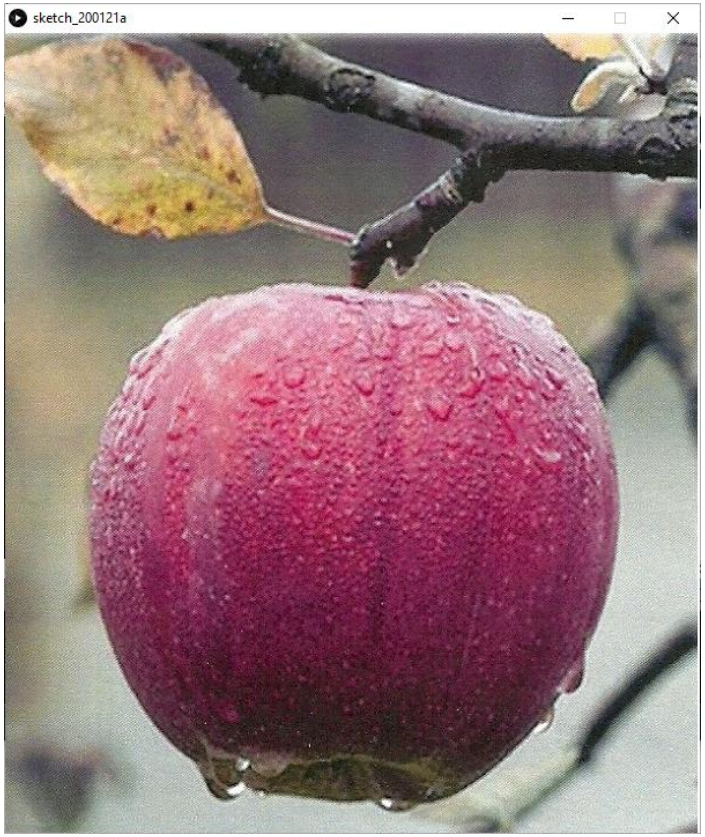


PROCESSING

Cargar imagen de fondo:

https://processing.org/reference/

```
size(601,692);

PImage imagen;
imagen=loadImage("manzana.jpg");
background(imagen);
```



PROCESSING

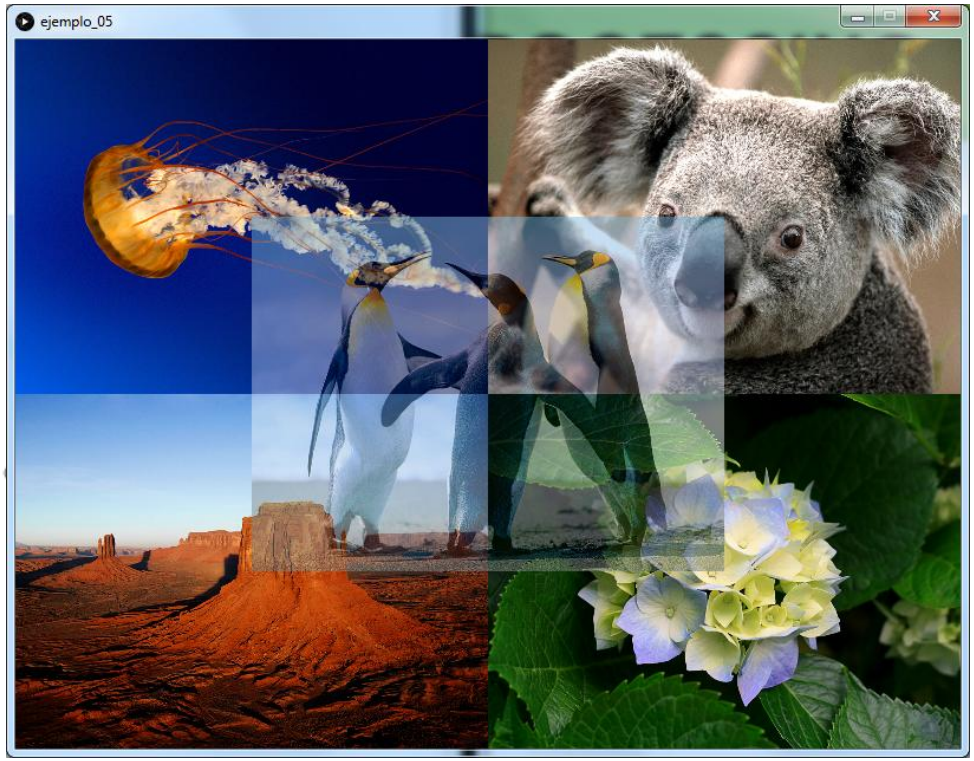
ejemplo_05

```
1  //Collage de imágenes
2
3  //Se declaran las variables
4  PImage imagen01, imagen02, imagen03, imagen04, imagen05;
5
6  size(800,600);
7
8  //Se cargan las imágenes
9  imagen01=loadImage("medusa.jpg");
10 imagen02=loadImage("koala.jpg");
11 imagen03=loadImage("desierto.jpg");
12 imagen04=loadImage("hortensias.jpg");
13 imagen05=loadImage("pinguinos.jpg");
14
15 //Se muestran las imágenes
16 image(imagen01,0,0,400,300);
17 image(imagen02,400,0,400,300);
18 image(imagen03,0,300,400,300);
19 image(imagen04,400,300,400,300);
20 image(imagen05,200,150,400,300);
```

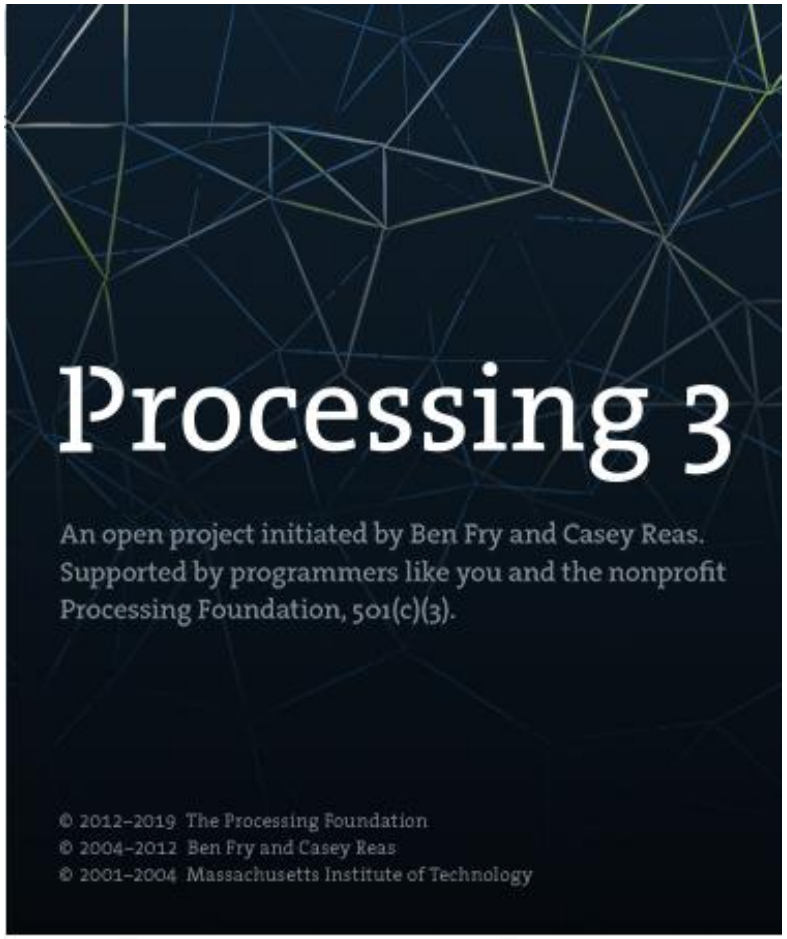
PROCESSING

ejemplo_05

```
1  //Collage de imágenes
2
3  //Se declaran las variables
4  PImage imagen01, imagen02, imagen03, imagen04, imagen05;
5
6  size(800,600);
7
8  //Se cargan las imágenes
9  imagen01=loadImage("medusa.jpg");
10 imagen02=loadImage("koala.jpg");
11 imagen03=loadImage("desierto.jpg");
12 imagen04=loadImage("hortensias.jpg");
13 imagen05=loadImage("pinguinos.jpg");
14
15 //Se muestran las imágenes
16 image(imagen01,0,0,400,300);
17 image(imagen02,400,0,400,300);
18 image(imagen03,0,300,400,300);
19 image(imagen04,400,300,400,300);
20 tint(255, 126);  // Se aplica transparencia con cambio
21 image(imagen05,200,150,400,300);
22
```



PROCESSING



Insertar texto

https://processing.org/reference/

PROCESSING

Tipografía:

PFont f = loadFont("Bodoni-Italic.vlw" **);** carga esa fuente dentro de la variable f.

textFont(f, size**);** determina la fuente actual y su tamaño antes de dibujar el texto.

text("handglove", x, y**);** renderiza el texto en el lugar.

```
size(200,100);
background(#FFFFFF);
fill(#000000);
PFont f = loadFont("Bodoni-Italic.vlw");
textFont(f, 50);
text("handglove", 14, 60);
```

Importante: Al igual que con las imágenes, para utilizar una determinada tipografía, antes hay que importarla ala carpeta data del sketch.

Para ello hay que utilizar la opción de menú **Tools -> Create Font…** para copiar el archivo de fuentes a la carpeta data.

https://processing.org/reference/

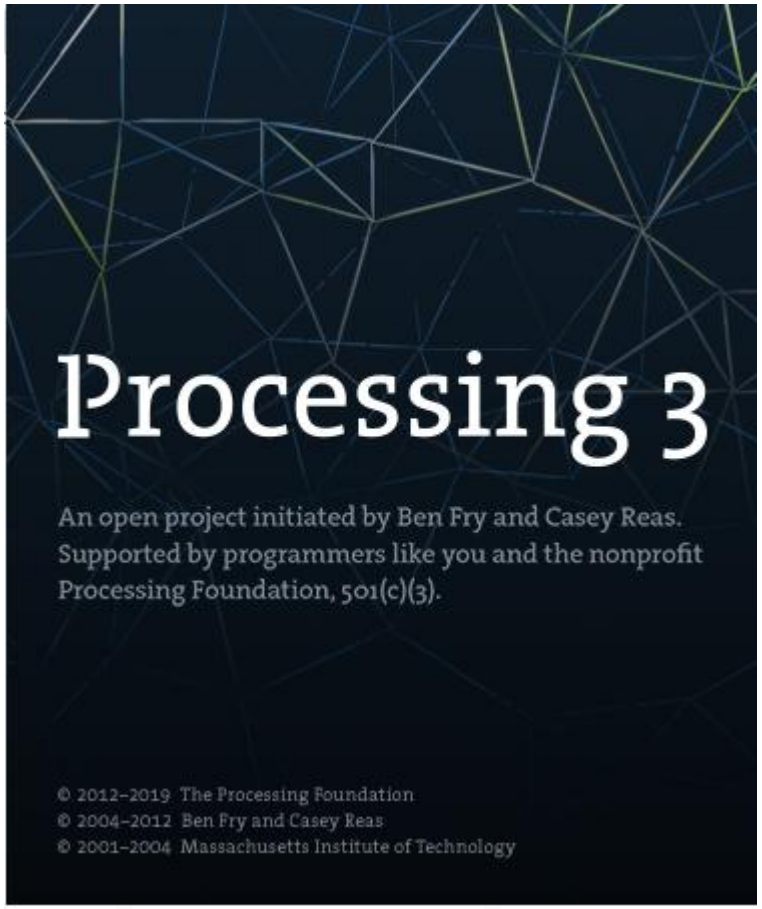
PROCESSING

Tipografía:

```
size(200,100);
background(#FFFFFF);
fill(#000000);
PFont f = loadFont("Bodoni-Italic.vlw");
textFont(f, 50);
text("handglove", 14, 60);
```


```
size(200,100);
noStroke( );
PFont f = loadFont("Univers66.vlw");
textFont(f, 50);
fill(#FFFFFF);
ellipse(-50,-55,150,150);
fill(#CC6600);
for(int i=0;i<20;i++){
    rotateZ(0.2);
    text("dizzy", 90,0);
}
```

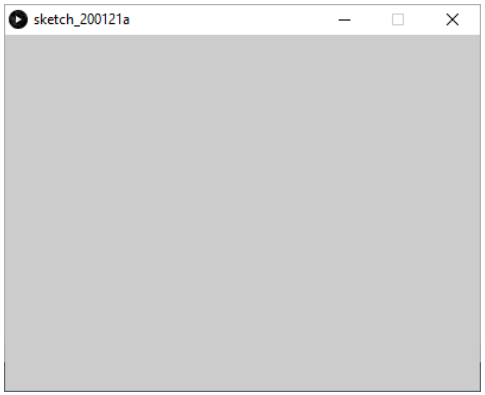
PROCESSING



Animaciones

PROCESSING

Animaciones (Métodos setup y draw):




```
// Zona de declaración de variables

void setup () {
   // Se ejecuta una vez al iniciarse el programa
}

void draw() {
   /* Se repite hasta que se sale de la aplicación o se
   cierra la ventana */
}
```

https://processing.org/reference/

PROCESSING

Animaciones (Métodos setup y draw):

```
void setup()
{
      // instrucciones de inicialización
}


void draw()
{
      // instrucciones de dibujo
}
```

← Estas instrucciones sólo se ejecutan una vez. Algunas instrucciones de setup pueden ser: size() (tamaño de la pantalla), background(), etc.

