Desconocido

## 10 Utilizar tkinter para mejorar los gráficos



El problema de usar una tortuga para dibujar es . . . que . . . las tortugas . . . . son . . . realmente . . . lentas. Incluso cuando una tortuga va a toda velocidad, no es muy rápida. Aunque esto no es realmente un problema para las tortugas, sí lo es para los gráficos por ordenador.

Los gráficos por ordenador normalmente necesitan moverse rápido. Si juegas en una videoconsola o en un ordenador, piensa por un momento en los gráficos que ves en la pantalla. Los gráficos bidimensionales (2D) son planos: los personajes generalmente sólo se mueven arriba y abajo o a izquierda y derecha, como en muchos juegos de Nintendo y teléfonos  . En los juegos pseudotridimensionales (3D) -los que son casi 3D- las imágenes parecen un poco más reales, pero los personajes generalmente se mueven sólo en relación con un plano plano (esto también se conoce como *gráficos isométricos* ). Por último, tenemos los juegos 3D, con gráficos que intentan imitar la realidad. Tanto si los juegos utilizan gráficos 2D, pseudo-3D o 3D, todos tienen algo en común: la necesidad de dibujar en la pantalla del ordenador con gran rapidez.



Si nunca has intentado crear tu propia animación, prueba con este sencillo proyecto:

1. Coge un bloc de papel en blanco, y en la esquina inferior de la primera página, dibuja algo (tal vez una figura de palo).
2. En la esquina de la página siguiente, dibuja la misma figura de palo, pero mueve ligeramente la pierna.
3. En la página siguiente, vuelve a dibujar la figura de palo, con la pierna un poco más movida.
4. Pasa gradualmente por cada página, dibujando una figura de palo modificada en la esquina inferior.

Cuando hayas terminado, pasa rápidamente las páginas y verás cómo se mueve la figura de palo. Éste es el método básico utilizado en toda animación, ya sean dibujos animados o videojuegos. Se dibuja una imagen y luego se vuelve a dibujar con un ligero cambio para crear la ilusión de movimiento. Para que una imagen parezca que se mueve, tienes que mostrar cada *fotograma -o* trozo de la animación- muy rápidamente.

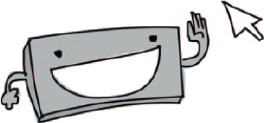
Python ofrece diferentes formas de crear gráficos. Además del módulo turtle , puedes utilizar módulos *externos* (que debes instalar por separado), así como el módulo tkinter , que ya deberías tener en tu instalación estándar de Python. El módulo tkinter puede utilizarse para crear aplicaciones completas, como un sencillo procesador de textos, así como dibujos. En este capítulo, exploraremos el uso de tkinter para crear gráficos.

### Crear un botón clicable

Para nuestro primer ejemplo, utilizaremos el módulo tkinter para crear una aplicación básica con un botón. Introduce este código:

>>> from tkinter import \*  
>>> tk = Tk()  
>>> btn = Button(tk, text='click me')  
>>> btn.pack()

En la primera línea, importamos el contenido del módulo tkinter . La línea from *module-name* import \* nos permite utilizar el contenido de un módulo sin utilizar su nombre. En cambio, al utilizar import turtle en los ejemplos anteriores, necesitábamos incluir el nombre del módulo para acceder a su contenido, de esta forma:



import turtle  
t = turtle.Turtle()

Cuando utilizamos import \* , no necesitamos llamar a turtle.Turtle , como hicimos en los [Capítulos 4](ch04.xhtml#ch04) y [9 .](ch09.xhtml#ch09) Esto es muy útil cuando utilizas módulos con muchas clases y funciones, porque reduce la cantidad que tienes que escribir:

from turtle import \*  
t = Turtle()

En la siguiente línea de nuestro ejemplo del botón, creamos una variable que contiene un objeto de la clase Tk con tk = Tk() , igual que creamos un objeto Turtle para la tortuga. El objeto tk crea una ventana básica a la que luego podemos añadir otras cosas, como botones, cuadros de entrada o un lienzo en el que dibujar. Ésta es la clase principal que proporciona el módulo tkinter ; sin crear un objeto de la clase Tk , no podrás hacer gráficos ni animaciones.

