Miniguía de Processing

Por: *Pedro Ruiz Fernández*
Versión 18/07/2017

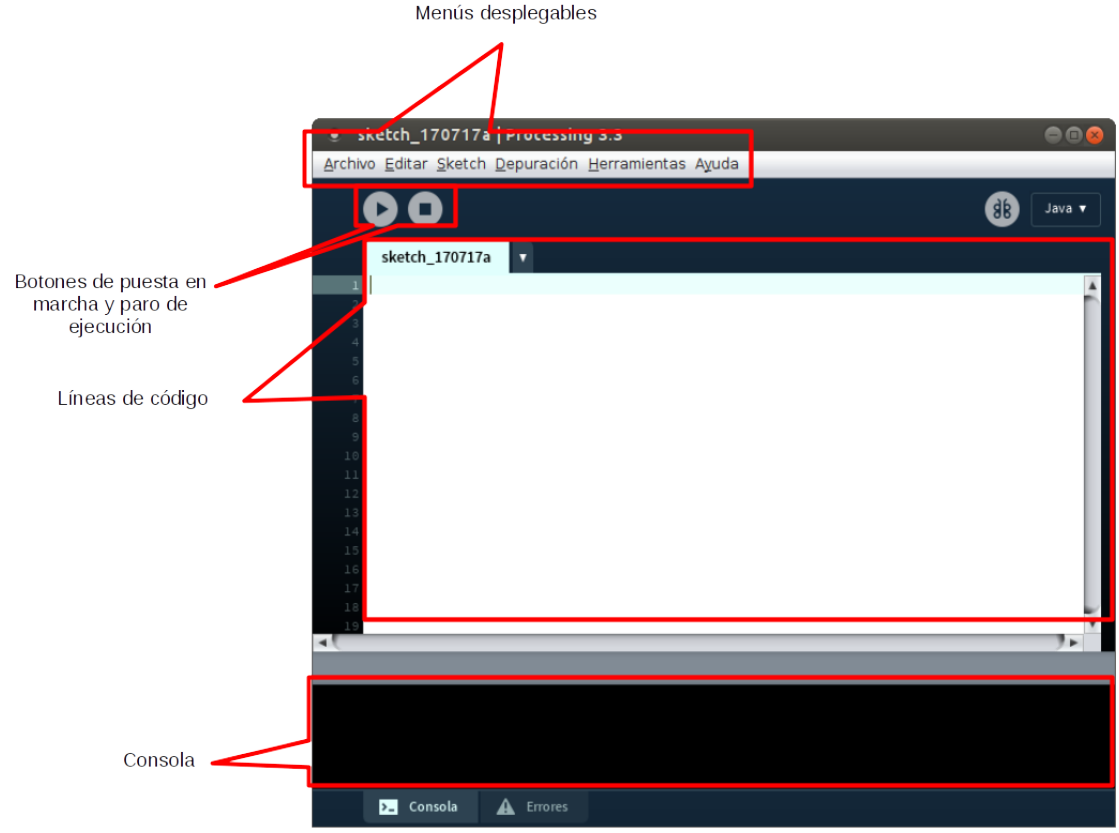
¿Qué es Processing?

Processing es un lenguaje orientado a objetos, multiplataforma que se basa en otros lenguajes como Java y c++. Está desarrollado en Java y su entorno de desarrollo (IDE) está basado en wiring como Arduino, de ahí su aspecto muy similar.

¿Dónde encuentro Processing?

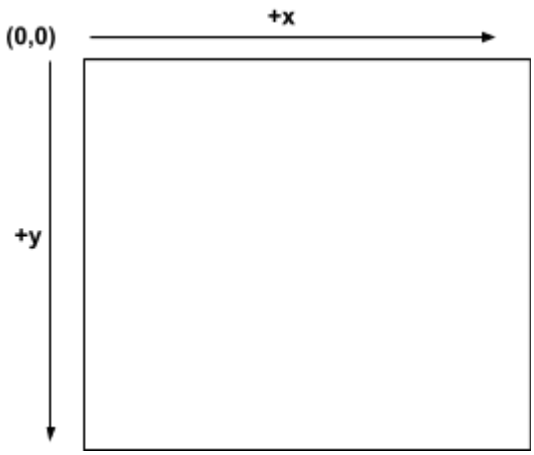
Processing lo puedo descargar desde la web oficial https://processing.org/, en ella son especialmente importantes entre otros los apartados: descargas (https://processing.org/download/), referencia (https://processing.org/reference/) el cual nos dá una ayuda magnífica a todas las órdenes del lenguaje, tutoriales (https://processing.org/tutorials/), etc.

Entorno de Processing



Sistema de coordenadas gráficas de Processing

En processing el origen de coordenadas (0,0) se encuentra en la esquina superior izquierda donde las x son crecientes hacia la derecha y las y son crecientes hacia abajo.

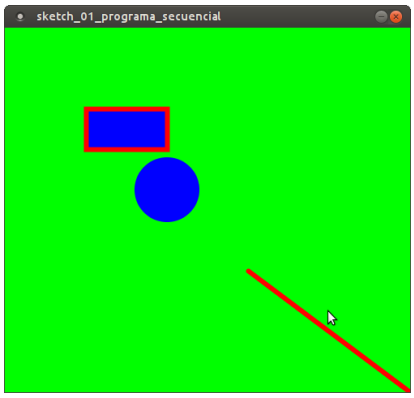


Instrucciones básicas de dibujo

- ***size (ancho, alto)***: establece el tamaño de la ventana gráfica.
- ***background (r,g,b)***: establece el color de fondo de la ventana gráfica, en este caso en RGB, cada valor entre 0 y 255.
- ***background (gris)***: establece el color de fondo de la ventana gráfica en escala de grises, el valor entre 0 (negro) y 255 (blanco).
- ***line (xPunto1,yPunto1,xPunto2,yPunto2)***: crea un línea entre dos puntos.
- ***rect (x,y,ancho,alto)***: crea un rectángulo, donde x e y reflejan la posición de la esquina superior izquierda.
- ***ellipse (x,y,diametroX,diametroY)***: crea una elipse o círculo, donde x e y establecen el centro de la misma, además tenemos que dar los valores de diámetros en los ejes X e Y, si dichos valores son los mismos tendremos un círculo.
- ***fill (r,g,b)***: establece color de relleno.