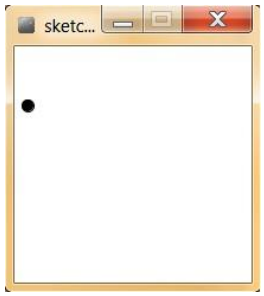
← En este bloque del programa van las instrucciones cuya ejecución se repite hasta que el programa llega a su fin o se cierra la ventana.

https://processing.org/reference/

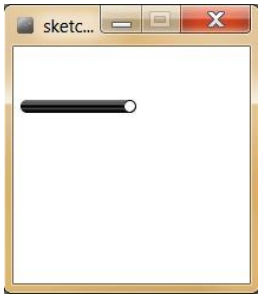
PROCESSING

Animaciones (Ejemplo):

```
int a;
a = 10;
size(200,200);
background(255);
fill(0);
a = (a+1) % 200;
ellipse(a,50,10,10);
```




```
int a;
void setup()
{
	a = 10; size(200,200);
	background(255);
}
void draw()
{
	a = (a+1) % 200;
	ellipse(a,50,10,10);
}
```



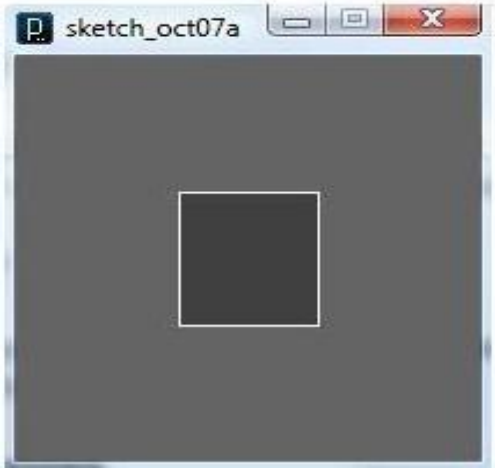
PROCESSING

Animaciones (Ejemplo):

```
int x = 0;

void setup() {
  size(200, 200);
  background(0);
  noStroke();
  fill(255,0,0);
}

void draw() {
  rect(x, 10, 2, 80);
  x = x + 1;
}
```

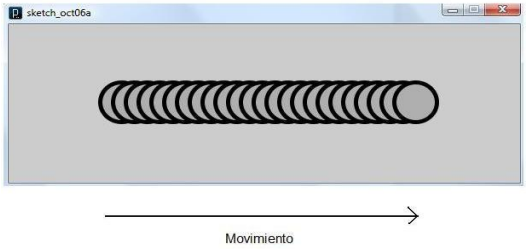
PROCESSING

Animaciones (Ejemplo):

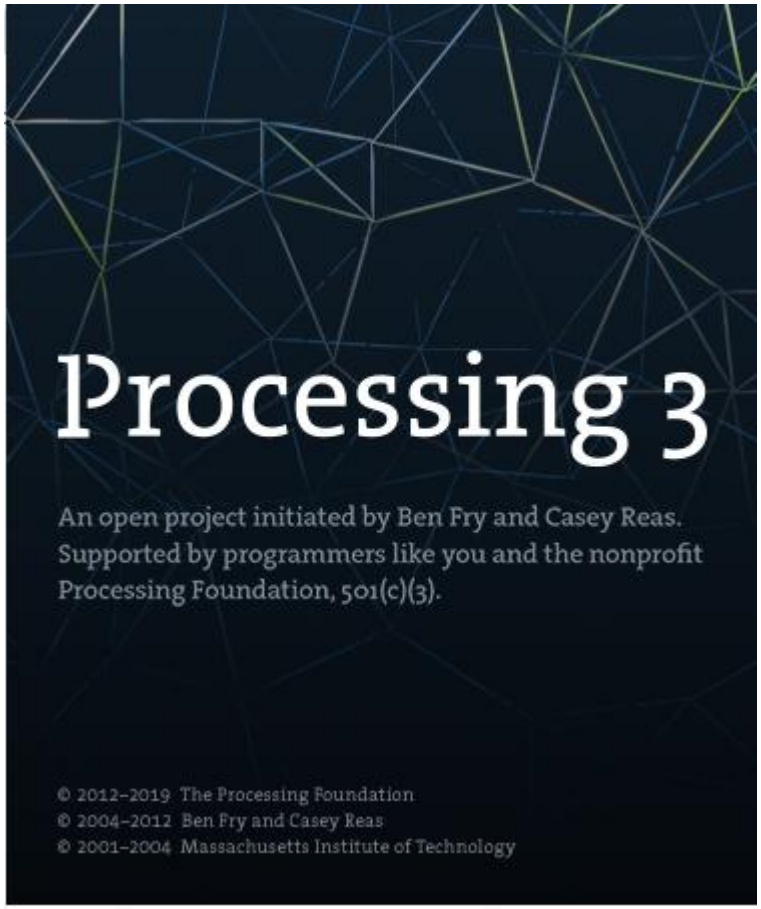


PROCESSING

Animaciones (Ejemplo):



PROCESSING



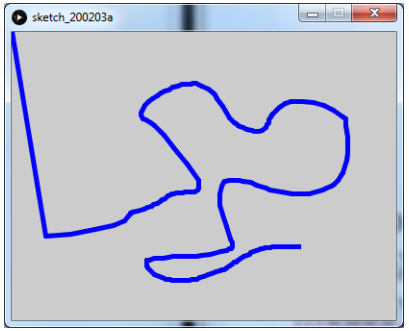
Ratón

PROCESSING

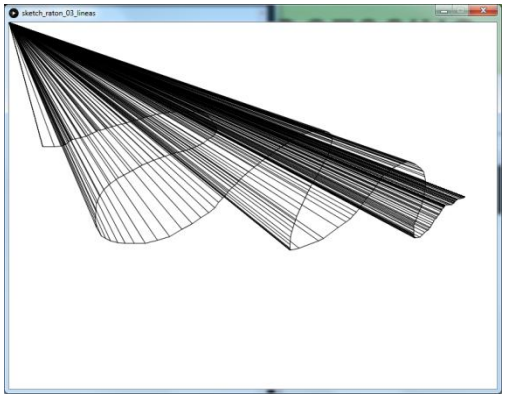
Ratón (coordenadas):

- mouseX
- mouseY
- pmouseX
- pmouseY

```
void setup(){
  size (400,300);
}

void draw(){
  strokeWeight(5);
  stroke(0, 0, 255);
  line (pmouseX, pmouseY, mouseX, mouseY);
}
```




```
int x, y;
void setup(){
  size(800,600);
  background(255);
}

void draw(){
  line(0, 0, mouseX, mouseY);
  line(x, y, mouseX, mouseY);
  x = mouseX;
  y = mouseY;
}
```



https://processing.org/reference/

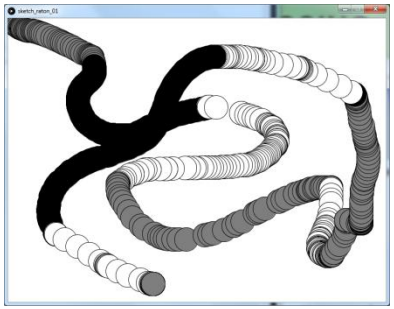
PROCESSING

Ratón (eventos):

- mouseButton
- mouseClicked()
- mouseDragged()
- mouseMoved()
- mousePressed()
- mousePressed
- mouseReleased()
- mouseWheel()

Ratón (coordenadas):

- mouseX
- mouseY
- pmouseX
- pmouseY

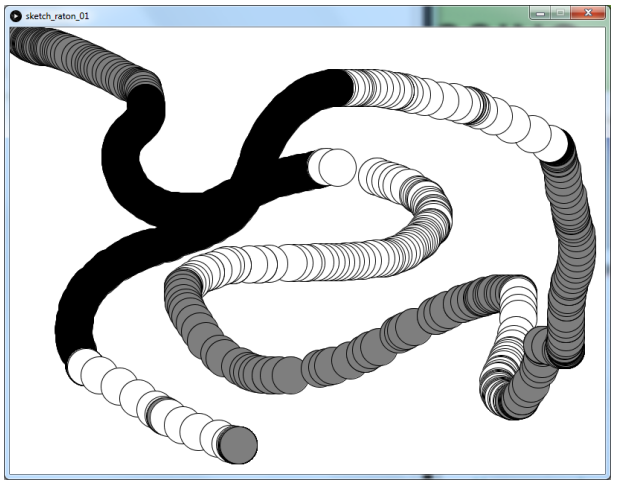


https://processing.org/reference/

PROCESSING

Ratón (Ejemplo):

```
void setup(){
  size (800,600);
  background (255);
  ellipseMode (CENTER);
}

void draw(){
  if (mousePressed && (mouseButton == LEFT)) {
    fill (0);
  } else if (mousePressed && (mouseButton == RIGHT)) {
    fill (255);
  } else {
    fill (126);
  }
  circle (mouseX, mouseY,50);
}
```



https://processing.org/reference/

PROCESSING

Ratón (Ejemplo):

```
void draw( ) {
   background(190);
   rect(mouseX-5, mouseY-5, 10, 10);
}

void mousePressed( ) {
   fill(0);
}
void mouseReleased( ) {
   fill(255);
}
```

En este sencillo ejemplo, un cuadrado es dibujado por donde el ratón vaya.

Si aprietas el ratón el cuadrado se tornará negro.

```
void draw( ) {
   if(keyPressed) {
      fill(102, 0, 0);
   } else {
      fill(204, 102, 0);
   }
   rect(30, 20, 55, 55);
}
```

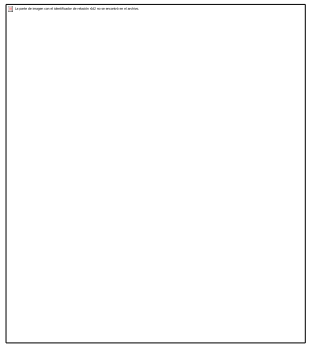
En este simple ejemplo el cuadrado cambia a rojo oscuro si cualquier tecla está siendo presionada. No hay necesidad de redibujar de fondo!

https://processing.org/reference/

PROCESSING

Ratón (Ejemplo):

```
int x, y;

void setup()
{
	size(200,200);
	background (255);
}

void draw()
{
	fill (255, 0, 0);
	x=mouseX-25;
	y=mouseY-25;
	ellipse (x, y, 50, 50);
}
```

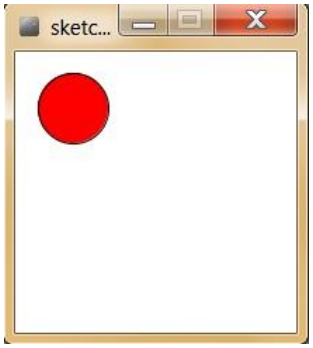


https://processing.org/reference/

PROCESSING

Ratón (Ejemplo):

```
int;
void setup()
{
	size (200,200);
}

void draw()
{
	background (255);
	fill (255, 0, 0);
	x=mouseX-25;
	y=mouseY-25;
	ellipse (x, y, 50, 50);
}
```



https://processing.org/reference/

PROCESSING

Ratón (Ejemplo):