En la tercera línea, creamos un botón con btn = Button , y pasamos la variable tk como primer parámetro y "haz clic en mí" como texto que mostrará el botón con text=’click me’ . Aunque hemos añadido este botón a la ventana, no se mostrará hasta que introduzcas la línea btn.pack() , que indica al botón que aparezca. También alinea todo correctamente en la pantalla si  hay otros botones u objetos que mostrar. El resultado debería ser algo parecido a [la Figura 10-1 .](ch10.xhtml#ch10fig01)

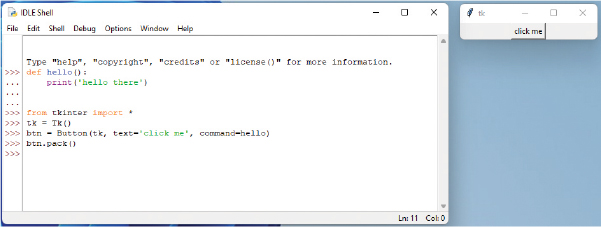


Figura 10-1*: Una aplicación tkinter con un solo* botón

Ahora mismo, el botón Haz clic en mí no hace gran cosa. Puedes pulsarlo todo el día, pero no pasará nada hasta que cambiemos un poco el código. (¡Asegúrate de cerrar la ventana que has creado antes!)

Primero, creamos una función para imprimir un texto:

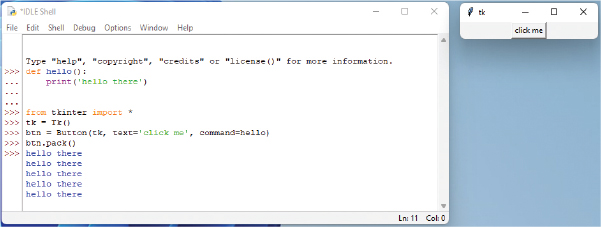
>>> def hello():  
 print('hello there')

A continuación, modificamos nuestro ejemplo para utilizar esta nueva función:

>>> from tkinter import \*  
>>> tk = Tk()  
>>> btn = Button(tk, text='click me', command=hello)  
>>> btn.pack()

Sólo hemos hecho un ligero cambio en la versión anterior de este código, añadiendo el parámetro command , que indica a Python que utilice la función hello cuando se pulse el botón.

Ahora, cuando pulses el botón, verás escrito "hola" en la Shell de Python. Esto aparecerá cada vez que se pulse el botón. En [la Figura 10-2](ch10.xhtml#ch10fig02) , he pulsado el botón cinco veces.



Figura*10-2: Pulsar el* botón

Ésta es la primera vez que utilizamos parámetros con nombre en cualquiera de nuestros ejemplos de código, así que vamos a hablar un poco de ellos antes de continuar con nuestro dibujo.

### Uso de parámetros con nombre

*Los* parámetros con nombre son como los parámetros normales, pero en lugar de utilizar el orden específico de los valores proporcionados a una función para determinar qué valor pertenece a qué parámetro (el primer valor es el primer parámetro, el segundo valor es el segundo parámetro, etc.), nombramos explícitamente los valores para que puedan aparecer en cualquier orden.

A veces las funciones tienen muchos parámetros, y no siempre necesitamos proporcionar un valor para cada uno. Con los parámetros con nombre, podemos proporcionar valores sólo para los parámetros a los que necesitamos dar valores.