- ***fill (escala de grises)***: establece el color de relleno en escala de grises, entre 0 (negro) y 255 (blanco).
- ***stroke (r,g,b)***: establece el color del borde.
- ***strokeWeight (grosor)***: establece el grosor de borde.
- ***noFill()***: elimina el relleno.
- ***noStroke()***: elimina los bordes.

Ejemplo de programa secuencial (sketch_01_programa_secuencial), utilizando las órdenes anteriores:

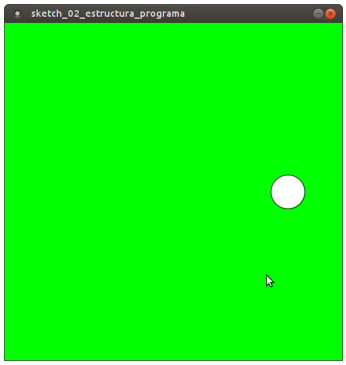



```
size (500,450);
background(0,255,0);
stroke (255,0,0);
strokeWeight (6);
fill (0,0,255);
rect (100,100,100,50);
noStroke();
ellipse (200,200,80,80);
noFill();
stroke (255,0,0);
line (300,300,500,450);
```

Estructura básica de programa en Processing

En processing tenemos dos funciones principales (void), una de ellas es *setup* y la otra *draw*, en la primera se realiza una configuración de nuestro programa, por ejemplo el tamaño de la ventana gráfica con *size (tamañoX, tamañoY)*, y se ejecuta una sóla vez. En la función *draw* todo lo que esté definido en la misma se va a repetir indefinidamente.

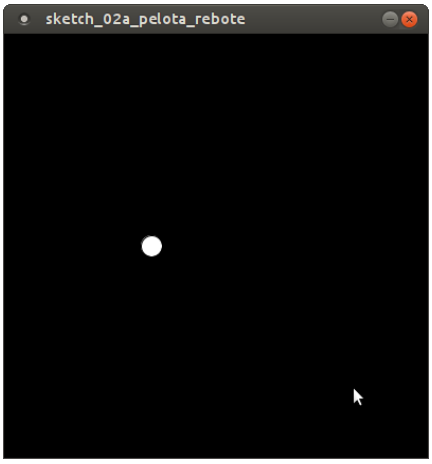
En el siguiente programa (sketch_02_estructura_programa) antes del *setup* definimos la variable x como entera (integer) (*int x*), y en el draw en cada ciclo aumentamos su valor (x=x+1), cambiando el color de fondo y desplazando hacia la derecha la circunferencia.




```
int x=0;

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

void draw() {
  background(0, x, 0);
  ellipse (x, 250, 50, 50);
  x=x+1;
}
```

Otros ejemplos:

El siguiente programa (sketch_02a_pelota_rebote) versa sobre una pelota de diámetro 20 que se dibuja en la mitad de la altura del área gráfica, y se desplaza de izquierda a derecha rebotando en las paredes laterales.




```
int x=10;
int velx=3;
int diametro=20;

void setup() {
 size (400,400);
 background(0);
 fill (255);
 }

void draw() {
 background (0);
 ellipse (x,height/2,diametro,diametro);
 if (x<(diametro/2)) {velx=-velx;}
 if (x>width-(diametro/2)) {velx=-velx;}
 x=x+velx;
}
```

Variables

En el ejemplo anterior hemos trabajado con variables, que son elementos sirven para guardar datos que usaremos luego en nuestro programa, las variables pueden ser globales (afectan a todos los procedimientos y se colocan antes de setup) o locales se definen y usan en una función y sólo sirven en ella (ya definiremos las funciones más tarde).

Tipo de variable	¿Qué guardan?	Ejemplo
integer	números enteros positivos y negativos. Entre 2,147,483,647 y -2,147,483,648	int x;
float	números decimales comprendidos entre 3.40282347E+38 y -3.40282347E+38	float x;
boolean	pueden tomar valores verdaderos (true) o falso (false)	boolean x;
char	puede guardar un conjunto de caracteres o un carácter. La definición de un conjunto de caracteres se hace con comillas dobles y la de un carácter con comillas simples	char palabra=”hola”; char caracter=’h’;

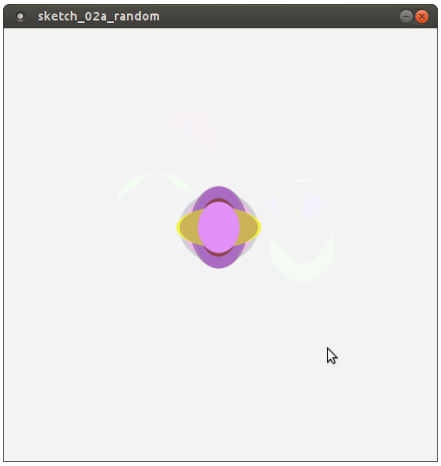
Números aleatorios (Random)

A veces es necesario la generación de número aleatorios, para ello usamos la orden *random* de dos formas diferentes:

- ***random (numero máximo)***: genera un número aleatorio entre o y el número dado, devuelve un valor tipo float (decimal).
- ***random (número mínimo, número máximo)***: genera un número aleatorio entre un número mínimo y el número máximo dado, devuelve un valor tipo float (decimal).

Para conocer mejor estas órdenes proponemos un ejercicio (sketch_02a_random) que genera elipses aleatorias de tamaños entre 0 y 100, de color de relleno aleatorio, sin bordes y sobre un fondo con transparencia. En este ejercicio introducimos las variables de sistema *mouseX*, *mouseY* que nos devuelven la coordenadas X e Y del puntero del ratón, y *width* y *height* que nos devuelven el ancho y alto de la ventana de trabajo.

