

TEMA 8

Python en un entorno con salida gráfica (Processing)



ÍNDICE

Primer programa en Processing	2
Creando animaciones	6
Ámbito de una variable	10
Cambios en el movimiento	13
Instrucciones de código usadas	16
Reto 1	17
Reto 2	17

Primer programa en Processing

Hasta ahora hemos estado trabajando con Python en scripts que ejecutaban un código con salida textual, es decir, sin gráficos. Los dos últimos capítulos de este curso los vamos a dedicar a programar con Python y salida gráfica.

Para ello vamos a usar un programa denominado Processing, que ya instalaste en la unidad 1. No es un programa que se use en el entorno profesional para crear scripts de Python con salida gráfica, pero es muy útil para aprender y hacer nuestras primeras animaciones y juegos.

Para usar **Processing**, tenemos una gran ayuda con su página web en la que hay una sección referencia con todas las instrucciones de código soportadas y ejemplos de cada una de ellas:

<http://py.processing.org/reference/>

En cualquier momento que no recuerdes cómo usar una función o quieras avanzar más allá de lo planteado puedes acudir al enlace anterior y ver cómo se podría hacer.

Una vez que ya habéis instalado Processing y estáis en el editor de código en Python, vamos a ver cómo creamos nuestro primer programa, que va a ser un **cuadrado centrado en el área de visualización**.

Para ello vamos a ver primero cómo se ejecuta un programa de Python para Processing, ya que difiere un poquito con lo que hemos visto hasta ahora.

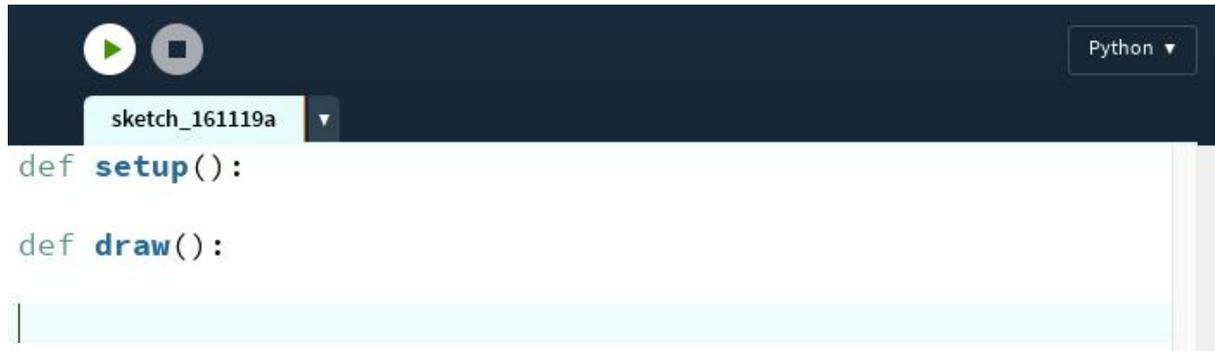
En Processing vamos a **programar en bucle** y no secuencialmente. Esto quiere decir que nuestro programa, sin tener que indicarle nada, se va a repetir una y otra vez. Hasta ahora nuestros programas se ejecutaban secuencialmente, aunque a veces teníamos bucles *while* o *for* que repetían una serie de veces una instrucción.

Ahora vamos a tener inicialmente siempre dos funciones ya creadas (recuerda que para crear funciones usábamos la instrucción *def*). La función *setup* y la función *draw*.

En la función *setup* vamos a poner todo aquello que define nuestro programa y que queremos que se haga una sola vez al principio del mismo, como por ejemplo el tamaño de la pantalla que queremos para nuestro script gráfico o el color de fondo que va a tener esa pantalla.

En el *draw* vamos a poner esas instrucciones que queremos que se repitan una y otra vez, como por ejemplo dibujar una pelota y moverla en una dirección poco a poco.