Por ejemplo, supongamos que tenemos una función person que toma dos parámetros, width y height :

>>> def person(width, height):  
 print(f'I am {width} feet wide, {height} feet high')

Normalmente, podríamos llamar a esta función así:

>>> person(4, 3)  
I am 4 feet wide, 3 feet high

Utilizando parámetros con nombre, podríamos llamar a esta función y especificar el nombre del parámetro con cada valor:

>>> person(height=3, width=4)  
I am 4 feet wide, 3 feet high

Los parámetros con nombre serán especialmente útiles a medida que hagamos más cosas con el módulo tkinter .

### Crear un lienzo para dibujar

Los botones son buenas herramientas, pero no son especialmente útiles cuando queremos dibujar cosas en la pantalla. A la hora de dibujar, necesitamos un componente diferente: un objeto canvas , que es un objeto de la clase Canvas (proporcionada por el módulo tkinter ).

Al crear un lienzo, pasamos la anchura y la altura (en píxeles) del lienzo a Python. Por lo demás, el código es similar al del botón. Por ejemplo

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=500, height=500)  
>>> canvas.pack()

Como en el ejemplo del botón, aparecerá una ventana cuando introduzcas tk = Tk() . En la última línea, empaquetamos el lienzo con canvas.pack() , que aplica los cambios al tamaño del lienzo (una anchura de 500 píxeles y una altura de 500 píxeles, como se especifica en la tercera línea de código).

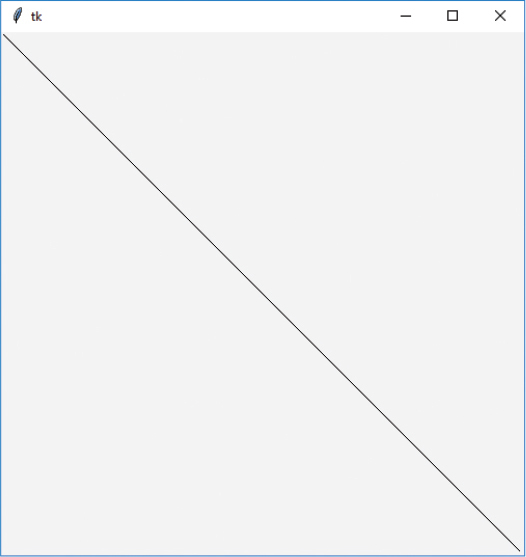
También como en el ejemplo del botón, la función pack indica al lienzo que se muestre en la posición correcta dentro de la ventana. Si no se llama a pack , nada se mostrará correctamente.



### Dibujar líneas

Para dibujar una línea en el lienzo, utilizamos coordenadas de píxel. *Las coordenadas* determinan las posiciones de los píxeles en una superficie. En un lienzo tkinter , las coordenadas describen a qué distancia del lienzo (de izquierda a derecha) y a qué distancia del lienzo (de arriba abajo) hay que colocar el píxel.

Por ejemplo, como nuestro lienzo tiene 500 píxeles de ancho por 500 de alto, las coordenadas de la esquina inferior derecha de la pantalla son (500, 500). Para dibujar la línea que se muestra en [la Figura 10-3](ch10.xhtml#ch10fig03) , utilizaríamos las coordenadas inicial (0, 0) y final (500, 500).



Figura*10-3: Dibujar una línea diagonal con tkinter*

Especificamos las coordenadas utilizando la función create\_line , como se muestra aquí:

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=500, height=500)  
>>> canvas.pack()  
>>> canvas.create\_line(0, 0, 500, 500)  
1

La función create\_line devuelve 1, que es un identificador; aprenderemos más sobre ello más adelante. Si hubiéramos hecho lo mismo con el módulo turtle , habríamos necesitado el siguiente código:

>>> import turtle  
>>> turtle.setup(width=500, height=500)  
>>> t = turtle.Turtle()  
>>> t.up()  
>>> t.goto(-250, 250)  
>>> t.down()  
>>> t.goto(500, -500)

En este código, el lienzo tiene 500 píxeles de ancho y 500 de alto, por lo que la tortuga aparece en la posición 250, 250 (en el centro del lienzo). Si utilizamos la función t.goto(-250, 250) , nos estamos moviendo hacia la izquierda 250 y hacia arriba 250 píxeles hacia la parte superior izquierda de la pantalla. Si llamamos a t.goto(500, -500) , nos estamos moviendo hacia la derecha 500 píxeles y hacia abajo 500 píxeles hasta la esquina inferior derecha.