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

void draw () {
  fill (255, 10);//rellena de blanco con transparencia 10%
  rect (0, 0, width, height);//realiza un rectángulo de toda la pantalla con transparencia
  fill (random(255), random(255), random(255));//cambiamos a relleno aleatorio
   ellipse (mouseX, mouseY, random (100), random(100));//dibuja elipses de tamaños aleatorios en la posición del ratón
}
```



Estructuras de control for y while

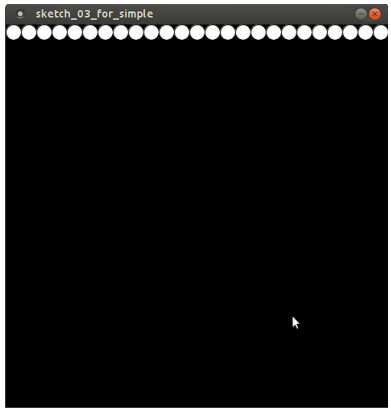
Con la estructura de control *for* vamos a realizar una serie de acciones mientras se cumpla una condición de salida del bucle.

for (valor inicial de condición;condición de salida;acción sobre condición) {instrucciones;}

Vamos a realizar un programa (sketch_03_for_simple) que repita un círculo de diámetro 20 de izquierda a derecha desde la primera posición (posición x=10) incrementando su posición en 20 cada vez, hasta que llegue al final de las posiciones x, la ventana gráfica será de 500x500.

Para ello hemos utilizado dos variables de processing que nos dan valores de la ventana gráfica, como:

- ***width***: variable predefinida que nos devuelve el ancho de la ventana gráfica.
- ***height***: variable predefinida que nos devuelve el alto de la ventana gráfica.




```
int x;
int y;
background (0);
size (500,500);
fill (255);

for (x=10; x<=width; x=x+20) {
 ellipse (x,10,20,20);
}
```

También el programa se puede hacer con un bloque de control *while*:

while (condición a cumplir) {instrucciones;}

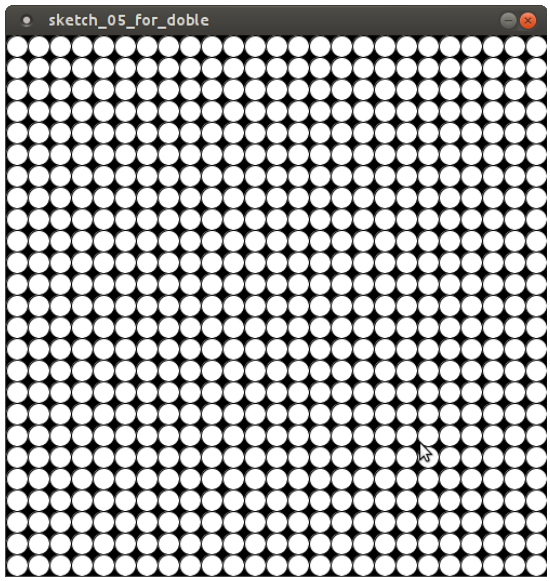
```
int x=10;
background (0);
size (500,500);
fill (255);

while (x<=width) {
 ellipse (x,10,20,20);
 x=x+20;
}
```

Ejercicio propuesto 1: realizar una serie de círculos concéntricos de color tipo (r,g,b) variable aleatorio, espaciados 3 unidades sin sobrepasar una ventana gráfica de 500 x 500. Investigar o estudiar la orden random antes de nada.

Ahora vamos a realizar no sólo la primera fila sino todas (sketch_05_for_doble), rellenando toda la pantalla gráfica de círculos, haciendo que se repita la primera desplazándola 20 unidades en y. Lo haremos tanto con *for* como con *while*.

Con *for*:

```
int x;
int y;
background (0);
size (500, 500);
fill (255);

for (y=10; y<=height; y=y+20) {

  for (x=10; x<=width; x=x+20) {
    ellipse (x, y, 20, 20);
  }
}
```



Con *while*:

```
int x=10;
int y=10;
background (0);
size (500, 500);
fill (255);

while (y<=height) {
  x=10;
  while (x<=width) {
    ellipse (x, y, 20, 20);
    x=x+20;
    println (y);
  }
  y=y+20;
}
```

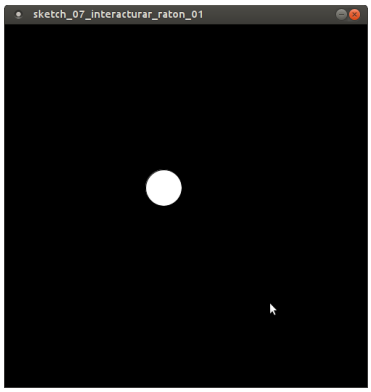
Interactuar con el ratón.

Variables de posición

En processing tenemos dos variables predefinidas que nos dan los valores de la posición en X e Y ne nuestro cursor del ratón, que son:

- ***mouseX***: variable predefinida que nos devuelve la posición en X del cursor.
- ***mouseY***: variable predefinida que nos devuelve la posición en Y del cursor.

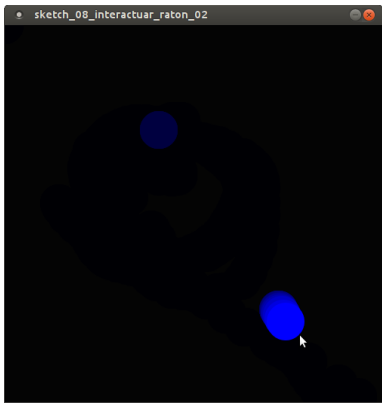
En el siguiente programa (sketch_07_interactuar_raton_01) movemos una circunferencia con el puntero del ratón. Observar que "background (0)" va dentro del *draw* para que se borre el fondo cada vez y usamos la orden "***noCursor()***" para que no se muestre el puntero del ratón.




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

void draw() {
  background (0);
  noCursor();
  ellipse (mouseX, mouseY, 50, 50);
}
```

Ejercicio propuesto 2: realizar una circunferencia con centro en el centro de la pantalla gráfica que cambie de diámetro moviendo el ratón.

El siguiente programa (sketch_08_interactuar_raton_02) es una variante del anterior donde ahora como fondo se pone un rectángulo negro con transparencia "*fill (0,0,0,40)*", y de tamaño de la ventana gráfica "*rec(0,0,width,height)*". El resultado es que en el desplazamiento de la circunferencia nos va a dejar como un halo o rastro.