Todo esto va a quedar mucho más claro a lo largo del tema, pero vamos a ver cómo sería esa estructura ya escrita en Processing:

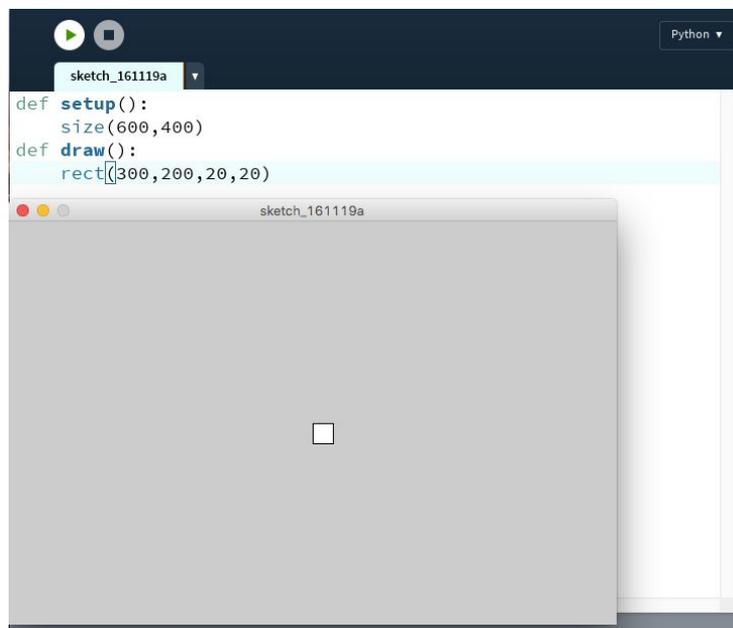


```
def setup():  
  
def draw():
```

Una vez escrito el código prueba con el siguiente programa y ejecútalo en Processing. Para ejecutarlo tendrás que darle al botón de play que hay encima del nombre del archivo.

```
def setup():  
    size(600,400)  
def draw():  
    rect(300,200,20,20)
```

Si lo ejecutas deberías conseguir lo siguiente:



Como podrás ver, al ejecutar un código en Processing se muestra, si el código es correcto y así lo indica, una ventana emergente del tamaño que hemos indicado, que es el área de

visualización. Esta pantalla en ocasiones puede quedar tras el editor de código, si no la ves prueba a moverlo para ver si realmente está escondida.

El código es sencillo, hemos indicado lo siguiente:

```
size(600,400)
```

Creamos una ventana de tamaño 600 píxels de ancho por 400 píxels de alto. El punto de partida (es decir, el punto 0, 0) es la esquina superior izquierda. Esto quiere decir que las dimensiones en x crecen hacia la derecha y en y hacia abajo.

```
rect(300,200,20,20)
```

Dibujamos un rectángulo en la posición 300 a lo ancho (es decir, en el eje x) y 200 a lo alto (es decir, en el eje y). El rectángulo tendrá 20 píxels de ancho y 20 píxels de alto.

En el caso del rectángulo, las dimensiones se cuentan tanto para la posición como para el tamaño desde la esquina superior izquierda del mismo.

Cambia las medidas de la pantalla y del rectángulo y prueba con ello a ver cómo cambia la imagen.

Una cosa algo complicada para mucha gente es interiorizar que en Processing las medidas verticales, es decir, el eje y, crece hacia abajo y decrece hacia arriba. En nuestro ejemplo y = 0 es la parte de arriba, en cambio y = 400 es la parte de abajo de la pantalla.