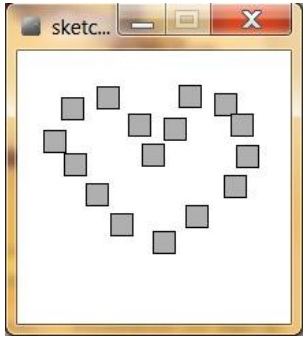
```
void setup()
{
	size(200, 200);
	background(255);
}

void draw()
{
	stroke(0);
	Line (pmouseX, pmouseY, mouseX, mouseY);
}
```

https://processing.org/reference/

PROCESSING

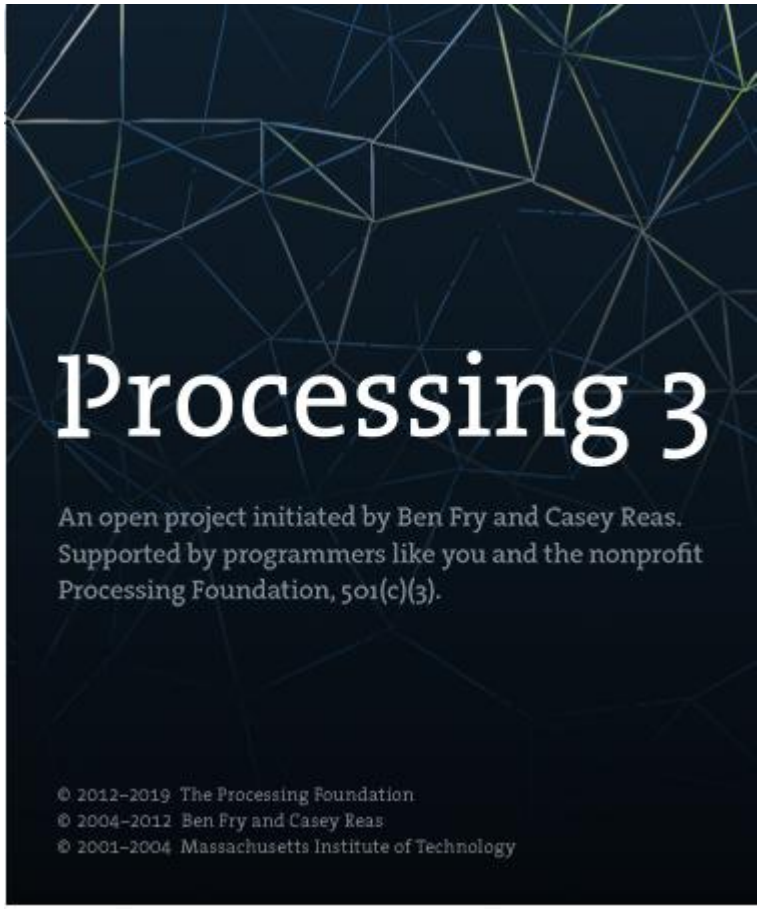
Ratón (Ejemplo):

```
void setup()
{
	size(200,200); background(255);
}

void draw()
{
}

void mousePressed()
{
	stroke (0);
	fill (175);
	rectMode (CENTER);
	rect(mouseX,mouseY,16,16);
}
```



https://processing.org/reference/

PROCESSING

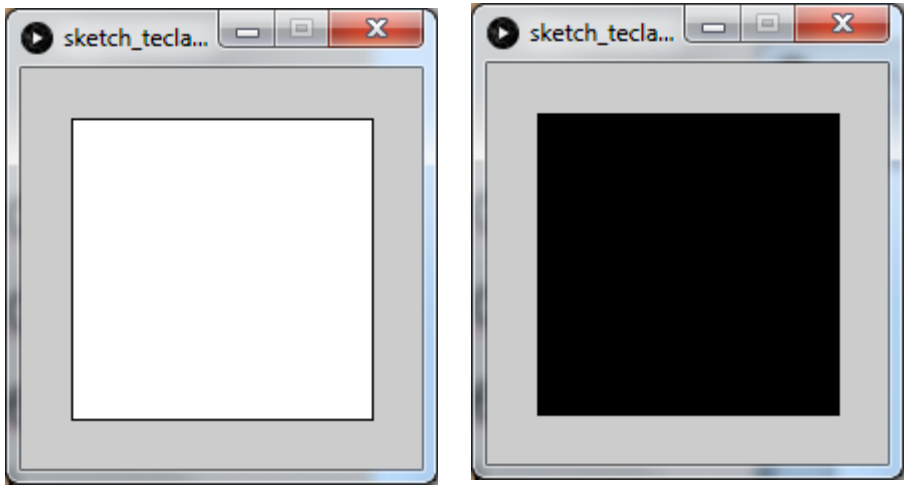


Teclado

PROCESSING

Teclado:

- key
- keyCode
- keyPressed()
- keyPressed
- keyReleased()
- keyTyped()

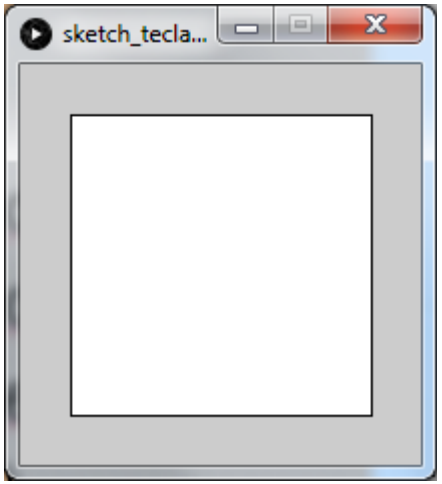


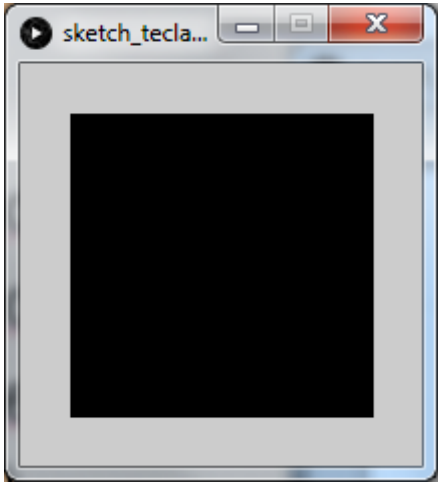
https://processing.org/reference/

PROCESSING

Teclado (Ejemplo):

```
void setup() {
  size(200,200);
}
color fillVal = color(126);

void draw() {
  fill(fillVal);
  rect(25, 25, 150, 150);
}

void keyPressed() {
  if (key == CODED) {
    if (keyCode == UP) {
      fillVal = 255;
    } else if (keyCode == DOWN) {
      fillVal = 0;
    }
  }
}
```



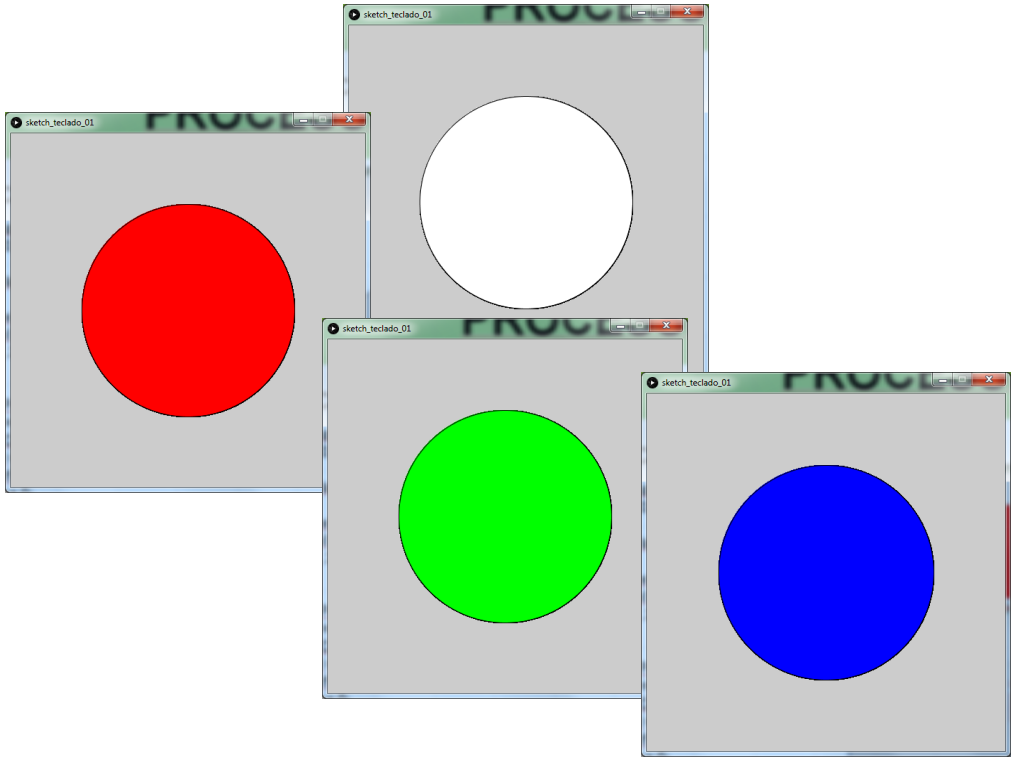


https://processing.org/reference/

PROCESSING

Teclado (Ejemplo):

```
void setup()
{
  size(500,500);
}

void draw()
{
  if (key=='r') //si pulso la tecla r
  {
    fill(255,0,0); //relleno de color ROJO
  }
  if (key=='g') //si pulso la tecla g
  {
    fill(0,255,0); //relleno de color VERDE
  }
  if (key=='b') //si pulso la tecla b
  {
    fill(0,0,255); //relleno de color AZUL
  }
  ellipse(250,250,300,300);
}
```



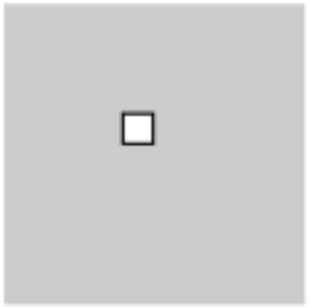
https://processing.org/reference/

PROCESSING

Teclado (Ejemplo):

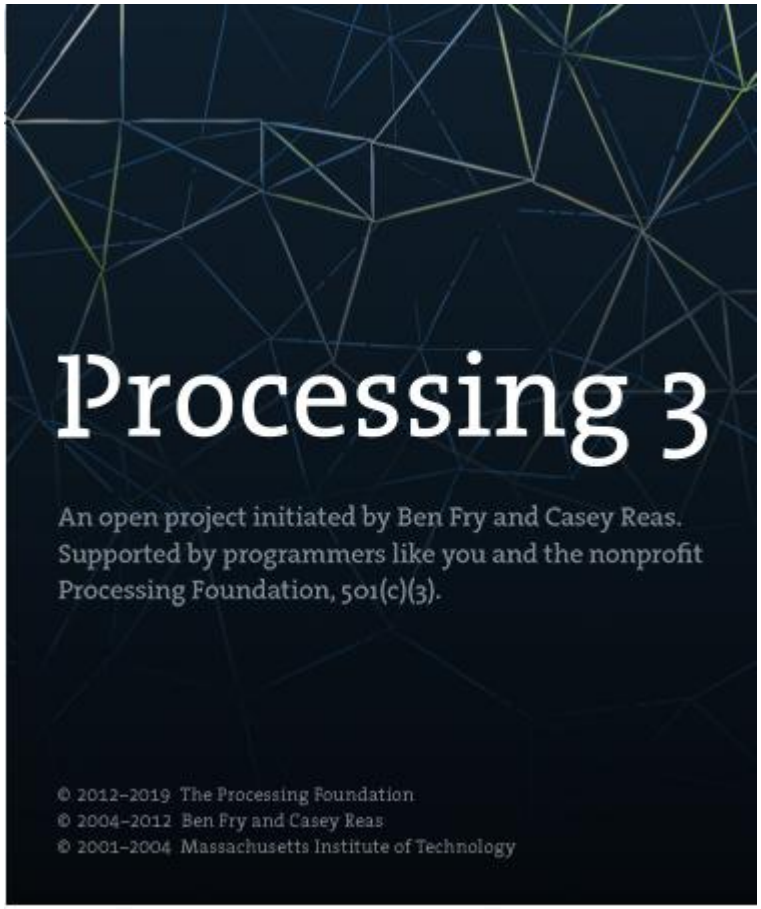
```
int x = 50;
int y = 50;

void draw( ){
   background(190);
   rect(x,y,10,10);
}

void keyPressed( ){
   if(key=='w'||key=='W'){
      y--;
   }else if(key=='s'||key=='S'){
      y++;
   }else if(key=='a'||key=='A'){
      x--;
   }else if(key=='d'||key=='D'){
      x++;
   }
}
```

En este ejemplo, las teclas del teclado moverán el cuadrado alrededor.



https://processing.org/reference/

PROCESSING



Estructuras de programación

PROCESSING

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (if y switch) e iteración (bucles for y while).