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

void draw() {
  fill (0, 0, 0, 30);
  rect (0, 0, width, height);
  noStroke();
  noCursor();
  fill (0, 0, 255);
  ellipse (mouseX, mouseY, 50, 50);
}
```

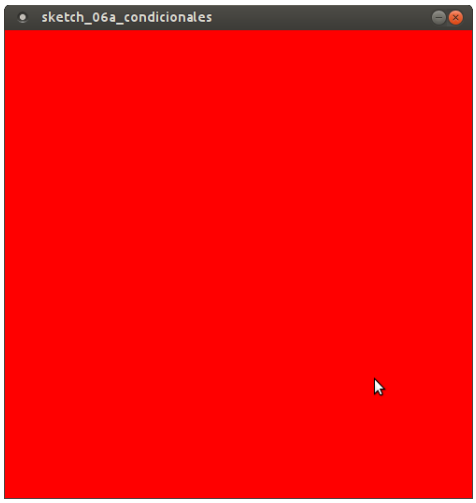
Condicionales. If else.

A la hora de tomar decisiones en un programa las estructuras condicionales son fundamentales. En esta sección te vamos a enseñar el uso de condicionales mediante la estructura *if*.

if (condición) {instrucciones;}
else {instrucciones;}

Si se cumple una condición se van a realizar unas instrucciones y en caso contrario otras. Esta estructura puede ir sin *else* (en caso contrario).

Como ejemplo proponemos un programa (sketch_06a_condicionales) que si el cursor del ratón está situado de la mitad de pantalla hacia la izquierda ponga un fondo rojo, y si lo está de la mitad de pantalla hacia la derecha ponga un fondo azul.




```
void setup () {
  size (500, 500);
  fill (0, 255, 0);
}

void draw() {
  if (mouseX <= width/2) {
    background (255, 0, 0);
  } else {
    background (0, 0, 255);
  }
}
```

else if

Finalmente, para probar condiciones múltiples, podemos emplear un "*else if*". Cuando un *else if* se utiliza, las sentencias condicionales se evalúan en el orden presentado. Tan

pronto como se compruebe que una expresión booleana es verdadera, se ejecuta el código correspondiente y se ignoran las demás expresiones booleanas.

Además para el siguiente ejemplo debemos conocer las órdenes de visualización de información a través de la consola o terminal:

- ***print (dato)***: visualiza el dato sin hacer salto de línea.
- ***println (dato)***: visualiza el dato realizando salto de línea.

Si el dato a representar es un texto fijo va entre comillas, si es una variable va sin comillas.

Dentro de los condicionales tenemos los elementos comparadores que son:

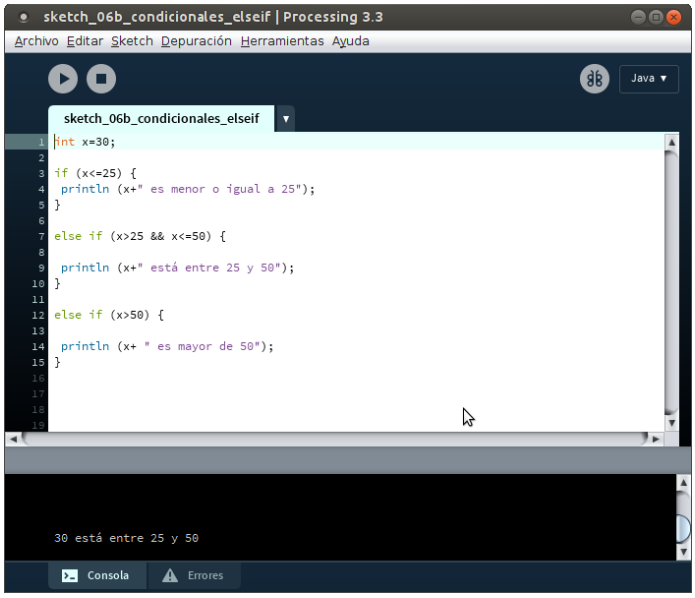
Mayor	Mayor o igual	Menor	Menor o igual	igual
>	>=	<	<=	==

Cuidado no confundir el igual de asignación de valor a una variable, por ejemplo "*int x=5*", con la comprobación si una variable es igual a un dato, por ejemplo "*if (x==5) {instrucciones a realizar}*".

Además en este ejemplo usamos en los condicionales combinación de comparadores dentro del *if*, para ello podemos usar:

- Unión de condiciones (una **y** la otra se deben cumplir) (***&&***): ejemplo "*if (x>=10 && x<15) {instrucciones a realizar}*"
- Disyunción de condiciones (una **o** la otra se deben cunplir) (||): ejemplo "*if (x>=10 || x<15) {instrucciones a realizar*}"

El siguiente programa (sketch_06b_condicionales_elseif) determina si un numero esta entre 0 y 25, 26 y 50, o más grande de 50, mostrándolo en la consola.



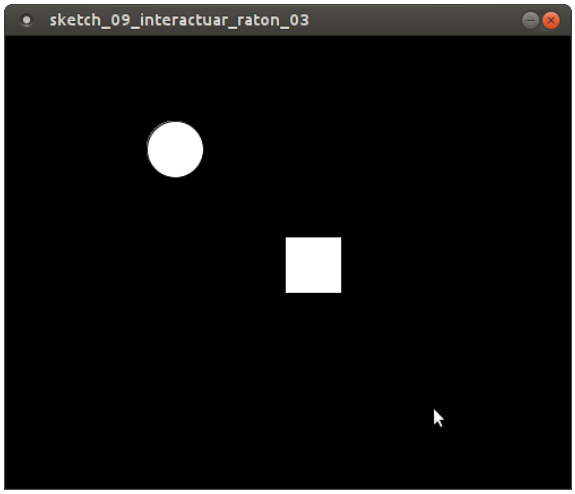

```
int x=30;

if (x<=25) {
 println (x+" es menor o igual a 25");
}

else if (x>25 && x<=50) {

 println (x+" está entre 25 y 50");
}

else if (x>50) {

 println (x+ " es mayor de 50");
}
```

Variables de estado

Además tenemos variables del sistema que controlan el estado del ratón como son:

- ***mousePressed***: nos indica si el ratón está presionado o no, puede tomar los valores "true" o "false".
- ***mouseButton***: nos indica el botón pulsado del ratón, puede tomar los valores "LEFT"."RIGHT","CENTER".

En el siguiente programa (sketch_09_intercatuar_raton_03) en función del botón pulsado del ratón dibuja un rectángulo (botón derecho), una circunferencia (botón izquierdo) o borra la pantalla (botón central).