Prueba el siguiente código para ver algunos cambios:

```
def setup():  
    size(600,400)  
    background(255)  
def draw():  
    fill(0,0,255)  
    rect(300,200,20,20)
```

Como habrás podido comprobar, ahora el fondo de la ventana es blanco y el rectángulo es azul. También, como podrás ver, estamos llamando todo el rato a funciones, algunas sin parámetros y otras con parámetros:

```
background(255)
```

Esta función cambia el color del fondo de la ventana. Está utilizada en escala de grises, siendo 0 negro y 255 blanco.

```
fill(0,0,255)
```

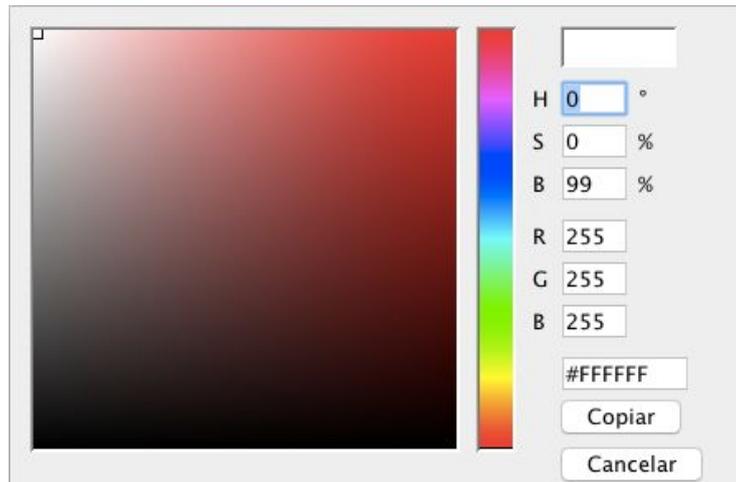
Esta función cambia el relleno de lo que tenga debajo. Está utilizada con colores **R, G, B**, es decir, red, green, blue (o rojo, verde y azul en castellano). Cada uno de ellos toma valores desde 0 hasta 255, siendo 0 la ausencia de ese color y 255 el máximo posible. Si jugamos con diferentes combinaciones veremos diferentes colores, prueba a cambiar los tres números de dentro del paréntesis.

Tres ceros indican el negro y tres 255 indican el blanco.

Para cambiar las líneas del borde de una figura hay que usar el comando *stroke*:

```
stroke(0,0,255)
```

Processing incluye un asistente para generar colores R,G,B, sólo tienes que ir al menú Herramientas y seleccionar Selector de colores...



En la misma podemos seleccionar colores y en el lateral derecho veremos la combinación R,G,B con la que se corresponde el color seleccionado.

Creando animaciones

Ahora bien, la diferencia entre lo que hemos hecho hasta ahora y un programa de diseño gráfico (o un simple *paint*) no es muy grande. Ya que podemos usar código lo suyo sería generar un programa que no se pudiese reproducir con un programa de diseño gráfico.

Vamos a ver cómo con código podemos hacer cosas ya algo más interesantes. Empecemos haciendo que, en la zona del bucle (*draw*) el cuadrado vaya cambiando de posición en cada iteración. Para ello vamos a utilizar una opción de Processing directa que genera números aleatorios: `random`.

Recuerda que en Python hasta ahora para usar un *random* teníamos que importar una biblioteca. En Processing viene incorporada ya que es habitual generar números aleatorios. Vamos, simplemente, a poner en la posición del *rect* que aparezca aleatoriamente en posiciones dentro del valor de nuestro área de visualización.

Para elegir un número aleatorio tengo dos opciones:

```
random(300)
random(100, 300)
```

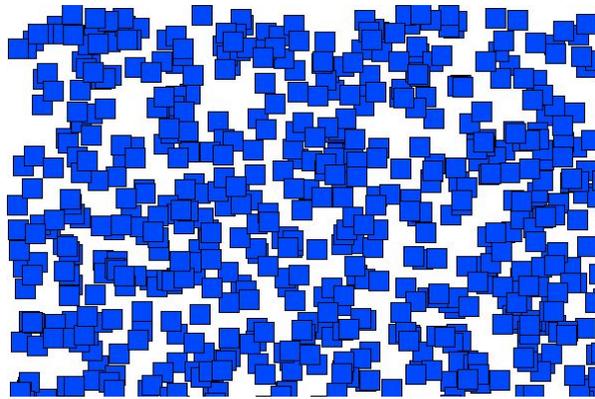
La primera opción genera un número aleatorio desde 0 hasta 300. El cero no hace falta indicarlo.