Estructuras básicas:

- Estructura secuencial.
- Estructura condicional.
- Estructura repetitiva o iterativa.

https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada

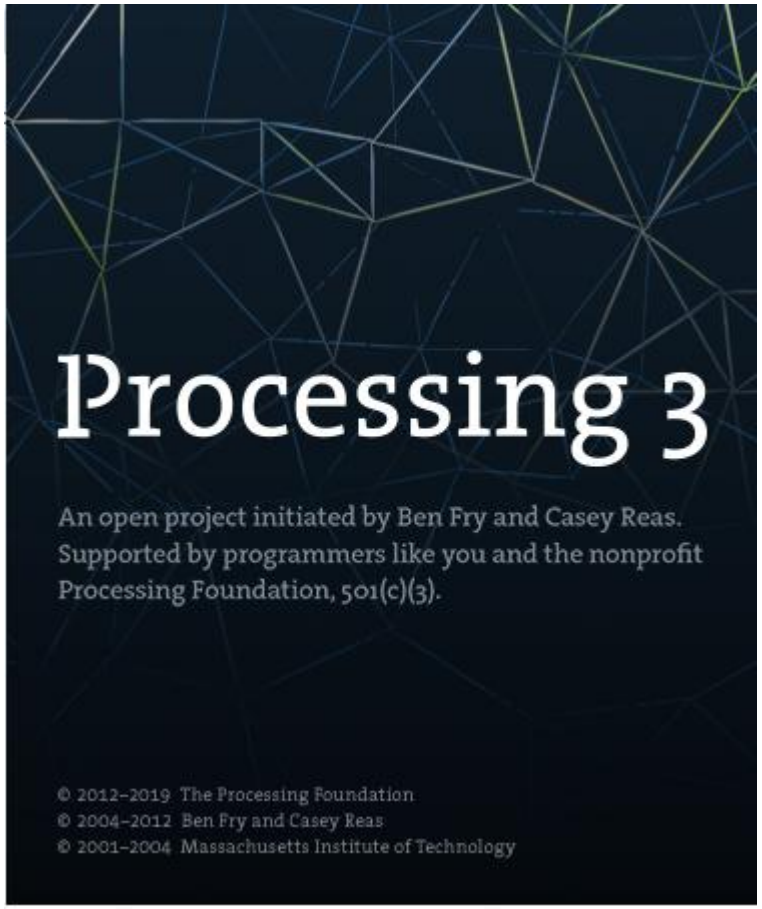
PROCESSING

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (if y switch) e iteración (bucles for y while).

Estructuras básicas:

- Estructura secuencial.
- Estructura condicional:
  - if
  - switch
- Estructura repetitiva o iterativa:
  - while
  - do while
  - for

PROCESSING

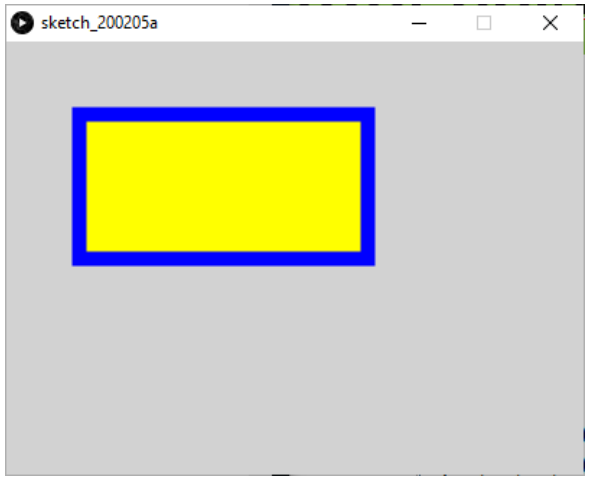


Estructura secuencial

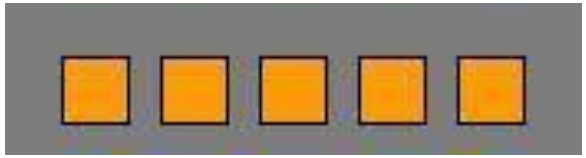
PROCESSING

Estructura secuencial:

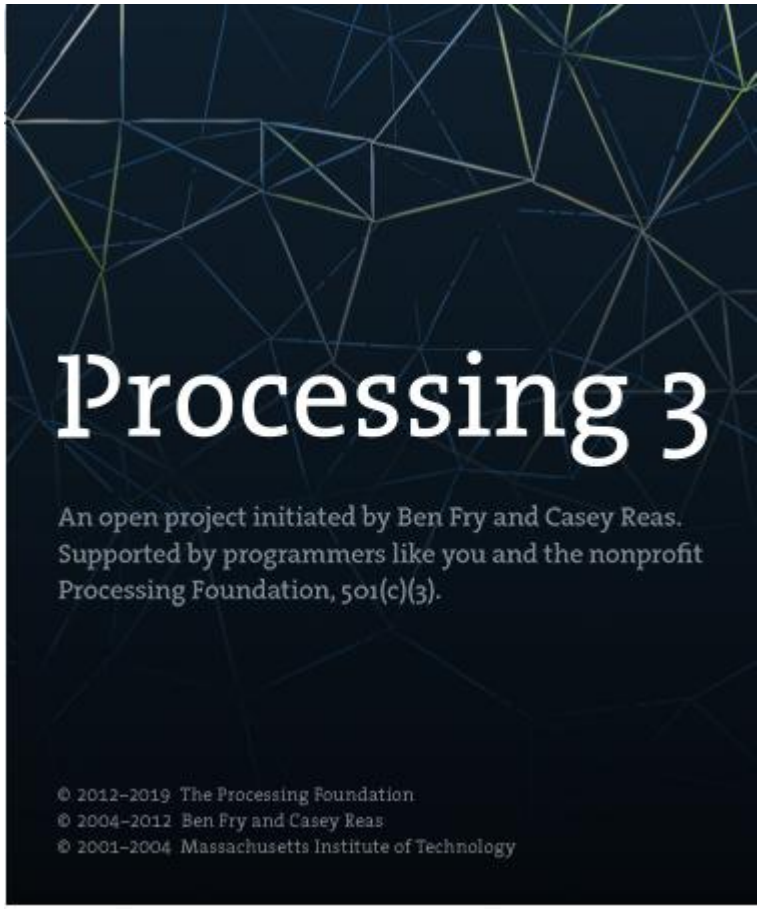
```
size (400, 300);          //Tamaño de la pantalla
background (255);         //Color de fondo
fill (255, 255, 0);       //Color de relleno
stroke (0, 0, 255);       //Color del borde
strokeWeight (10);        //Grosor del borde
rect (50,50, 200,100);    //Dibujar rectángulo
```




```
rect (20,20,20,20);
rect (50,20,20,20);
rect (80,20,20,20);
rect (110,20,20,20);
rect (140,20,20,20);
```



PROCESSING



El cursor

PROCESSING

Cursor (tipos):

- cursor (ARROW)
- cursor (CROSS)
- cursor (HAND)
- cursor (MOVE)
- cursor (TEXT)
- cursor (WAIT)
- noCursor()

Ocultar el cursor:

- noCursor();

Mostrar el cursor:

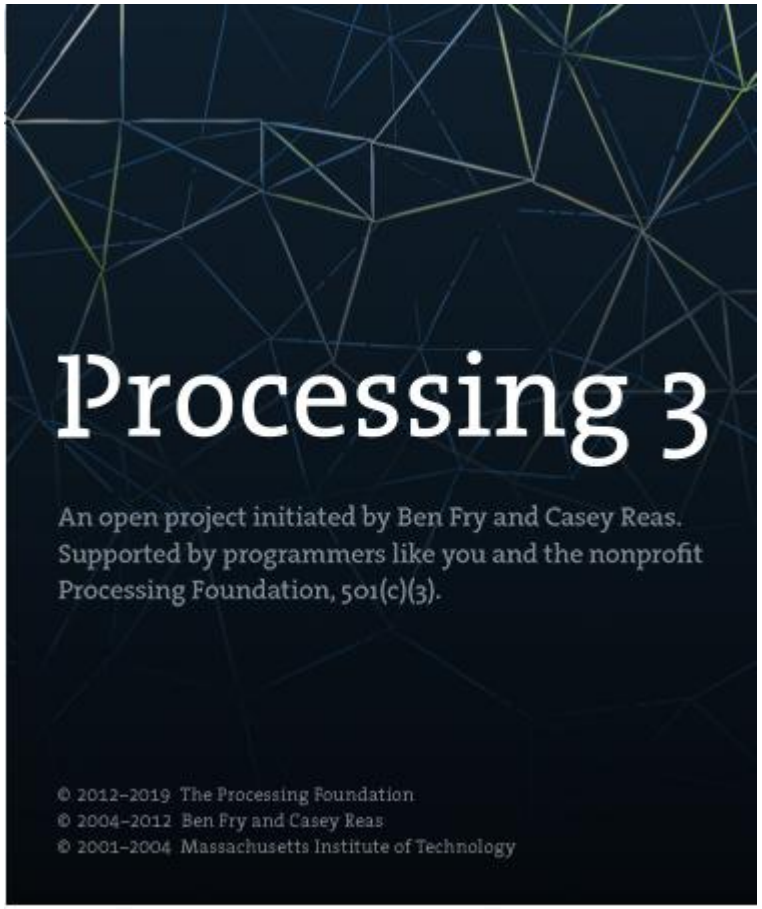
- cursor (tipo de cursor);

```
size(400,300);
cursor(CROSS);
```

```
void setup() {
  size(100, 100);
}

void draw() {
  if (mouseX < 50) {
    cursor(CROSS);
  } else {
    cursor(HAND);
  }
}
```

https://processing.org/reference/

PROCESSING



Fecha y hora

PROCESSING

Fecha y hora:

- hour()
- minute ()
- second ()
- millis()
- year()
- month()
- day()

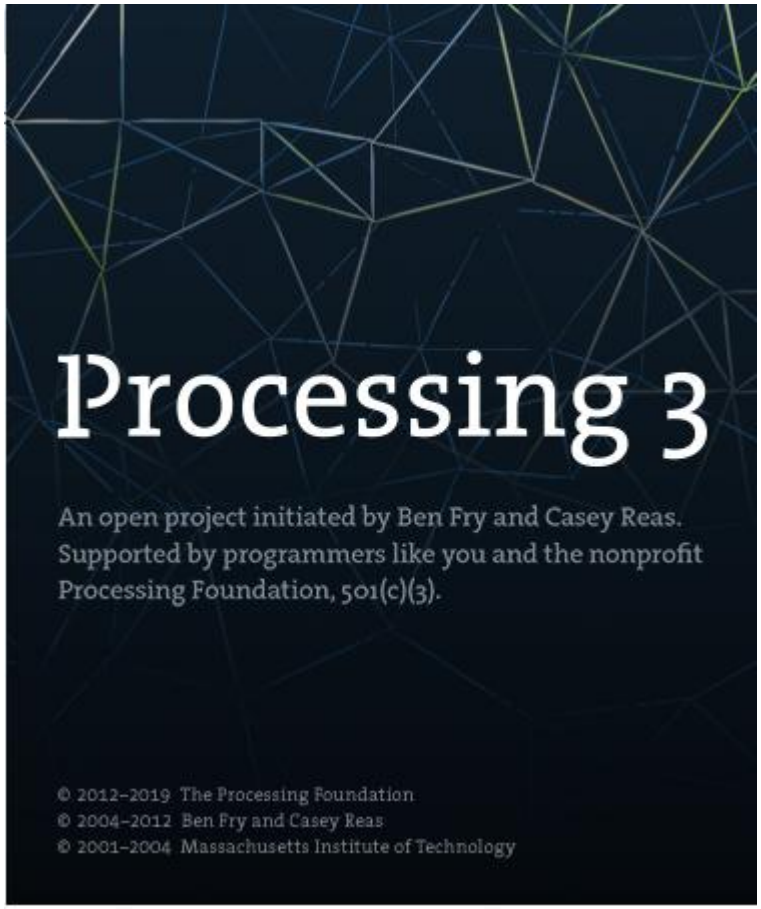
millis() devuelve el número de milisegundos (milésimas de segundo) desde que inició el programa. Esta información se usa a menudo para cronometrar eventos y secuencias de animación.