```
int x=0;
void setup() {

  size (500, 400);
  background(0);
}

void draw () {
  if (mousePressed==true) {
    if (mouseButton==LEFT) {
      ellipse (mouseX, mouseY, 50, 50);
```

```
    }

    if (mouseButton==RIGHT) {
      rect (mouseX, mouseY, 50, 50);
    }

    if (mouseButton==CENTER) {
      background(0);
    }
  }
  println (mouseButton);
}
```

Ejercicio propuesto 3: realiza un programa que dibuje líneas blancas a base de círculos de diámetro 20 dejando pulsado el botón izquierdo y moviendo el ratón, y pulsando el botón derecho y moviendo el ratón borre. El fondo será negro y los dibujos en blanco.

Funciones o eventos

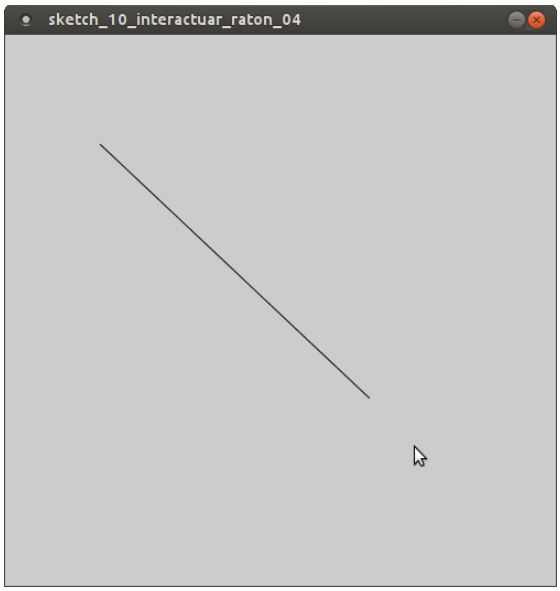
¿Qué es una función?

Una función es un conjunto de instrucciones que tienen entidad propia y que se ejecutan cuando llamamos a la misma. Ya conocemos dos: *setup* y *draw*.

También se puede interactuar con el ratón con eventos que son funciones que responden a acciones que se realizan en este caso con el ratón, estas funciones son:

- ***mousePressed()***: entra en juego cuando se pulsa el ratón.
- ***mouseReleased()***: se pone en funcionamiento cuando se suelta el botón del ratón.
- ***mouseMove()***: se pone en marcha cuando se mueve el ratón.
- ***mouseDragged()***: se activa cuando el ratón se pincha y arrastra.

El siguiente programa dibuja líneas, el primer punto lo marcas haciendo clic en el ratón y el segundo punto es dónde sueltas el ratón.




```
int x1, y1, x2, y2;
void setup () {
  size (500, 500);
}
void draw() {
}

void mousePressed() {
  x1=mouseX;
  y1=mouseY;
  println (x1, y1);
}

void mouseReleased() {
  x2=mouseX;
  y2=mouseY;
  println (x2, y2);
  line (x1, y1, x2, y2);
}
```

Interactuar con el teclado.

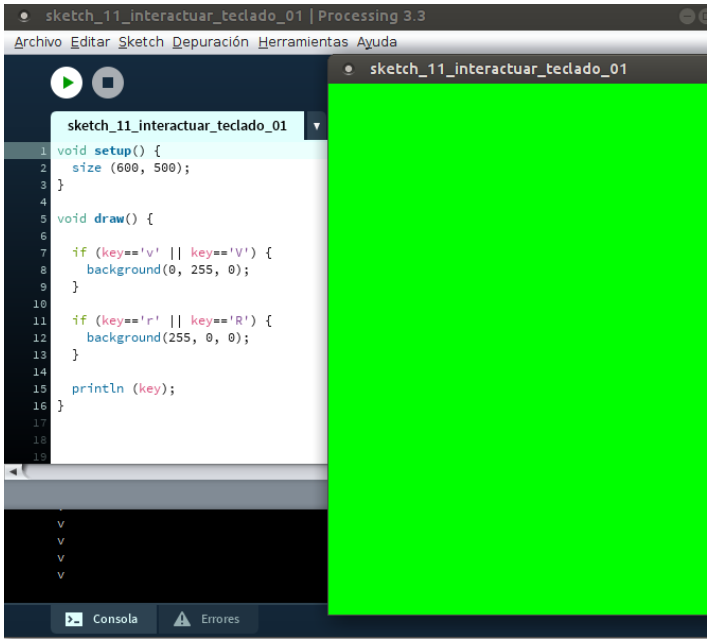
Variables de estado

Para interactuar con el teclado podemos usar variables del sistema, como:

- ***keyPressed***: nos devuelve el estado actual de pulsación del teclado, puede ser true o false.
- ***key***: devuelve la última tecla pulsada.

En el siguiente ejemplo (sketch_11_interactuar_teclado_01) si la última tecla pulsada es “v” o “V” el fondo lo hace verde y si es “r” o “R” lo cambia a rojo. Además nos colocamos un “chivato” en forma de texto que se imprime en la consola con la orden “*println (key)*”, que visualiza constantemente en la consola de texto el valor de “*key*”, que recordamos que es la última tecla pulsada.

