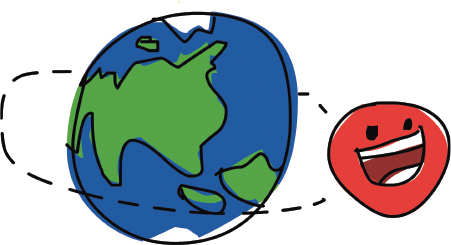
Desconocido

### Añadir alguna acción

Ahora que tenemos configurada la clase Ball , es hora de animar la pelota. Haremos que se mueva, rebote y cambie de dirección.



#### Mover la pelota

Para mover la pelota, modifica la función draw como se indica a continuación:

class Ball:  
 def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
  
 def draw(self):  
 self.canvas.move(self.id, 0, -1)

Como \_\_init\_\_ guardó el parámetro canvas como variable del objeto canvas , utilizamos esa variable con self.canvas y llamamos a la función move en el lienzo.

Pasamos tres parámetros a move : el id del óvalo, y los números 0 y -1 . El 0 indica a la bola que no se mueva horizontalmente, y el -1 indica a la bola que se mueva 1 píxel hacia arriba en la pantalla.

Hacemos este pequeño cambio porque es beneficioso ir probando cosas sobre la marcha. Imagina que escribimos todo el código de nuestro juego de una vez y luego descubrimos que no funciona. ¿Por dónde empezaríamos a buscar para averiguar qué falló?

También cambiaremos el bucle principal al final de nuestro programa. En el bloque del bucle while (nuestro bucle principal), añadimos una llamada a la función draw del objeto ball , de esta forma:

while True:  
 ball.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

Si ejecutas este código ahora, la pelota debería moverse por el lienzo y desaparecer de la parte superior de la pantalla: los comandos update \_idletasks y update indican a tkinter que se dé prisa en dibujar lo que hay en el lienzo.

El comando time.sleep es una llamada a la función sleep del módulo time , que le dice a Python que duerma durante una centésima de segundo ( 0.01 ). Esto garantiza que nuestro programa no se ejecute tan rápido que la bola desaparezca antes incluso de que la veas.

Este bucle está diciendo básicamente: "Mueve la bola un poco, vuelve a dibujar la pantalla con la nueva posición, duerme un momento y vuelve a empezar".

**NOTA**

*Puede que veas mensajes de error escritos en la Shell de Python cuando cierres la ventana del juego. Esto se debe a que al cerrar la ventana se interrumpe lo que tkinter está haciendo, y Python se está quejando de ello. Podemos ignorar con seguridad este tipo de errores.*

Ahora tu código de juego debería tener este aspecto:

from tkinter import \*  
import random  
import time  
  
class Ball:  
 def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
   
 def draw(self):  
 self.canvas.move(self.id, 0, -1)  
  
tk = Tk()  
tk.title('Bounce Game')  
tk.resizable(0, 0)  
tk.wm\_attributes('-topmost', 1)  
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)  
canvas.pack()  
tk.update()  
  
ball = Ball(canvas, 'red')  
  
while True:  
 ball.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

Si ejecutas este código, la pelota empezará a moverse hacia arriba y saldrá por la parte superior de la ventana.

#### Hacer que la pelota rebote

Una pelota que desaparece de la parte superior de la pantalla no es especialmente útil para nuestro juego, así que hagamos que rebote. En primer lugar, guardaremos unas cuantas variables de objeto adicionales en la función de inicialización de la clase Ball , como se indica a continuación:

def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
 self.x = 0  
 self.y = -1  
 self.canvas\_height = self.canvas.winfo\_height()

Hemos añadido tres líneas más a nuestro programa. Con self.x = 0 , establecemos la variable de objeto x en 0; con self.y = -1 , establecemos la variable y en -1. Por último, fijamos la variable de objeto canvas\_height llamando a la función de lienzo winfo\_height . Esta función devuelve la altura actual del lienzo.

A continuación, volvemos a cambiar la función draw :

def draw(self):  
 self.canvas.move(self.id, self.x, self.y)  
 pos = self.canvas.coords(self.id)  
 if pos[1] <= 0:  
 self.y = 1   
 if pos[3] >= self.canvas\_height:  
 self.y = -1

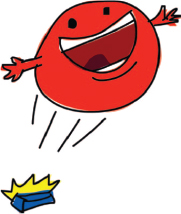
Primero cambiamos la llamada a la función move del lienzo pasando las variables de objeto x y y . A continuación, creamos una variable llamada pos llamando a la función coords del lienzo. Esta función devuelve las *coordenadas* *x e* *y* actuales de cualquier cosa dibujada en el lienzo, siempre que conozcas su número identificativo. En este caso, pasamos a coords la variable objeto id , que contiene el identificador del óvalo.

La función coords devuelve las coordenadas como una lista de cuatro números. Si imprimimos los resultados de llamar a esta función, veremos algo como esto

print(self.canvas.coords(self.id))  
[255.0, 29.0, 270.0, 44.0]

Los dos primeros números de la lista ( 255.0 y 29.0 ) contienen las coordenadas superiores izquierdas del óvalo ( *x1* e *y1* ), y el segundo par ( 270.0 y 44.0 ) son las coordenadas inferiores derechas *x2* e *y2* . Utilizaremos estos valores en las siguientes líneas de código.