La segunda opción genera un aleatorio entre 100 y 200.

Prueba el siguiente código:

```
def setup():
    size(600, 400)
    background(255)
def draw():
    fill(0, 0, 255)
    rect(random(600), random(400), 20, 20)
```

Interesante, ¿verdad?:



Verás que los rectángulos azules se generan bastante rápido. Vamos a hacer un simple cambio para que veas una diferencia muy importante. Prueba el siguiente código:

```
def setup():  
    size(600,400)  
def draw():  
    background(255)  
    fill(0,0,255)  
    rect(random(600),random(400),20,20)
```

Ahora sólo hay un cuadrado que cambia de posición a toda velocidad. Este pequeño cambio produce un efecto muy importante. Cuando usamos la función *background* estamos pidiendo a Processing que limpie la pantalla y ponga un color. Si lo ponemos en el *setup* pondrá ese color sólo una vez, en el momento en que llegue a esa instrucción.

En cambio, al ponerlo al principio del *draw*, el fondo se pondrá del color indicado cada vez que volvamos a iniciar una iteración del bucle, por ello el rectángulo anteriormente dibujado desaparece y sólo tenemos uno.

Es importante tener esto claro, vamos a probar algún cambio más, para que entiendas la importancia de dónde sitúo yo la instrucción de código. Intenta descubrir la diferencia entre estos dos códigos.

Código 1:

```
def setup():  
    background(255)  
    size(600,400)  
def draw():  
    fill(0,0,255)  
    rect(random(600),random(400),20,20)
```

Código 2:

```
def setup():  
    background(255)  
    size(600,400)  
def draw():  
    rect(random(600),random(400),20,20)  
    fill(0,0,255)
```

¿Has visto la diferencia? Hay que prestar atención. En el primer código generamos cuadrados azules que van llenando la pantalla y en el segundo también... pero hay uno que es blanco.

Y hay uno que es blanco porque el relleno (*fill*) indica que quieres rellenar una figura de un color, pero rellena la figura que tiene a continuación. Por ello, el primer rectángulo que creamos se crea sin un relleno, por lo que sale blanco (color por defecto). A continuación el programa encuentra un *fill* y posteriormente vuelve a empezar con un *rect*, que ya sí es azul...

Por último antes de continuar, podemos utilizar *random* allá donde queramos, incluso en los colores de rellenos. Prueba el siguiente código:

```
def setup():  
    background(255)  
    size(600,400)  
def draw():  
    fill(random(255),random(255),random(255))  
    rect(random(600),random(400),random(10,60),random(10,60))
```

Una auténtica obra de arte, ¿verdad?

Ahora que ya has trabajado con un rectángulo vamos a aprender a dibujar una circunferencia:

```
def setup():  
    background(255)  
    size(600,400)  
def draw():  
    fill(0,200,200)  
    ellipse(300,200,40,40)
```

En el caso de la *ellipse* los dos primeros parámetros indican dónde sitúa el programa el centro de la misma y los dos últimos cuánto miden sus diámetros de ancho y de alto. Si ponemos dos valores iguales tendremos un círculo y si ponemos dos diferentes tendremos una elipse.

Ahora que tenemos una circunferencia vamos a intentar que se mueva por la pantalla. Empecemos haciendo algo sencillo: el círculo debe moverse desde la parte izquierda hacia la derecha. Para ello vamos a dar un primer paso, observa y prueba el siguiente código:

```
def setup():
    background(255)
    size(600,400)
def draw():
    posicionX = 20
    posicionY = 200
    fill(0,200,200)
    ellipse(posicionX,posicionY,40,40)
```

Perfecto, ahora tenemos la posición de la elipse guardada en variables, una para X y otra para Y. El siguiente paso sería ir cambiando el valor de la variable *posicionX* para que el círculo fuese cambiando de posición. Observa y prueba el siguiente código:

```
def setup():
    background(255)
    size(600,400)
def draw():
    posicionX = 20
    posicionY = 200
    fill(0,200,200)
    ellipse(posicionX,posicionY,40,40)
    posicionX = posicionX + 1
```