```
void setup() {
  size (600, 500);
}

void draw() {

  if (key=='v' || key=='V') {
    background(0, 255, 0);
  }

  if (key=='r' || key=='R') {
    background(255, 0, 0);
  }

  println (key);
}
```



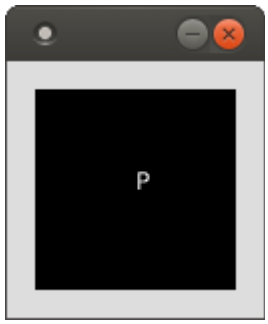
Funciones o eventos

Para el teclado además existen funciones que responden a eventos son:

- ***keyPressed ()***: se ejecuta una vez si se pulsa el teclado.
- ***keyReleased ()***: se ejecuta una vez si se suelta cualquier tecla pulsada anteriormente.

En el siguiente ejemplo (sketch_12_interactuar_teclado_02) vamos a dibujar en el centro de la pantalla gráfica el carácter pulsado en el teclado.

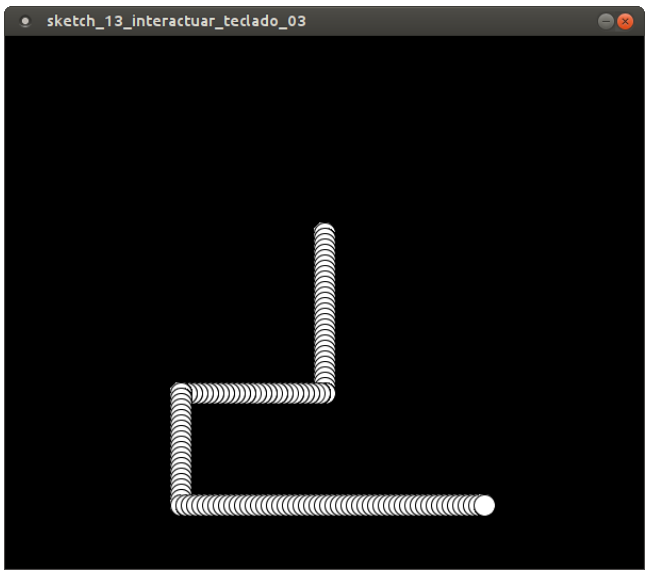
```
void setup() {
  size (100,100);
  background (0);
}

void draw() {
}

void keyPressed() {
  background(0);
  text (key, width/2, height/2);
}
```



Actividad de investigación 1: investiga sobre la orden "text" en el manual de referencia de processing.

El siguiente ejercicio dibuja líneas blancas sobre fondo negro a base de círculos de diámetro 20, estas líneas se mueven con las teclas "w" (arriba), "s" (abajo), "a" izquierda y "d" (derecha). Los centros de los círculos no sobrepasan los límites del área de dibujo.

```
int coorX, coorY;

void setup() {
  background(0);
  size (600, 500);
  coorX=width/2;
  coorY=height/2;
}

void draw() {
  ellipse(coorX, coorY, 20, 20);
}

void keyPressed() {

  if (key=='d') {
    coorX=coorX+5;
  }

  if (key=='a') {
```



```
    coorX=coorX-5;
  }

  if (key=='w') {
    coorY=coorY-5;
  }

  if (key=='s') {
    coorY=coorY+5;
  }

  if (coorX>=width-10) {
    coorX=width-10;
  }
  if (coorX<=10) {
    coorX=10;
  }
  if (coorY>=height-10) {
    coorY=height-10;
  }
  if (coorY<=10) {
    coorY=10;
  }

  println(key);
}
```

Estructura de control *Switch case*

Vamos a resolver el ejercicio anterior pero aprendiendo una nueva estructura de control de flujo de nuestro programa (sketch_14_interactuar_teclado_04_shitch_case), *switch*, que permite evaluar los valores de una expresión y en función de los mismos realizar acciones diferentes, esto también se puede solventar con varios *if*, o *if else*:

```
switch (expresion)
{
  case valor1:
  instrucciones;
  case valor2:
  instrucciones;
  default:
  instrucciones;
}

int coorX, coorY;

void setup() {
  background(0);
  size (600, 500);
  coorX=width/2;
  coorY=height/2;
}

void draw() {
  ellipse(coorX, coorY, 20, 20);
}

void keyPressed() {

  switch (key) {
  case 'w':
    coorY=coorY-5;
  case 's':
    coorY=coorY+5;
  case 'd':
    coorX=coorX+5;
  case 'a':
    coorX=coorX-5;
```

```
  }

  if (coorX>=width-10) {
    coorX=width-10;
  }
  if (coorX<=10) {
    coorX=10;
  }
  if (coorY>=height-10) {
    coorY=height-10;
  }
  if (coorY<=10) {
    coorY=10;
  }

  println(key);
}
```

Vamos a resolver el ejercicio anterior pero se va a estar moviendo la línea blanca en una dirección hasta que no cambies de tecla (sketch_15_interactuar_teclado_05). La clave está en que la variable *key* siempre toma el valor de la última tecla pulsada, y no cambia hasta que se pulse otra.

```
int coorX, coorY;

void setup() {
  background(0);
  size (600, 500);
  coorX=width/2;
  coorY=height/2;
}

void draw() {
  ellipse(coorX, coorY, 20, 20);

  if (key=='d') {
    coorX=coorX+5;
  }

  if (key=='a') {
    coorX=coorX-5;
  }

  if (key=='w') {
    coorY=coorY-5;
  }

  if (key=='s') {
    coorY=coorY+5;
  }

  if (coorX>=width-10) {
    coorX=width-10;
  }
  if (coorX<=10) {
    coorX=10;
  }
  if (coorY>=height-10) {
    coorY=height-10;
  }
  if (coorY<=10) {
    coorY=10;
  }