ej_06_reloj_01

```
1  PFont font;
2  String tiempo;
3
4  void setup(){
5
6    size(600,400);
7    font=loadFont("AgencyFB-Bold-48.vlw");
8    textFont(font, 32);
9
10 }
11
12 void draw(){
13
14   background(222);
15   //Dibujar un rectángulo alrededor de la hora
16   rectMode(CENTER);
17   fill(255,0,0);
18   strokeWeight(5);
19   strokeJoin(ROUND);
20   rect(width/2, height/2,300,100);
21
22   //Mostrar la hora
23   tiempo=str(hour()) + ':' + str(minute()) + ':' + str(second());
24   fill(255,255,255);
25   textAlign(CENTER, CENTER);
26   textSize(72);
27   text(tiempo, width/2, height/2);
28
29 }
30
```

https://processing.org/reference/

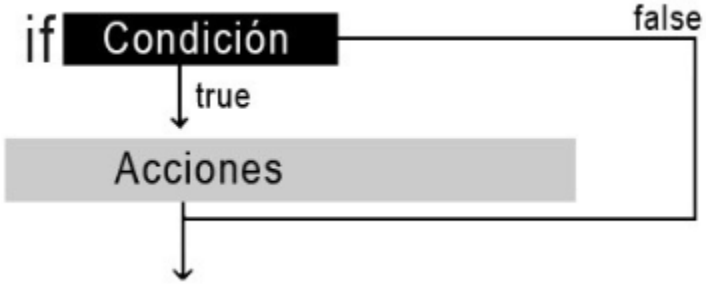
PROCESSING



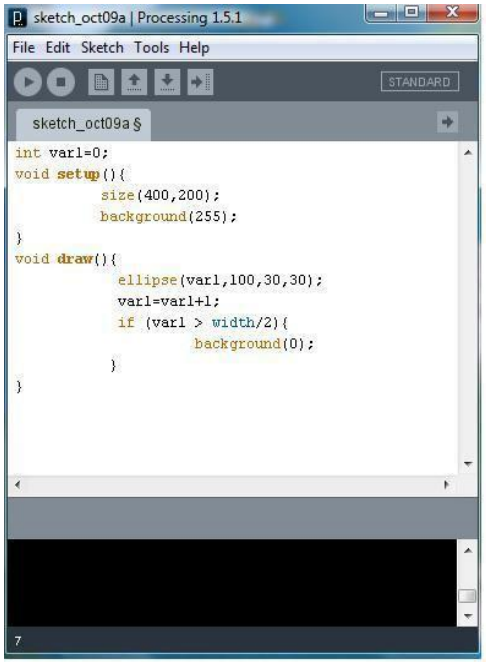
Estructura condicional

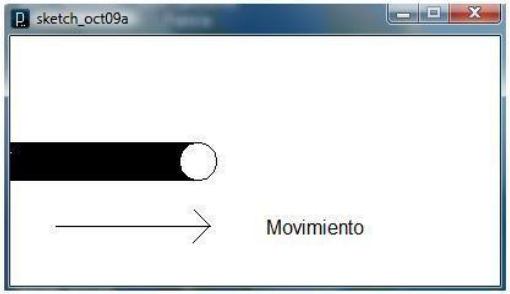
PROCESSING

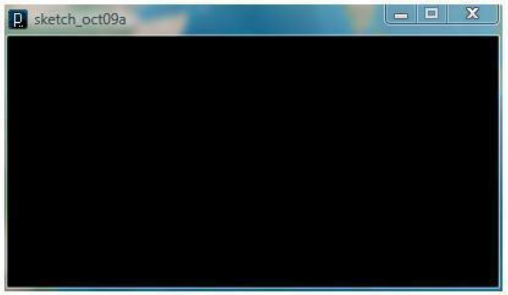
Estructura condicional (1):




```
if (condición) {
    …
}
```





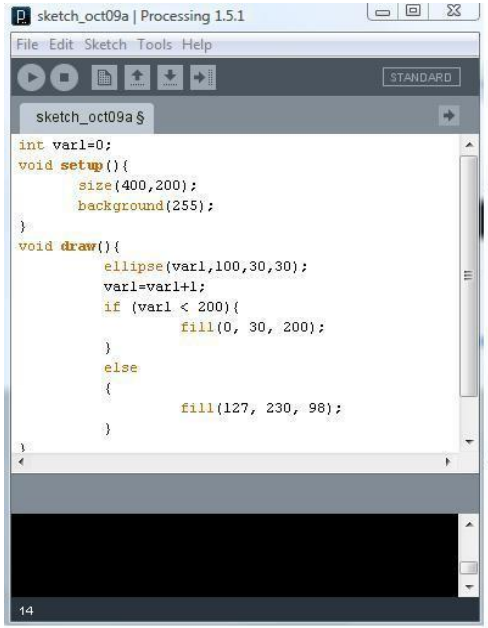


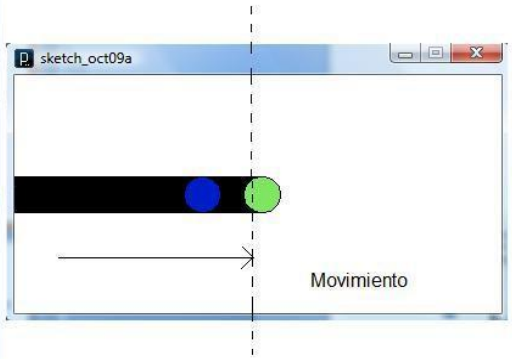

```
if (expresión booleana){
    /*Este conjunto de
    instrucciones se ejecutan si
    la condición es verdadera*/
}
```

PROCESSING

Estructura condicional (2):

```
if (condición) {
        ...
}
else {
        ...
}
```

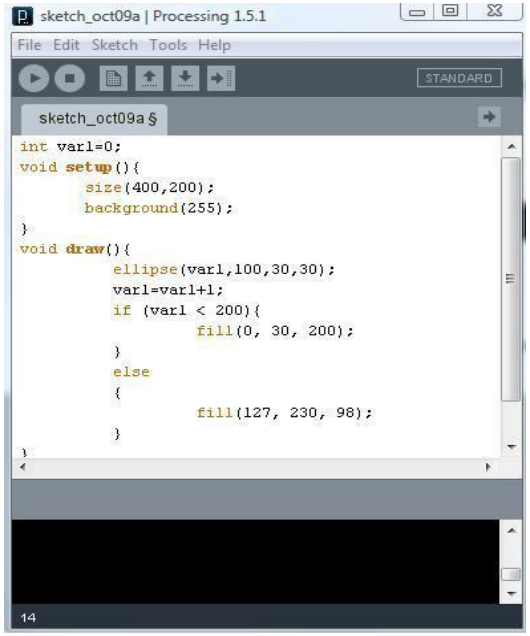


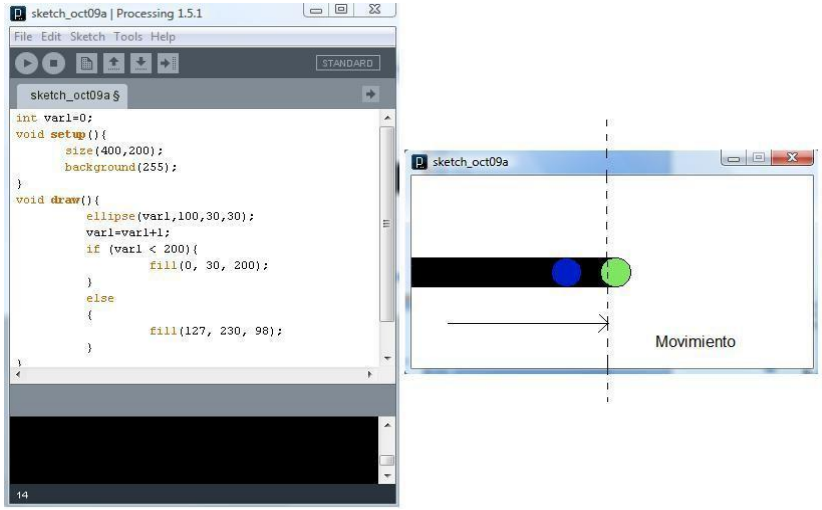


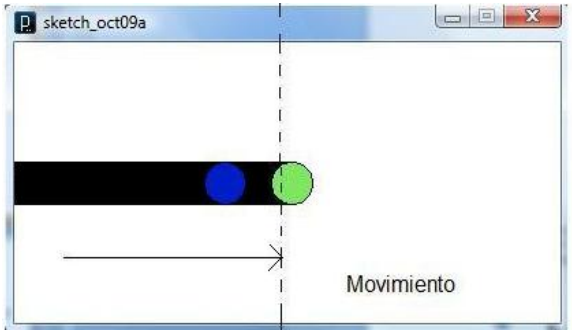
PROCESSING

Estructura condicional (2):

```
if (expresión booleana){
        /* Código que se ejecuta si la condición es
        verdadera*/
} else {
        /* Código que se ejecuta si la condición es
        falsa*/
}
```



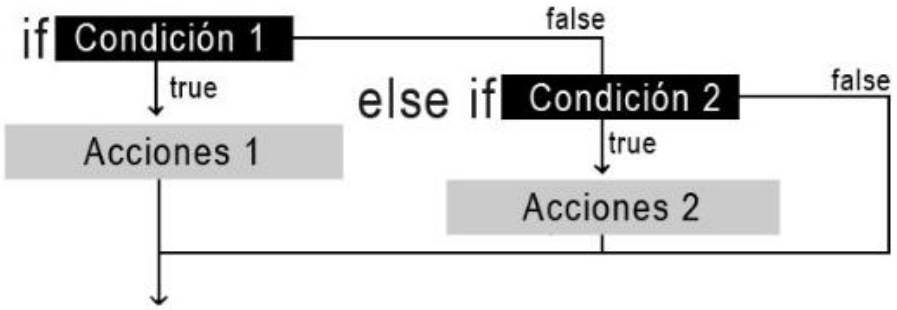


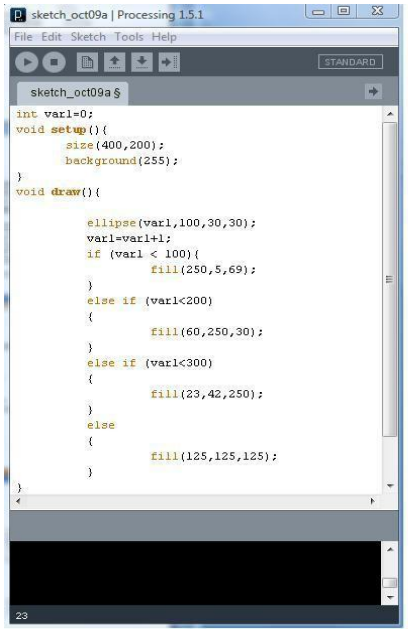


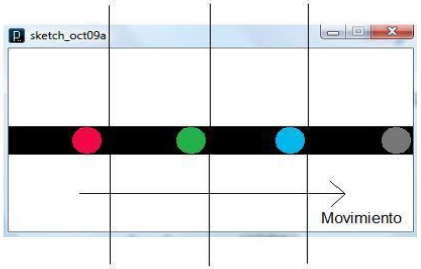
PROCESSING

Estructura condicional (3):

```
if (condición 1) {
        ...
}
else if (condición 2) {
        …
}
else {
        ...
}
```