Así que podemos ver que el código de tkinter ya es una mejora. Es ligeramente más corto y más sencillo. Veamos ahora algunas de las funciones disponibles en el objeto canvas que podemos utilizar para crear dibujos más interesantes.

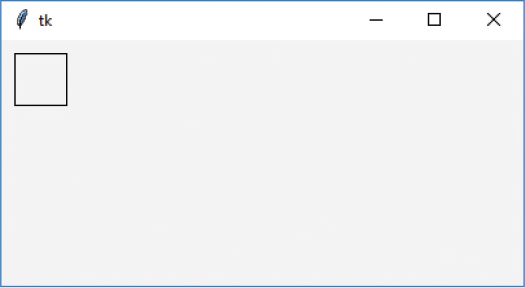
### Cajas de dibujo

Con el módulo turtle , dibujábamos una caja avanzando, girando, avanzando, girando de nuevo, y así sucesivamente. Finalmente, pudimos dibujar una caja rectangular o cuadrada cambiando la distancia a la que nos movíamos hacia delante.

El módulo tkinter hace que sea mucho más fácil dibujar un cuadrado o un rectángulo. Todo lo que necesitas saber son las coordenadas de las esquinas. Prueba el siguiente ejemplo (ya puedes cerrar las demás ventanas):

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=400, height=400)  
>>> canvas.pack()  
>>> canvas.create\_rectangle(10, 10, 50, 50)

En este código, utilizamos tkinter para crear un lienzo de 400 píxeles de ancho por 400 píxeles de alto, y luego dibujamos un cuadrado en la esquina superior izquierda de la ventana, como en [la Figura 10-4 .](ch10.xhtml#ch10fig04)



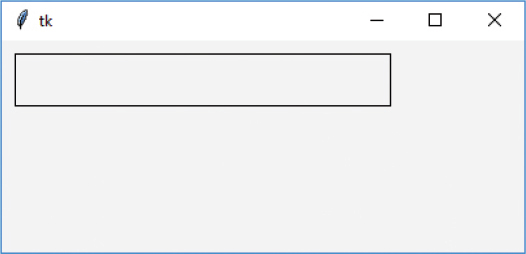
Figura*10-4: Dibujar un rec* uadro

Los parámetros que pasamos a canvas.create\_rectangle en la última línea del código son las coordenadas de las esquinas superior izquierda e inferior derecha del cuadrado. Proporcionamos estas coordenadas como la distancia desde el lado izquierdo del lienzo y la distancia desde la parte superior del lienzo. En este caso, las dos primeras coordenadas (la esquina superior izquierda) están a 10 píxeles de la izquierda y a 10 píxeles de la parte superior; esos son los primeros números: 10, 10 . La esquina inferior derecha del cuadrado está a 50 píxeles de la izquierda y a 50 píxeles de la parte superior -los segundos números: 50, 50 .

Nos referiremos a estos dos conjuntos de coordenadas como *x1* , *y1* y *x2* , *y2 .* Para dibujar un rectángulo, podemos aumentar la distancia de la segunda esquina desde el lado del lienzo (aumentando el valor del parámetro *x2* ), de este modo:

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=400, height=400)  
>>> canvas.pack()  
>>> canvas.create\_rectangle(10, 10, 300, 50)

En este ejemplo, las coordenadas superiores izquierdas del rectángulo (su posición en la pantalla) son ( 10, 10 ), y las coordenadas inferiores derechas son ( 300, 50 ). El resultado es un rectángulo de la misma altura que nuestro cuadrado original (40 píxeles), pero mucho más ancho.