  println(key);
}
```

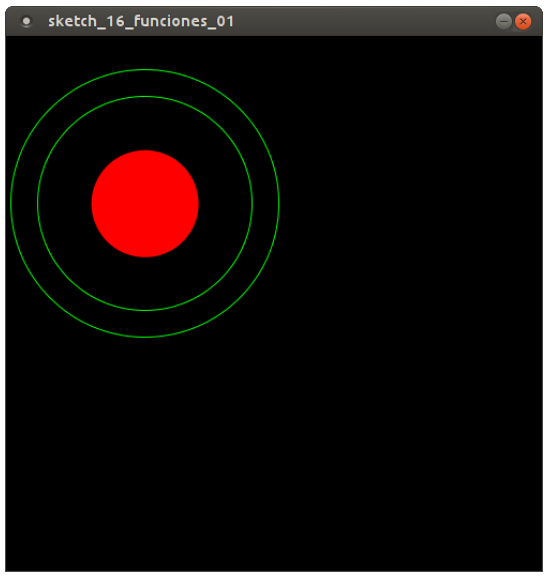
Funciones propias

En processing también podemos definir funciones nuestras además de las funciones predefinidas como *setup*, *draw*, *mousePressed*, etc.

En el ejemplo siguiente (sketch_16_funciones_01) creamos una función *planeta* que llamamos dentro de la función *draw*.

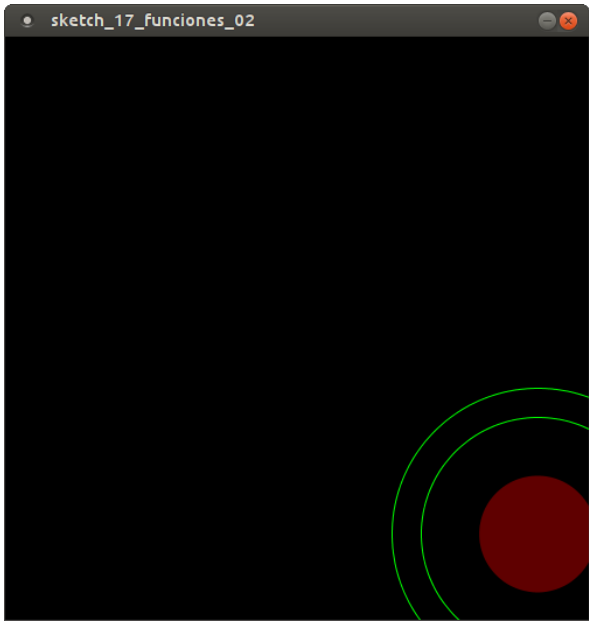
El programa llama a la función *planeta* dentro de la función *draw*. Además hay dos cuestiones nuevas, las variables *centroX* y *centroY* que definen el centro de los círculos son **variables locales**, esto significa que sólo están definidas en la función planeta, y no se pueden usar en otras funciones. Si estuvieran definidas antes del *setup* serían **variables globales** y se podrían usar en cualquier parte del programa. Además está la orden ***delay (tiempo)*** que supone un retardo de un tiempo en milisegundos.

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

void draw() {
  background(0);
  planeta();
  delay (1000);
}

void planeta() {
float centroX=random (500);
float centroY=random (500);
 noStroke();
 fill (255,0,0);
 ellipse (centroX,centroY,100,100);
 noFill();
 stroke (0,255,0);
 ellipse (centroX,centroY,200,200);
 ellipse (centroX,centroY,250,250);
}
```



El siguiente programa (sketch_17_funciones_02) es muy parecido al anterior pero ahora llamamos a la función *planeta* en *draw* pasándole parámetros o valores a variables de la función planeta como son: *centroX*, *centroY* y *colorR*.

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

void draw() {
  background(0);
  planeta(random (width),random(height), random (255));
  delay (1000);
}

void planeta(float centroX,float centroY, float colorR) {
 noStroke();
 fill (colorR,0,0);
 ellipse (centroX,centroY,100,100);
 noFill();
 stroke (0,255,0);
 ellipse (centroX,centroY,200,200);
 ellipse (centroX,centroY,250,250);
}
```

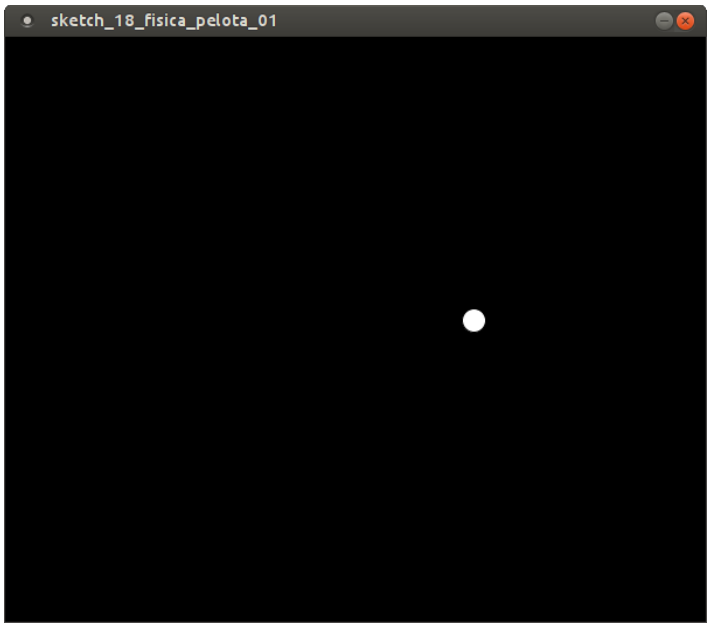


Algo de física con processing

Pelota con aceleración gravitatoria

Establecemos variables de velocidad en x e y, y aceleración en y. Si la pelota llega al final o principio de y la velocidad cambia de sentido, pero la aceleración siempre es positiva luego para abajo acelera y para arriba decelera.

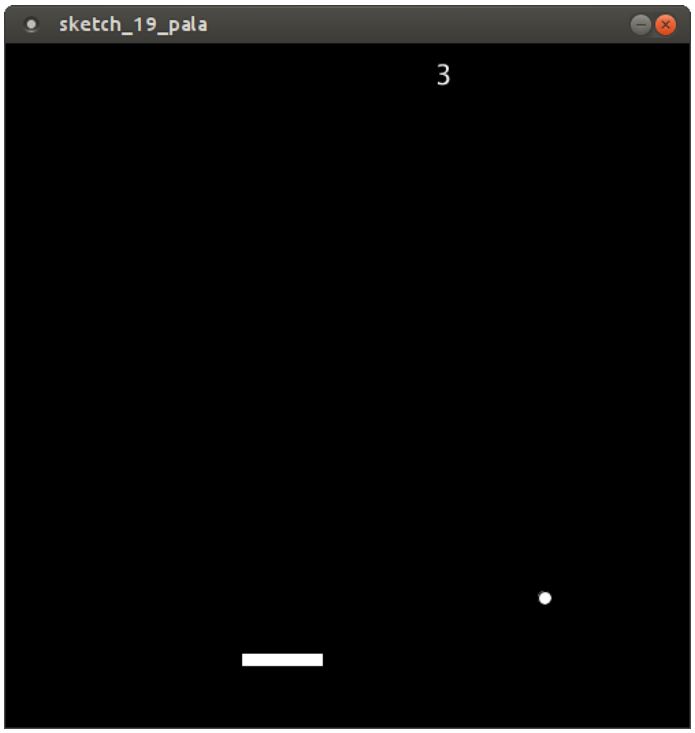
```
float velX=0;
float velY=1;
float acelY=0.1;
float posX;
float posY=1;

void setup() {
size (600,500);
background(0);
posX=random (width);
}

void draw() {
  background(0);
  ellipse (posX,posY,20,20);
  velX=random(-5,5);
  posX=posX+velX;
  velY=(velY+acelY);
  posY=posY+velY;
  if (posY>height) { posY=height;}
  if (posY<0) {posY=0;}
  if (posY>=height || posY<=0) {
  velY=-velY;
  //acelY=-acelY;
  }
  if (posX>width) {
  posX=width;
  }
  if (posX<0) {
    posX=0;}
println (velY);
}
```



Arkanoid con Processing

Se trata del clásico juego de control de pala horizontal que debemos hacer rebotar en ella una pelota que desciende.