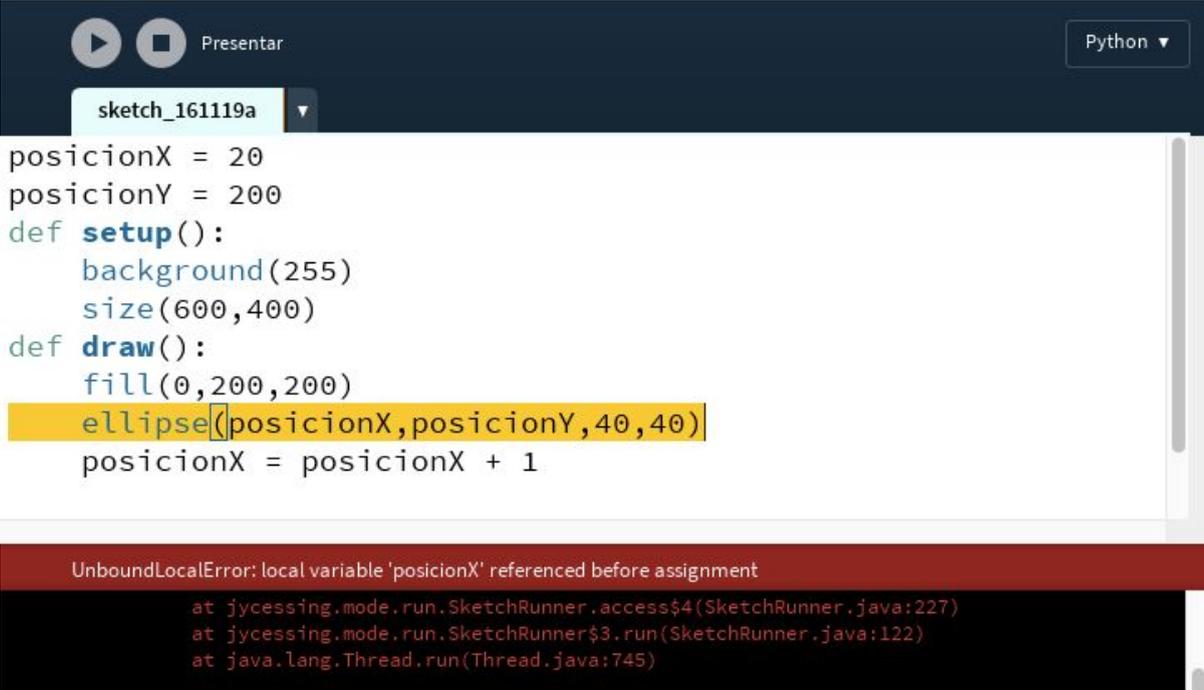
En el bucle (*draw*) estamos creando una variable con un valor y estamos usando esa variable para la posición del círculo. Posteriormente estamos sumando 1 a la variable *posicionX* con la intención de mover el círculo. El problema es que *draw* es un bucle y una vez que termina vuelve a empezar y vuelve a declarar la variable *posicionX* con el valor que tenía inicialmente, por lo que nunca moveremos nuestra *ellipse*.

Ámbito de una variable

Nos viene perfecto el ejemplo anterior para ver cómo podemos utilizar variables en diferentes ámbitos. Necesitamos declarar la variable *posiciónX* fuera del ámbito *draw* y cambiar el valor dentro del ámbito *draw*:

```
posicionX = 20
posicionY = 200
def setup():
    background(255)
    size(600,400)
def draw():
    fill(0,200,200)
    ellipse(posicionX,posicionY,40,40)
    posicionX = posicionX + 1
```