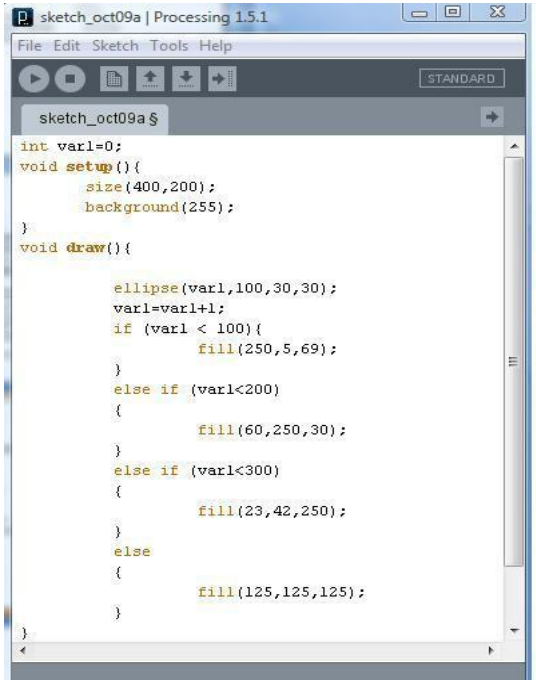


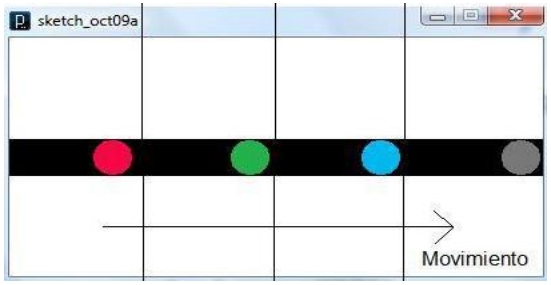


PROCESSING

Estructura condicional (3):

```
if (expresión booleana 1){

        // El código que se ejecuta si la condición 1 es verdadera
}
else if (expresión booleana 2){

        // El código que se ejecuta si la condición 2 es verdadera
}
else if (expresión booleana n){

        // El código que se ejecuta si la condición n es verdadera
}
else {
        // Ejecución si ninguna de las condiciones anteriores
        // fueron verdaderas
}
```





PROCESSING

Estructura condicional (Ejemplo):

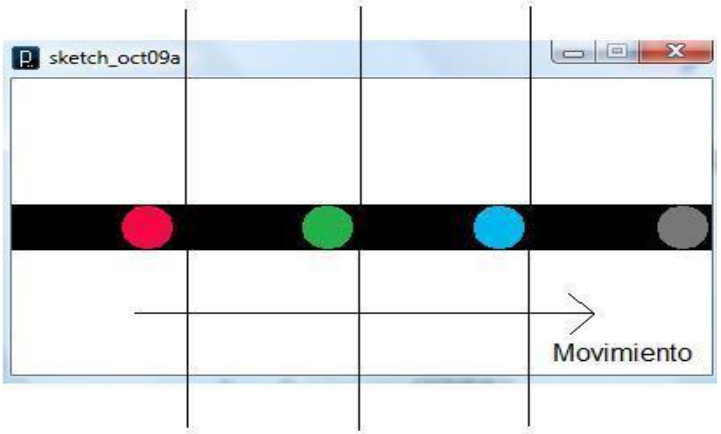
sketch_oct09a | Processing 1.5.1

File Edit Sketch Tools Help

STANDARD

sketch_oct09a §

```
int var1=0;
void setup(){
        size(400,200);
        background(255);
}
void draw(){

          ellipse(var1,100,30,30);
          var1=var1+1;
          if (var1 < 100){
                    fill(250,5,69);
          }
          else if (var1<200)
          {
                    fill(60,250,30);
          }
          else if (var1<300)
          {
                    fill(23,42,250);
          }
          else
          {
                    fill(125,125,125);
          }
}
```



PROCESSING

Estructura condicional (Ejemplo):

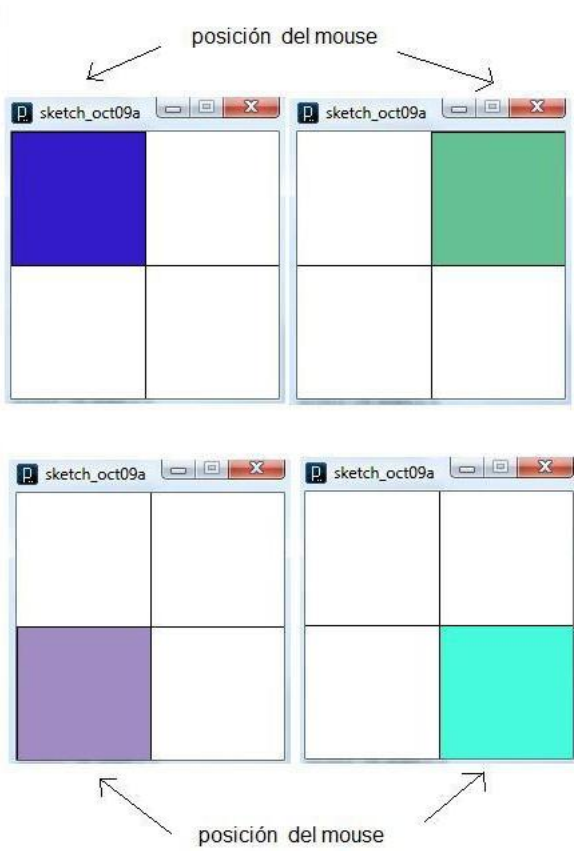
sketch_oct09a | Processing 1.5.1

File Edit Sketch Tools Help

STANDARD

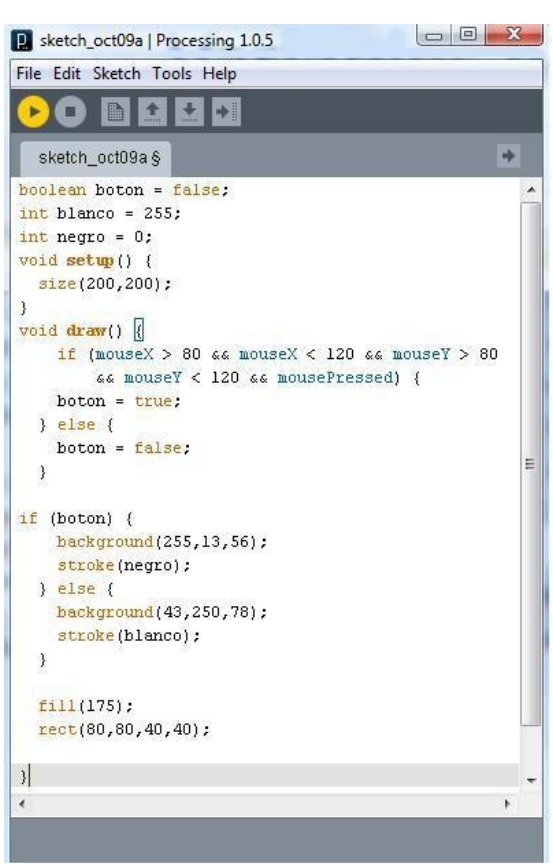
sketch_oct09a §

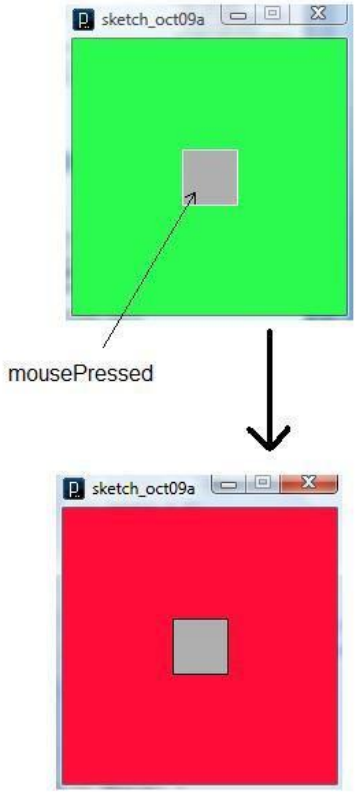
```
void setup() {
  size(200,200);
}

void draw() {
   background(255);
  stroke(0);
  line(width/2,0,width/2,height);
  line(0,height/2,width,height/2);
  rectMode(CORNER);
  fill(0);
  if (mouseX < width/2 && mouseY < height/2 ) {
          fill(random(255), random(255), random(255));
          rect(0,0, width/2, height/2);
  } else if (mouseX > width/2 && mouseY < height/2) {
          fill(random(255), random(255), random(255));
          rect(width/2,0, width/2, height/2);
  } else if (mouseX < width/2 && mouseY > height/2) {
          fill(random(255), random(255), random(255));
          rect(0, height/2, width/2, height/2);
  } else{
         fill(random(255), random(255), random(255));
         rect(width/2, height/2, width/2, height/2);
  }
 }
```

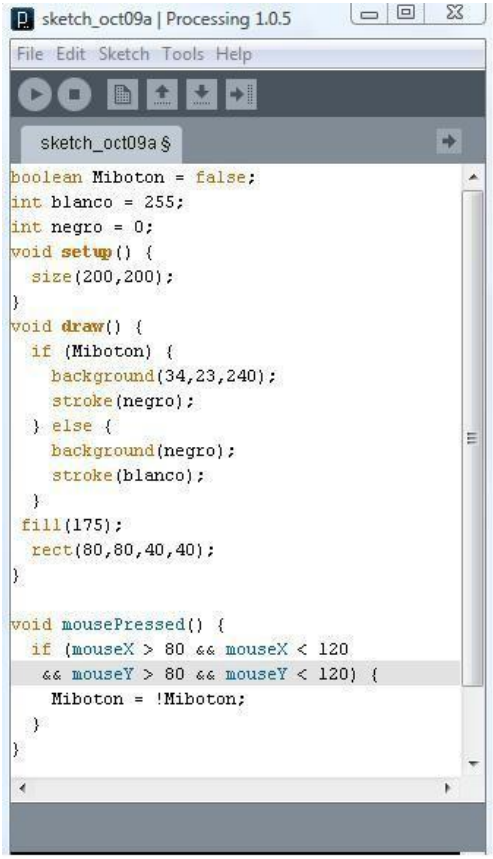


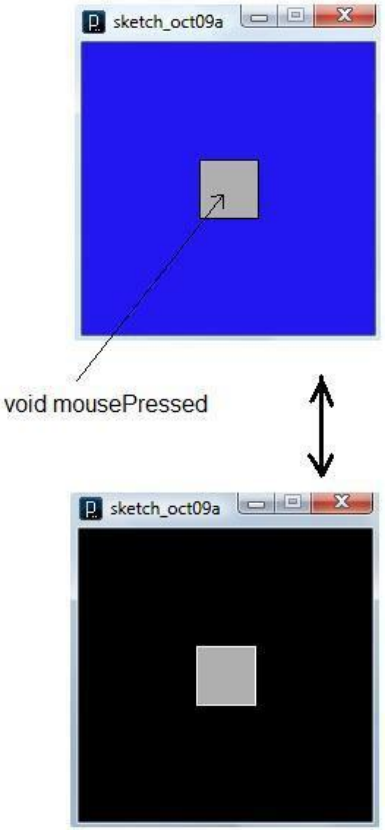
PROCESSING

Estructura condicional (Ejemplo):

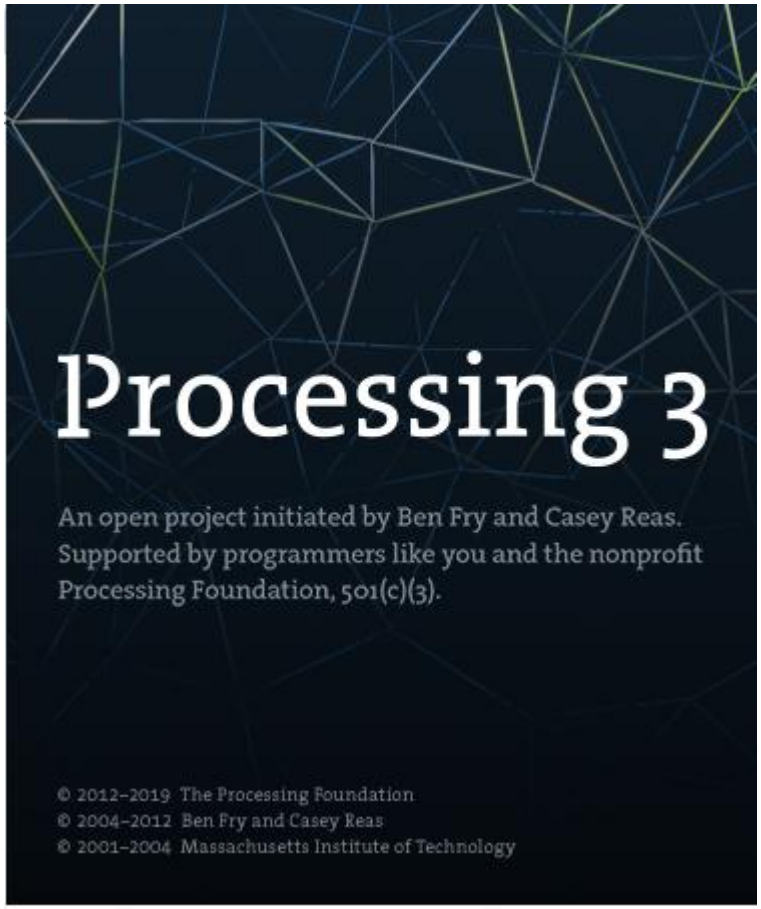








PROCESSING



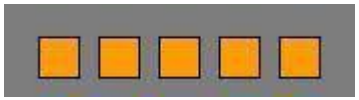
Estructura repetitiva

PROCESSING

Estructura iterativa (while):

```
while (condición) {
    ...
}
```

```
int posicion=20;
while(posicion<=140){
        Rect (posicion,20,20,20);
        posicion = posicion + 30;
}
```