Continuamos nuestro código viendo si la coordenada *y1* (¡esa es la parte superior de la bola!) es menor o igual que 0. Si es así, establecemos la variable de objeto y en 1 . En efecto, estamos diciendo que si golpeas la parte superior de la pantalla, tkinter dejará de restar 1 a la posición vertical, y la bola dejará de moverse hacia arriba (es una versión simple de la *detección de colisiones* ).

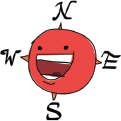


A continuación, vemos si la coordenada *y2* (¡esa es la parte inferior de la bola!) es mayor o igual que la variable canvas\_height . Si lo es, volvemos a poner la variable de objeto y en -1 . Ahora la bola dejará de moverse hacia abajo y volverá a subir.

Ejecuta ahora este código y la pelota debería rebotar arriba y abajo por el lienzo hasta que cierres la ventana.

#### Cambiar la dirección inicial de la bola

Hacer que una pelota rebote lentamente hacia arriba y hacia abajo no es un gran juego, así que mejoremos un poco las cosas cambiando la dirección inicial de la pelota, es decir, el ángulo en que se mueve cuando empieza el juego.



En la función \_\_init\_\_ , cambia estas dos líneas

self.x = 0  
 self.y = -1

por el código siguiente (asegúrate de que tienes el número correcto de espacios -hay ocho- al principio de cada línea):

starts = [-3, -2, -1, 1, 2, 3]  
 self.x = random.choice(starts)  
 self.y = 3

Comenzamos creando la variable starts con una lista de seis números. A continuación, establecemos el valor de la variable x , utilizando la función random.choice , que devuelve un elemento aleatorio de una lista. Al utilizar esa función, x puede ser cualquier número de la lista, de -3 a 3.

Por último, cambiamos y a -3 (para que la bola empiece el juego moviéndose hacia arriba). Ahora nuestra bola puede moverse en cualquier dirección, pero tenemos que hacer algunos añadidos más para asegurarnos de que no se desvanecerá por el lateral de la pantalla. Añade la siguiente línea al final de la función \_\_init\_\_ para guardar la anchura del lienzo en una nueva variable de objeto canvas\_width :

self.canvas\_width = self.canvas.winfo\_width()

Utilizaremos esta nueva variable de objeto en la función draw para ver si la bola ha golpeado el lado izquierdo o derecho del lienzo:

if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.x = self.x \* -1

Si la posición más a la izquierda de la bola es menor o igual que 0, o la posición más a la derecha de la bola es mayor o igual que la anchura del lienzo, hacemos este pequeño y extraño cálculo self.x = self.x \* -1 . La variable x se establece en el valor actual de x multiplicado por -1. Así, si el valor de x es 2, el nuevo valor será -2. Si el valor de x es -3, el nuevo valor será 3. Por tanto, cuando la pelota golpee un lado, rebotará en la dirección opuesta. Podemos hacer una comprobación similar para la parte superior e inferior del lienzo, utilizando la altura del lienzo y multiplicando la variable y por -1. Tu función draw debería tener ahora este aspecto:

def draw(self):  
 self.canvas.move(self.id, self.x, self.y)  
 pos = self.canvas.coords(self.id)  
 if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.x = self.x \* -1  
 if pos[1] <= 0 or pos[3] >= self.canvas\_height:  
 self.y = self.y \* -1

El programa completo debería tener este aspecto:

from tkinter import \*  
import random  
import time  
  
class Ball:  
 def \_\_init\_\_(self, canvas, color):  
 self.canvas = canvas  
 self.id = canvas.create\_oval(10, 10, 25, 25, fill=color)  
 self.canvas.move(self.id, 245, 100)  
 starts = [-3, -2, -1, 1, 2, 3]  
 self.x = random.choice(starts)  
 self.y = -3  
 self.canvas\_height = self.canvas.winfo\_height()  
 self.canvas\_width = self.canvas.winfo\_width()  
   
 def draw(self):  
 self.canvas.move(self.id, self.x, self.y)  
 pos = self.canvas.coords(self.id)  
 if pos[0] <= 0 or pos[2] >= self.canvas\_width:  
 self.x = self.x \* -1  
 if pos[1] <= 0 or pos[3] >= self.canvas\_height:  
 self.y = self.y \* -1  
  
tk = Tk()  
tk.title('Bounce Game')  
tk.resizable(0, 0)  
tk.wm\_attributes('-topmost', 1)  
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)  
canvas.pack()  
tk.update()  
  
ball = Ball(canvas, 'red')  
  
while True:  
 ball.draw()  
 tk.update\_idletasks()  
 tk.update()  
 time.sleep(0.01)

Guarda y ejecuta el código, y la pelota debería rebotar por la pantalla sin desaparecer.

[anterior](ch11_4.html)[Subtema 5 de 7: (Ver todo)](ch11.html)[siguiente](ch11_6.html)