*Figura 10-5: Un rectángulo ancho*

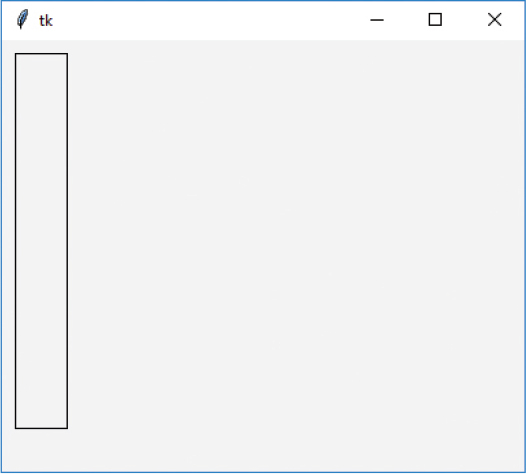
También podemos dibujar un rectángulo aumentando la distancia de la segunda esquina desde la parte superior del lienzo (aumentando el valor del parámetro *y2* ), así

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=400, height=400)  
>>> canvas.pack()  
>>> canvas.create\_rectangle(10, 10, 50, 300)

En esta llamada a la función create\_rectangle , le estamos diciendo a tkinter que:

* Se desplace 10 píxeles a través del lienzo (desde la parte superior izquierda).
* Ir 10 píxeles hacia abajo en el lienzo. Esta es la esquina inicial del rectángulo.
* Dibuja el rectángulo transversalmente hasta 50 píxeles.
* Dibuja hacia abajo hasta 300 píxeles.

El resultado final debe parecerse a [la Figura 10-6 .](ch10.xhtml#ch10fig06)



Figura*10-6: Un rectángulo alto*

### Dibujar muchos rectángulos

Probemos a llenar el lienzo con rectángulos de diferentes tamaños importando el módulo random y creando una función que utilice un número aleatorio para las coordenadas de las esquinas superior izquierda e inferior derecha del rectángulo.

Utilizaremos la función randrange proporcionada por el módulo random . Cuando le damos un número a esta función, devuelve un número entero aleatorio comprendido entre 0 y el número que le demos. Por ejemplo, llamar a randrange(10) devuelve un número entre 0 y 9, randrange(100) devuelve un número entre 0 y 99, y así sucesivamente.

Para utilizar randrange en una función, crea una nueva ventana seleccionando **Archivo ▸ Nuevo Archivo** , e introduce el siguiente código:

from tkinter import \*  
import random  
tk = Tk()  
canvas = Canvas(tk, width=400, height=400)  
canvas.pack()  
  
def random\_rectangle(width, height):  
 x1 = random.randrange(width)  
 y1 = random.randrange(height)  
 x2 = x1 + random.randrange(width)  
 y2 = y1 + random.randrange(height)  
 canvas.create\_rectangle(x1, y1, x2, y2)

Primero definimos nuestra función ( random\_rectangle ) como una función que toma dos parámetros: width y height . A continuación, creamos variables para la esquina superior izquierda del rectángulo utilizando la función randrange , pasando la anchura y la altura como parámetros con x1 = random .randrange(width) y y1 = random.randrange(height) , respectivamente. Con la segunda línea de esta función, estamos diciendo: "Crea una variable llamada x1 y establece su valor en un número aleatorio entre 0 y el valor del parámetro width ."

Las dos líneas siguientes crean variables para la esquina inferior derecha del rectángulo, teniendo en cuenta las coordenadas superiores izquierdas ( x1 o y1 ) y añadiendo un número aleatorio a esos valores. La tercera línea de la función está diciendo efectivamente: "Crea la variable x2 añadiendo un número aleatorio al valor que ya hemos calculado para x1 ."

Por último, con canvas.create\_rectangle , utilizamos las variables x1 , y1 , x2 , y y2 para dibujar el rectángulo en el lienzo.