Ahora ya tenemos las variables fuera del ámbito *draw*, vamos a ver qué ocurre, prueba el código anterior:



```
Presentar Python
sketch_161119a
posicionX = 20
posicionY = 200
def setup():
    background(255)
    size(600,400)
def draw():
    fill(0,200,200)
    ellipse(posicionX,posicionY,40,40)
    posicionX = posicionX + 1
```

UnboundLocalError: local variable 'posicionX' referenced before assignment

```
at jycessing.mode.run.SketchRunner.access$4(SketchRunner.java:227)
at jycessing.mode.run.SketchRunner$3.run(SketchRunner.java:122)
at java.lang.Thread.run(Thread.java:745)
```

Resulta que tenemos un error, tenemos las variables creadas antes de definir nuestros dos ámbitos (*setup* y *draw*) y luego queremos usarlas desde el ámbito *draw* y Processing nos dice: *local variable "posicionX" referenced before assignment*.

Dicho en castellano: has intentado usar la variable *posicionX* antes de su asignación (es decir, de su creación).

Vamos a dejar claro cómo podemos usar las variables en Python:

- Las variables pueden ser locales y globales. Si no se indica nada las variables son locales.
- Una variable local puede ser leída desde cualquier ámbito y se puede utilizar su valor, pero no puede cambiarse su valor.
- Una variable global puede ser usada y cambiada desde cualquier ámbito, pero hace falta indicar que queremos usar la variable global.

En nuestro código el problema es que no hemos indicado que queremos, en el *draw*, utilizar dos variables globales, por lo que Python, al usar la expresión *posicionX = posicionX + 1* interpreta que dicha variable es local y trata de buscar su valor en ese ámbito, cosa imposible porque no se trata de una variable local.

Para solucionarlo basta con indicar en un ámbito que queremos usar una variable global. En nuestro caso basta con indicar, antes de utilizarla, que al mencionar la variable *posicionX* nos estamos refiriendo a una variable global (*global posicionX*):

```
posicionX = 20
posicionY = 200
def setup():
    background(255)
    size(600,400)
def draw():
    global posicionX
    fill(0,200,200)
    ellipse(posicionX,posicionY,40,40)
    posicionX = posicionX + 1
```

Prueba el código anterior antes de seguir.

Como habrás podido comprobar, el círculo se mueve, pero deja rastro. ¿Por qué puede ser? Pista: tiene que ver con *background*. ¿Se te ocurre una solución?

Efectivamente, basta con cambiar dónde situamos el fondo, prueba a cambiar `background(255)` al ámbito `draw` a ver si así se soluciona.

Ya está solucionado. Ahora sólo hay un último problema. El movimiento del círculo continúa aunque llegue al final del área de visualización y no sería mala idea conseguir que el círculo se dé la vuelta al llegar al final.

Cambios en el movimiento

Para conseguirlo vamos a utilizar también una variable global para la cantidad de píxeles que movemos la circunferencia (en el ejemplo era 1):

```
posicionX = 20
posicionY = 200
movimiento = 1
def setup():
    size(600,400)
def draw():
    background(255)
    global posicionX
    global movimiento
    fill(0,200,200)
    ellipse(posicionX,posicionY,40,40)
    posicionX = posicionX + movimiento
```

Prueba el código anterior y verás que funciona exactamente igual que el primero.

Ahora sólo queda incluir, al final del programa, un condicional que haga que si el círculo va a salirse del área de visualización en vez de sumar la variable *movimiento* se empiece a restar o bien que **siga sumándola pero la variable pase a ser negativa**.

Para realizar el condicional hay que pensar un poquito en dónde están los bordes del área de visualización y dónde está situado el círculo (en su centro). Ésto hace que tengamos que jugar con el radio del círculo para simular el rebote. Lo vas a entender a la hora de ejecutar el próximo código, intenta entenderlo:

```
posicionX = 20
posicionY = 200
movimiento = 1
def setup():
    background(255)
    size(600,400)
def draw():
    background(255)
    global posicionX
    global movimiento
    fill(0,200,200)
    ellipse(posicionX,posicionY,40,40)
    posicionX = posicionX + movimiento
```

```
if posicionX > 600:  
    movimiento = -movimiento
```

Como habrás visto el círculo rebota más allá del borde. Esto ocurre porque la *posicionX* del círculo está en su centro y le hemos dicho que rebote si su centro está más allá de la posición 600 en x.