PROCESSING

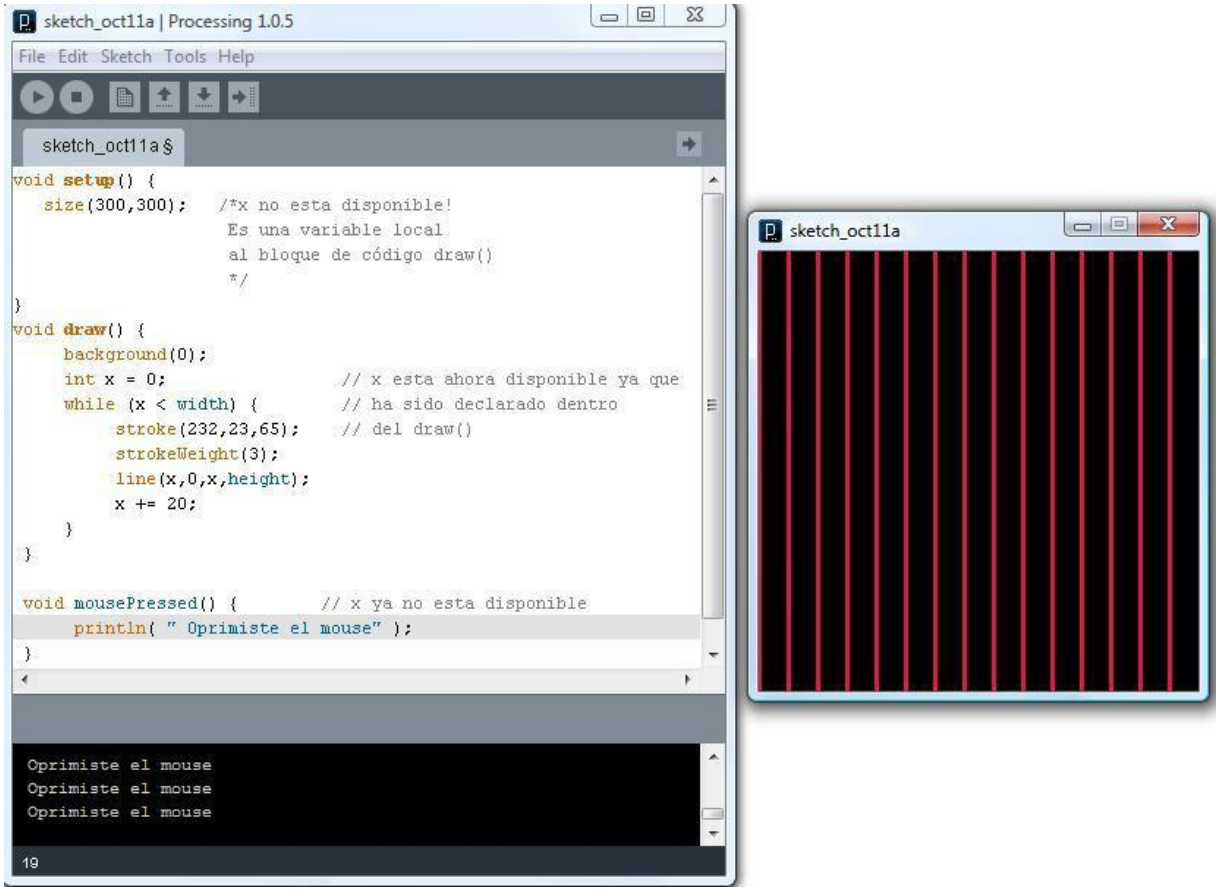
Estructura iterativa (while):

```
while (expresión booleana){
    // Conjunto de instrucciones que se ejecutan
    // si la condición es verdadera
}
```

```
int posicion=20;
while(posicion<=140){
    rect(posicion,20,20,20);
    posicion = posicion + 30;
}
```

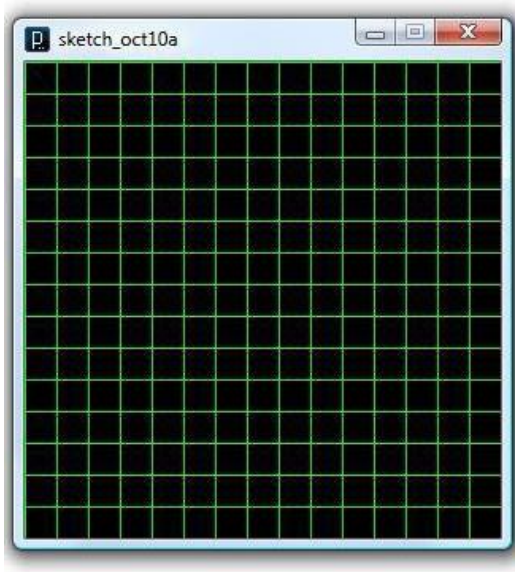
PROCESSING

Estructura iterativa (Ejemplo):



PROCESSING

Estructura iterativa (Ejemplo):



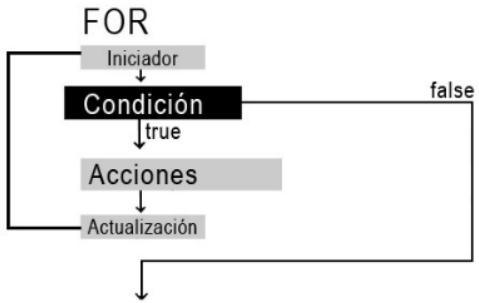
PROCESSING

Estructura iterativa (do while):

```
do {
    ...
} while (condición) ;
```

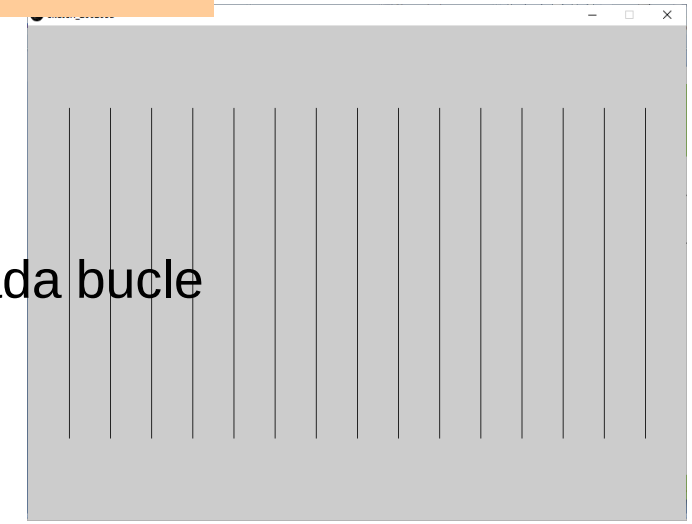
PROCESSING

Estructura iterativa (for):



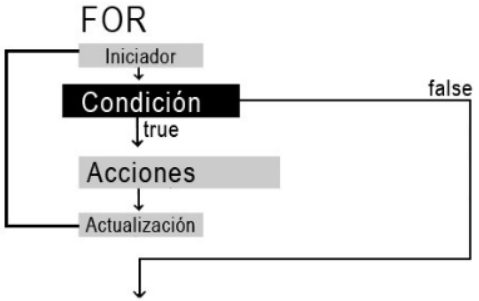

```
for (inicialización; condición; actualización) {
        ...
}
```

```
size(800,600);
for (int i = 50; // índice de control
    i < width;   // condición booleana
    i+=50)      // modificación del índice tras cada bucle
{
      line(i,100,i,500);
}
```



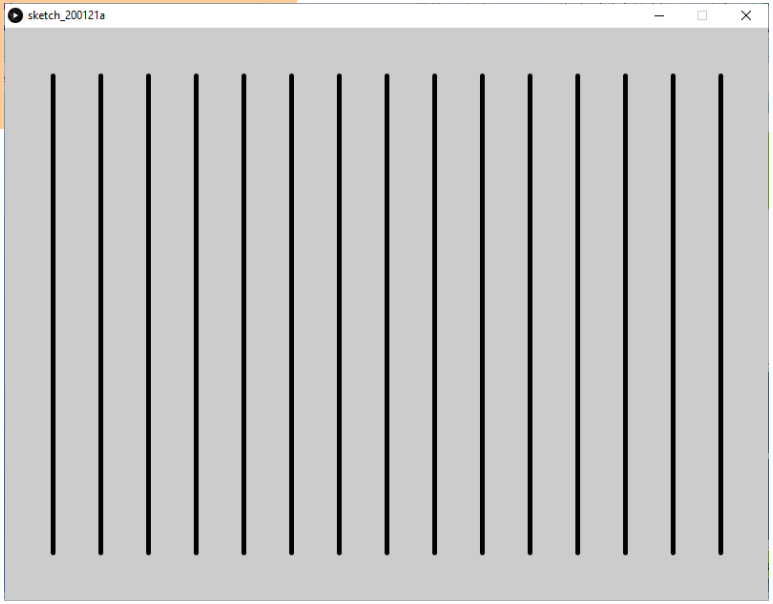
PROCESSING

Estructura iterativa (for):




```
for (inicialización; condición; actualización) {
    // Conjunto de instrucciones que
    // se ejecutan n veces
}
```

```
size(800,600);
for (int x=50; x<width; x=x+50){
  line(x,50,x,550);
}
```

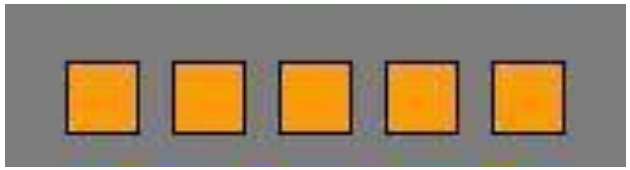


PROCESSING

Estructura iterativa (for):

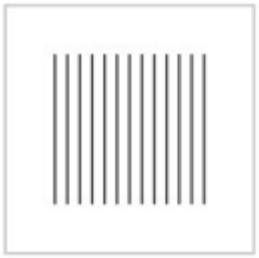
```
for (inicialización, condición, actualización)
{
    // Este es el conjunto de instrucciones que se ejecutan n veces
}
```

```
for (int posicion=20; posicion<=140; posicion=posicion+30)
{
    rect(posicion,20,20,20);
}
```



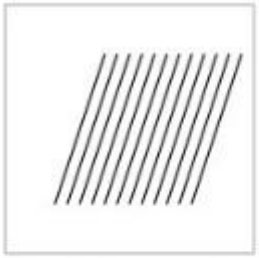
PROCESSING

Estructura iterativa (Ejemplos):

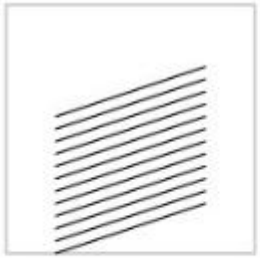


```
for (int x = 20; x <= 80; x += 5) {
  line(x, 20, x, 80);
}
```

```
for (int x = 20; x <= 80; x += 5) {
  line(20, x, 80, x);
}
```

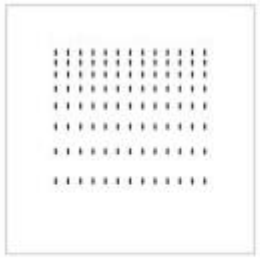


```
for (int x = 20; x < 80; x += 5) {
  line(x+20, 20, x, 80);
}
```



```
for (float x = 80; x > 20; x -= 5) {
  line(20, x+20, 80, x);
}
```

```
for (int y = 20; y <= 80; y += 10) {
  for (int x = 20; x <= y; x += 5) {
    line(x, y, x-3, y-3);
  }
```