```
boolean gameOver = true;
int score1;
float posY;
float posX;
int speedX;
float speedY;

void setup () {
  size(500, 500);
  posX = width/2; //posición X de pelota
  posY = height/2; //posición y de pelota
  speedX = 2; //velocidad x de pelota
  speedY= -5; //velocidad y de pelota
  score1= 0;//marcador
  rectMode(CENTER);
}

void draw () {
  background(0);

   if (gameOver == true) {///si la variable gameOver es verdadera
     textAlign(CENTER);
     textSize(14);
     text ("Press a Key", width/2, (height/2)+20);
     textSize(20);
     text ("GAME OVER", width/2, height/2);
   }
   else {/// en caso que gameOver sea falsa vamos a jugar

     fill(255);
```



```
    rect( mouseX, height*0.9, 60, 10);//dibujamos la pala
    fill(255);
    ellipse(posX, posY, 10, 10);//dibujamos la pelota

    if (posY< 0) {// si la pelota está arriba del todo, cambiamos sentido
      speedY= - speedY;
      posY= 0;
    }

    if (posY >height) {// si la pelota está abajo del todo termina la partida
      posX = width/2;
      posY = height/2;
      gameOver = true;
      speedX = speedX ;
      speedY = speedY ;
    }
    if (posX> width) {//si la pelota llega a la derecha cambia de sentido
      speedX= - speedX;
      posX= width;
    }

    if (posX< 0) {//si la pelota llega a la izquierda cambia de sentido
      speedX= - speedX;
      posX= 0;
    }

    if (posX > mouseX-30 && posX < mouseX + 30 && posY > 0.9*height-5 && posY < 0.9*height+5) {
      //este if controla que esté tocando la pala
      speedX = speedX;
      speedY = -1*speedY;
      score1= score1+1;
    }
    else {// si no toca la pala

      text(score1, 320, 30);
    }

    println(posY);
    posX= posX + speedX;
    posY= posY + speedY;
  }
}

void keyPressed() {
  if (gameOver == true) {
    gameOver = false;
    score1=0;
  }
}
```

Glosario de instrucciones

Dibujo

- ***size (ancho, alto)***: establece el tamaño de la ventana gráfica.
- ***background (r,g,b)***: establece el color de fondo de la ventana gráfica, en este caso en RGB, cada valor entre 0 y 255.
- ***background (gris)***: establece el color de fondo de la ventana gráfica en escala de grises, el valor entre 0 (negro) y 255 (blanco).
- ***line (xPunto1,yPunto1,xPunto2,yPunto2)***: crea un línea entre dos puntos.
- ***rect (x,y,ancho,alto)***: crea un rectángulo, donde x e y reflejan la posición de la esquina superior izquierda.
- ***ellipse (x,y,diametroX,diametroY)***: crea una elipse o círculo, donde x e y establecen el centro de la misma, además tenemos que dar los valores de diámetros en los ejes X e Y, si dichos valores son los mismos tendremos un círculo.
- ***fill (r,g,b)***: establece color de relleno.
- ***fill (escala de grises)***: establece el color de relleno en escala de grises, entre 0 (negro) y 255 (blanco).
- ***stroke (r,g,b)***: establece el color del borde.
- ***strokeWeight (grosor)***: establece el grosor de borde.
- ***noFill()***: elimina el relleno.
- ***noStroke()***: elimina los bordes.

Control

for (valor inicial de condición;condición de salida;acción sobre condición) {instrucciones;}

while (condición a cumplir) {instrucciones;}

if (condición) {instrucciones;}
else {instrucciones;}

```
switch (expresion)
{
 case valor1:
 instrucciones;
 case valor2:
 instrucciones;
 default:
 instrucciones;
}
```

Variables de sistema

- ***width***: variable predefinida que nos devuelve el ancho de la ventana gráfica.
- ***height***: variable predefinida que nos devuelve el alto de la ventana gráfica.
- ***mouseX***: variable predefinida que nos devuelve la posición en X del cursor.
- ***mouseY***: variable predefinida que nos devuelve la posición en Y del cursor.
- ***mousePressed***: nos indica si el ratón está presionado o no, puede tomar los valores “true” o “false”
- ***mouseButton***: nos indica el botón pulsado del ratón, puede tomar los valores “LEFT”.”RIGHT”,”CENTER”.
- ***keyPressed***: nos devuelve el estado actual de pulsación del teclado, puede ser true o false.
- ***key***: que devuelve la última tecla pulsada.
- ***delay (tiempo)***: establecemos una espera en milisegundos para ejecutar la siguiente instrucción.

Funciones de sistema

- ***setup()***: se ejecuta su contenido una vez y sirve para configurar parámetros de nuestro programa.
- ***draw()***: se ejecuta cíclicamente su contenido.
- ***mousePressed()***: entra en juego cuando se pulsa el ratón.
- ***mouseReleased()***: se pone en funcionamiento cuando se suelta el botón del ratón.
- ***mouseMove()***: se pone en marcha cuando se mueve el ratón.
- ***mouseDragged()***: se activa cuando el ratón se pincha y arrastra.
- ***keyPressed ()***: se ejecuta una vez si se pulsa el teclado.
- ***keyReleased ()***: se ejecuta una vez si se suelta cualquier tecla pulsada anteriormente.

Operaciones matemáticas

- ***random (numero máximo)***: genera un número aleatorio entre 0 y el número dado, devuelve un valor tipo float (decimal).

- ***random (número mínimo, número máximo)***: genera un número aleatorio entre un número mínimo y el número máximo dado, devuelve un valor tipo float (decimal).

Consola

- ***print (dato)***: visualiza el dato en la consola sin hacer salto de línea.
- ***println (dato)***: visualiza el dato en la consola realizando salto de línea.