Para probar nuestra función random\_rectangle , le pasaremos la anchura y la altura del lienzo. Añade el siguiente código debajo de la función que acabas de introducir:

random\_rectangle(400, 400)

Guarda el código que has introducido (selecciona **Archivo ▸ Guardar** e introduce un nombre de archivo como *randomrect.py* ), y luego selecciona **Ejecutar ▸ Ejecutar Módulo .**

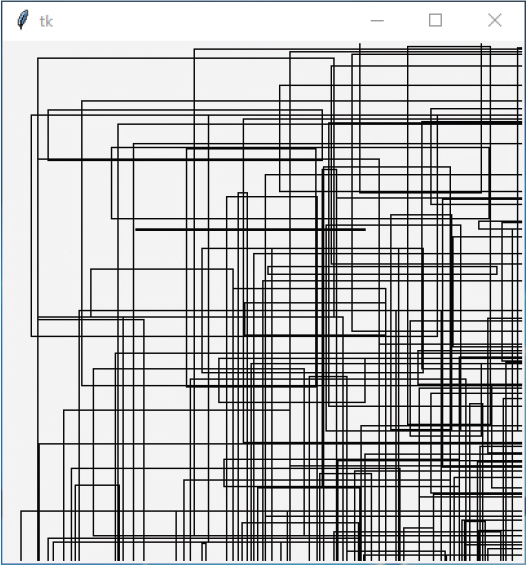
**NOTA**

*Nuestra función random\_rectangle puede dibujar un rectángulo fuera del lateral o de la parte inferior del lienzo.* Esto se*debe a que la esquina superior izquierda del rectángulo puede estar en cualquier parte del lienzo (incluso en la esquina inferior derecha), y no causa ningún error dibujar más allá de la anchura o altura del lienzo.*

Una vez que hayas visto que la función funciona, llena la pantalla de rectángulos creando un bucle para llamar a random\_rectangle varias veces. Probemos con un bucle for de 100 rectángulos aleatorios. Añade el siguiente código, guarda tu trabajo e intenta ejecutarlo de nuevo:

for x in range(0, 100):  
 random\_rectangle(400, 400)

Este código produce un poco de desorden, pero es una especie de arte moderno ( [Figura 10-7](ch10.xhtml#ch10fig07) ).



Figura*10-7: Arte moderno con tkinter*

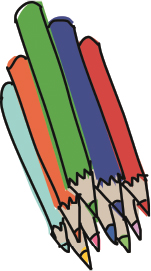
### Configurar el color

Vamos a añadir interés a nuestros gráficos con el color. Cambiaremos la función random\_rectangle para pasar un color para el rectángulo como parámetro adicional ( fill\_color ). Introduce este código en una nueva ventana, y cuando guardes, llama al archivo *colorrect.py :*

from tkinter import \*  
import random  
tk = Tk()  
canvas = Canvas(tk, width=400, height=400)  
canvas.pack()  
  
def random\_rectangle(width, height, fill\_color):  
 x1 = random.randrange(width)  
 y1 = random.randrange(height)  
 x2 = random.randrange(x1 + random.randrange(width))  
 y2 = random.randrange(y1 + random.randrange(height))  
 canvas.create\_rectangle(x1, y1, x2, y2, fill=fill\_color)

La función create\_rectangle toma ahora un parámetro, fill\_color , que especifica el color a utilizar al dibujar el rectángulo.