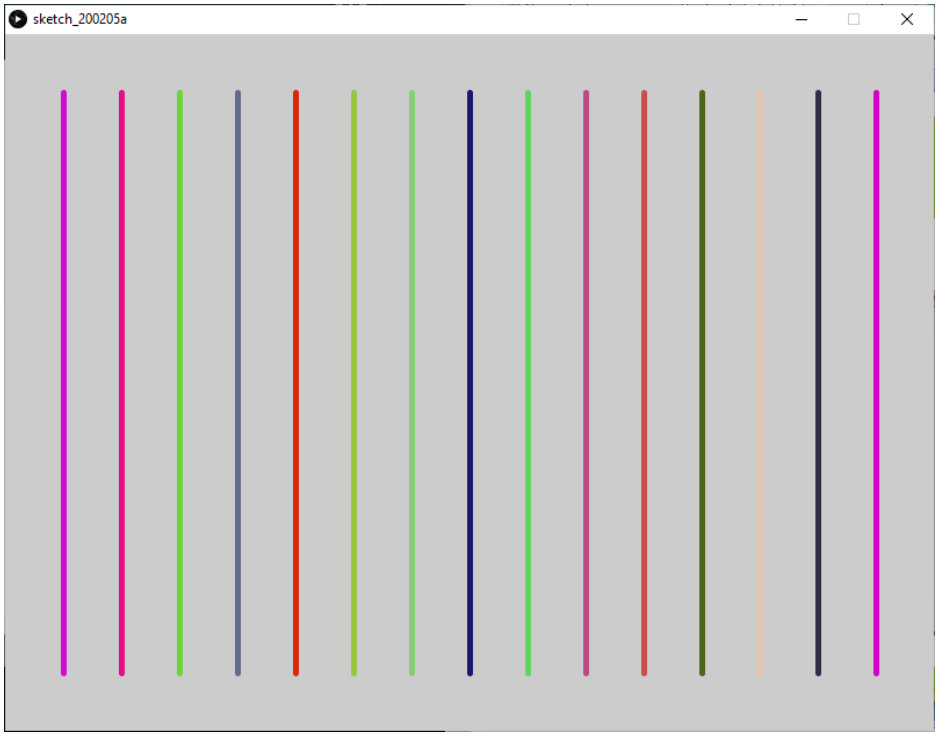


```
for (float y = 20; y <= 80; y *= 1.2) {
  for (int x = 20; x <= 80; x += 5) {
    line(x, y, x, y-2);
  }
```

PROCESSING

Estructura iterativa (Ejemplo):

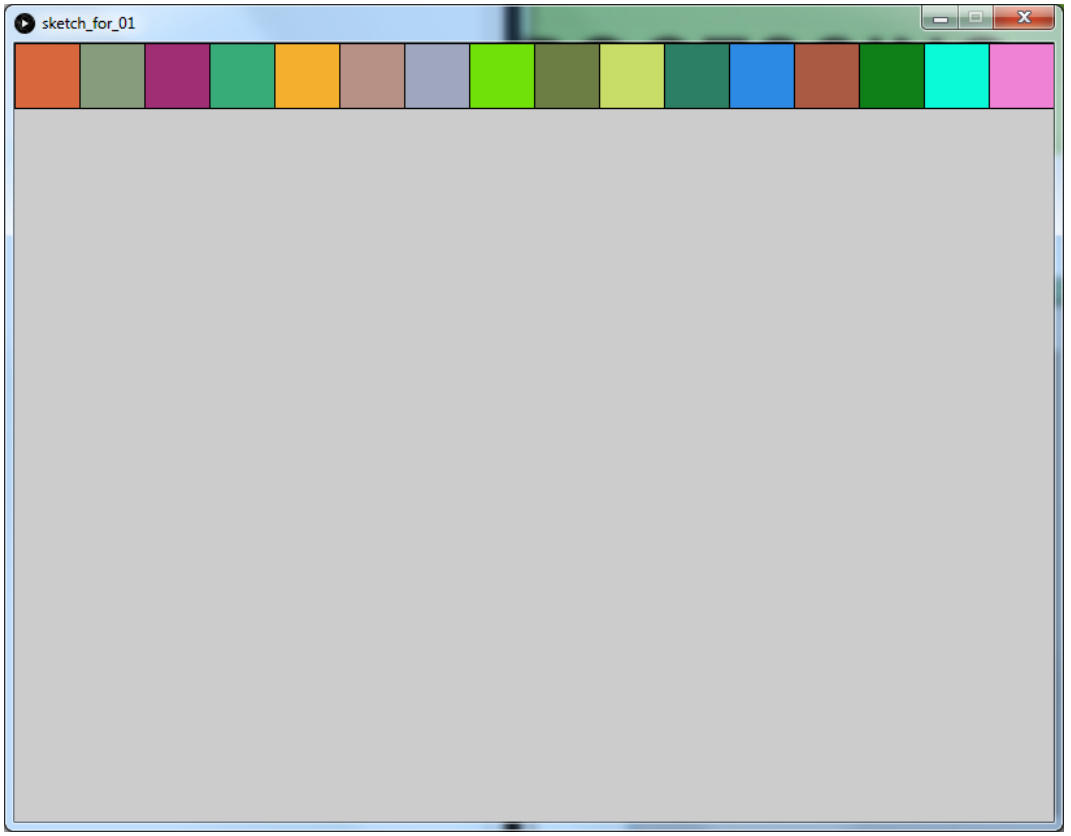
```
float r, g, b;

size(800,600);
strokeWeight(5);

for (int x=50; x<width; x=x+50) {
        r = random(255);
        g = random(255);
        b = random(255);

        stroke(r,g,b);
        line(x,50,x,550);
}
```



PROCESSING

Estructura iterativa (Ejemplo):

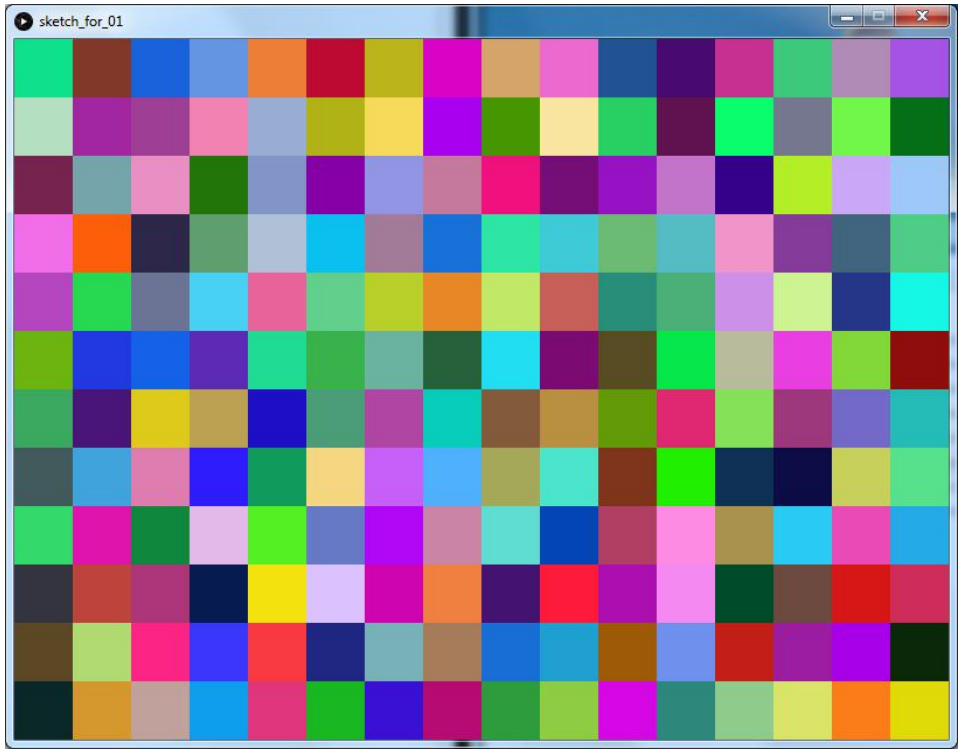
```
float r, g, b;

size(800,600);

for (int x=0; x<width; x=x+50){
  r=random(255);
  g=random(255);
  b=random(255);
  fill(r,g,b);
  rect(x,0,50,50);
}
```



PROCESSING

Estructura iterativa (Ejemplo):

```
float r, g, b;
size(800,600);
noStroke();

for (int x=0; x<width; x=x+50){
  for (int y=0; y<height; y=y+50){
    r=random(255);
    g=random(255);
    b=random(255);
    fill(r,g,b);
    rect(x,y,50,50);
  }
}
```



PROCESSING

Estructura iterativa (Ejemplo):

sketch_oct11a | Processing 1.0.5

File Edit Sketch Tools Help

Run

sketch_oct11a §

```
size(300,300);
background(0);
stroke(23,230,21);

for (int lineax =0; lineax<width; lineax=lineax+20){
  for (int lineay =0; lineay<height; lineay=lineay+20){
       line(0,lineay,width,lineay);
  }
  line(lineax,0,lineax,height);
}
```

PROCESSING

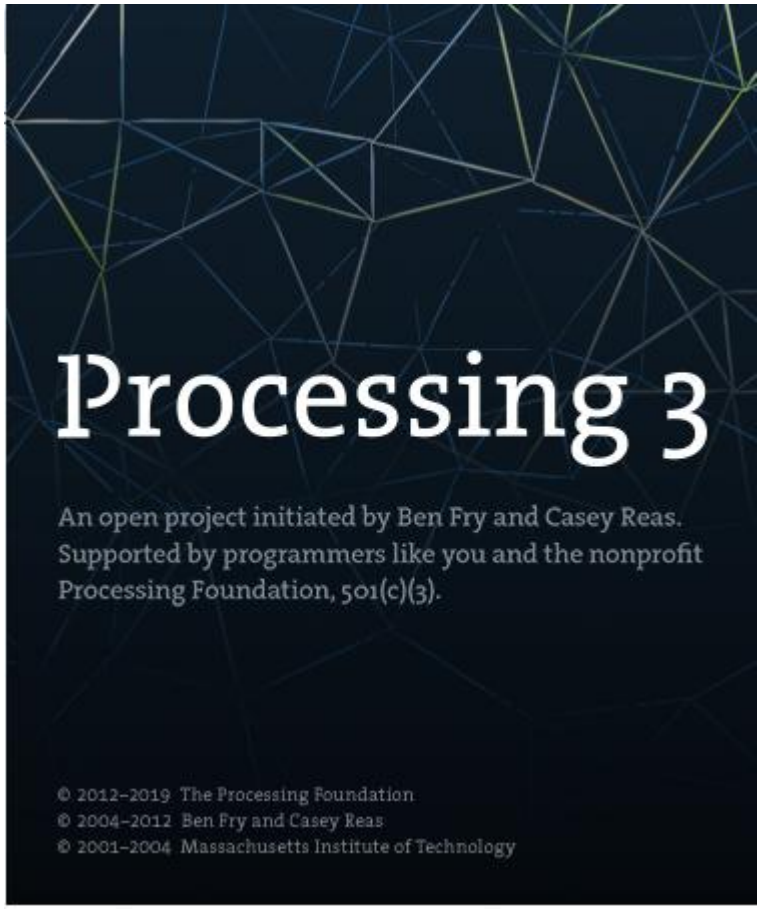
Estructuras secuencial y repetitiva (while y for):

```
rect (20,20,20,20);
rect (50,20,20,20);
rect (80,20,20,20);
rect (110,20,20,20);
rect (140,20,20,20);
```

```
int posicion=20;
while(posicion<=140){
        rect(posicion,20,20,20);
        posicion = posicion + 30;
}
```

```
for (int posicion=20; posicion<=140; posicion=posicion+30)
{
        rect(posicion,20,20,20);
}
```

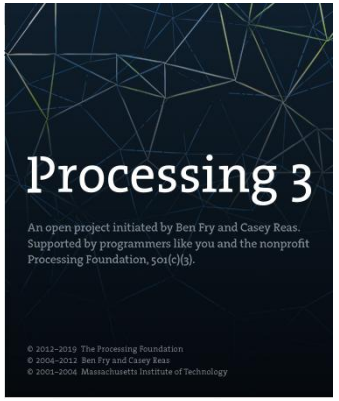
PROCESSING



Enlaces

PROCESSING

Enlaces de interés:



- https://processing.org/
- https://processing.org/reference/

- http://www.avantspace.org/processing/index.htm
- http://www.programacionyrobotica.com/processing/
- http://www.programacionyrobotica.com/practicas-processing/
- http://dunadigital.com/processing/

HARDWARE

FIN

Muchas gracias por su atención