Antes de seguir te lanzo unos retos:

- Intenta que el círculo se mueva más rápido.
- Intenta que el círculo rebote justo en el borde.
- Intenta que el círculo también rebote en el borde izquierdo.

¿Lo has conseguido? Si no es así enseguida podrás ver el código, pero antes un apunte: al ancho del área de visualización se le llama *width* y al alto *height*.

Veamos cómo hacemos todas las cosas anteriormente mencionadas:

```
posicionX = 20  
posicionY = 200  
movimiento = 3  
def setup():  
    background(255)  
    size(600,400)  
def draw():  
    background(255)  
    global posicionX  
    global movimiento  
    fill(0,200,200)  
    ellipse(posicionX,posicionY,40,40)  
    posicionX = posicionX + movimiento  
    if posicionX > width-20 or posicionX < 20:  
        movimiento = -movimiento
```

Para que el círculo se mueva más rápido: *movimiento* = 3 o cualquier otro valor.

Para que el círculo rebote justo en el borde hay que restarle el radio en el condicional: *width - 20*

Para que rebote en ambos bordes hay que incluir un *or* en el *if* con la condición para que rebote el círculo en la izquierda (*posicionX < 20*, siendo 20 el resultado de sumar a 0 el radio del círculo).

Ahora que ya hemos conseguido nuestra primera secuencia en movimiento vamos a ver cómo podría hacer que este código fuese más funcional utilizando más variables y menos números en los parámetros:

```
posicionX = 20
posicionY = 200
movimiento = 3
def setup():
    background(255)
    size(600,400)
def draw():
    background(255)
    global posicionX
    global movimiento
    diametro = 40
    fill(0,200,200)
    ellipse(posicionX,posicionY,diametro,diametro)
    posicionX = posicionX + movimiento
    if posicionX > width-diametro/2 or posicionX < diametro/2:
        movimiento = -movimiento
```

Utilizar variables y no magnitudes directamente en las funciones ayuda a parametrizarlas, es decir, conseguir que puedas cambiar todo el programa únicamente cambiando el valor de una variable. Con el programa anterior podemos hacer que la elipse sea más grande o más pequeña o que el tamaño del área de visualización crezca o decrezca sin que la funcionalidad del mismo se vea afectada.

Llegados a este punto puedes realizar tus actividades.

Instrucciones de código usadas

```
#Función inicial setup
def setup():
#Función inicial bucle
def draw():
#Tamaño del área de visualización en píxeles
size(ancho,alto)
#Dibujar un rectángulo
rect(posicion_en_x,posicion_en_y,ancho,alto)
#Fondo del área de visualización
background(color)
#Relleno de uno o varios elementos
fill(color)
#Color del borde de un elemento
stroke(color)
#Generar valores aleatorios desde cero hasta un valor
random(valor)
#Generar valores aleatorios entre dos valores
random(valor1, valor2)
#Dibujar una elipse
ellipse(posicion_en_x,posicion_en_y,diametro_en_x,diametro_en_y)
#Ancho del área de visualización
width
#Alto del área de visualización
height
```

Reto 1

Consigue que el círculo de la actividad anterior se desplace según una dirección no horizontal (es decir, que cambie su posición en x e y) y que rebote en los cuatro bordes.

Reto 2

Consigue que el círculo sea algo más visual: un muñeco de nieve, por ejemplo. Debes dibujar con círculos y rectángulos y tener en cuenta que todos deben tener en su posición la misma variable para que pueda moverse. A dicha variable se le puede sumar algún valor:

```
ellipse(posicionX, posicionY-30, 30, 30)  
ellipse(posicionX, posicionY, 40, 40)
```