Podemos pasar colores con nombre a la función de esta forma para crear un montón de rectángulos de colores únicos. Si pruebas este ejemplo, considera la posibilidad de copiar y pegar, después de introducir la primera línea, para ahorrarte el tecleo. Para ello, selecciona el texto a copiar, pulsa CTRL-C para copiarlo, haz clic en una línea en blanco y pulsa CTRL-V para pegar. Añade este código a *colorrect.py* , justo debajo de la función



random\_rectangle(400, 400, 'green')  
random\_rectangle(400, 400, 'red')  
random\_rectangle(400, 400, 'blue')  
random\_rectangle(400, 400, 'orange')  
random\_rectangle(400, 400, 'yellow')  
random\_rectangle(400, 400, 'pink')  
random\_rectangle(400, 400, 'purple')  
random\_rectangle(400, 400, 'violet')  
random\_rectangle(400, 400, 'magenta')  
random\_rectangle(400, 400, 'cyan')

Muchos de estos colores con nombre mostrarán el color que esperas ver, pero otros pueden producir un mensaje de error (dependiendo de si utilizas Windows, macOS o Linux). Pero, ¿qué ocurre con un color personalizado que no es exactamente igual a un color con nombre? Recuerda que en el [Capítulo 9](ch09.xhtml#ch09) establecimos el color de la pluma de la tortuga utilizando porcentajes de los colores rojo, verde y azul. Establecer la cantidad de cada color primario (rojo, verde y azul) que se utilizará en una combinación de colores con tkinter es un poco más complicado, pero lo resolveremos.

Cuando trabajamos con el módulo turtle , creamos el dorado utilizando un 90% de rojo, un 75% de verde y nada de azul. En tkinter , podemos crear el mismo color dorado utilizando esta línea:

random\_rectangle(400, 400, '#e5d800')

La almohadilla ( # ) antes del valor ffd800 indica a Python que estamos proporcionando un número *hexadecimal* . El hexadecimal es una forma de representar números habitual en programación informática. Utiliza una base de 16 (del 0 al 9, y luego de la A a la F) en lugar del decimal, que tiene una base de 10 (del 0 al 9). Si no has aprendido nada sobre bases en matemáticas, debes saber que puedes convertir un número decimal normal a hexadecimal utilizando un *marcador de formato* en una cadena: {:x} (consulta "Incrustar valores en cadenas" en [la página 29](ch03.xhtml#ch03lev1sec4) ). Por ejemplo, para convertir el número decimal 15 a hexadecimal, podrías hacer lo siguiente:

>>> print(f'{15:x}')  
f

Esto es una cadena f con un modificador de formato especial (que es :x ) que indica a Python que convierta el número a hexadecimal.

Para asegurarnos de que nuestro número tiene al menos dos dígitos, podemos cambiar ligeramente el marcador de posición de formato, a esto:

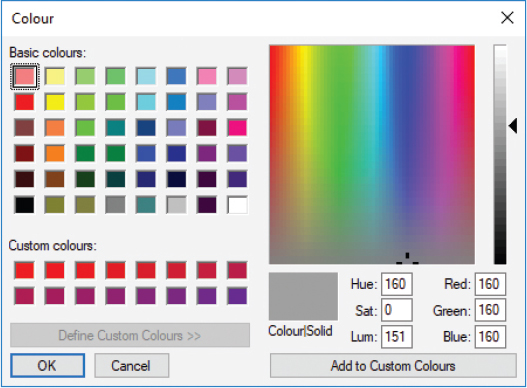
>>> print(f'{15:02x}')  
0f

Esta vez tenemos un modificador de formato ligeramente diferente ( 02x ) que dice que queremos la conversión hexadecimal, pero con dos dígitos (utilizando 0 para cualquier dígito que falte).

El módulo tkinter proporciona una forma sencilla de obtener valores de color hexadecimales. Prueba a ejecutar el siguiente código en IDLE:

from tkinter import \*  
from tkinter import colorchooser  
tk = Tk()  
tk.update()  
print(colorchooser.askcolor())

Este código muestra un selector de colores, que se muestra en la [Figura 10-8 .](ch10.xhtml#ch10fig08) Ten en cuenta que tienes que importar explícitamente el módulo colorchooser porque no está disponible automáticamente en Python cuando utilizas from tkinter import \* .



*Figura 10-8: El selector de color tkinter (que puede tener un aspecto diferente en tu SO)*

Cuando selecciones un color y pulses **OK** , se mostrará una tupla. Esta tupla contiene otra tupla con tres números y una cadena:

>>> print(colorchooser.askcolor())  
((157, 163, 164), '#9da3a4')

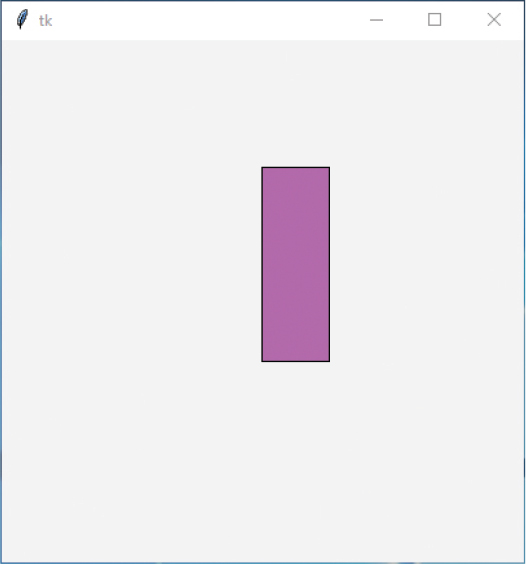
Los tres números representan las cantidades de rojo, verde y azul. En tkinter , la cantidad de cada color primario a utilizar en una combinación de colores se representa mediante un número entre 0 y 255 (que es diferente de utilizar un porcentaje para cada color primario con el módulo turtle ). La cadena de la tupla contiene la versión hexadecimal de esos tres números.

Puedes copiar y pegar el valor de la cadena para utilizarlo, o almacenar la tupla como una variable y utilizar la posición del índice del valor hexadecimal.

Utilicemos la función random\_rectangle para ver cómo funciona, sustituyendo todas las llamadas a random\_rectangle en la parte inferior de *colorrect.* py por el siguiente código:

from tkinter import colorchooser  
c = colorchooser.askcolor()  
random\_rectangle(400, 400, c[1])

Puedes ver el resultado en la [Figura 10-9 .](ch10.xhtml#ch10fig09)

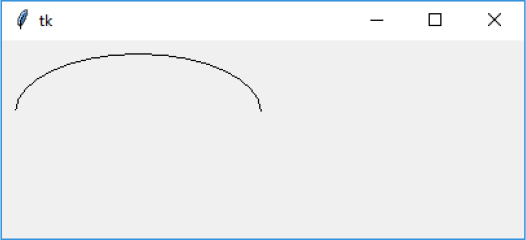


Figura*10-9: Dibujar un rectángulo morado*

### Dibujar arcos

Un *arco* es un segmento de la circunferencia de un círculo o curva. Para dibujar un arco con tkinter , tienes que dibujarlo dentro de un rectángulo utilizando la función create\_arc con un código como éste:

canvas.create\_arc(10, 10, 200, 100, extent=180, style=ARC)



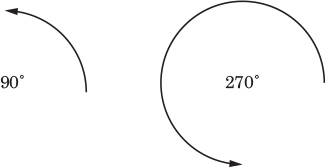
*Figura 10-10: Dibujar un* arco

**NOTA**

*Si has cerrado todas las ventanas de tkinter o has reiniciado IDLE, asegúrate de volver a importar tkinter y luego vuelve a crear el lienzo con este código:*

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=400, height=400)  
>>> canvas.pack()  
>>> canvas.create\_arc(10, 10, 200, 100, extent=180, style=ARC)

Este código coloca la esquina superior izquierda del rectángulo que contendrá el arco en las coordenadas ( 10, 10 ), que son 10 píxeles a lo ancho y 10 píxeles a lo bajo, y su esquina inferior derecha en las coordenadas ( 200, 100 ), es decir, 200 píxeles a lo ancho y 100 píxeles a lo bajo. El siguiente parámetro, extent , se utiliza para especificar los grados del ángulo del arco. Recuerda del [Capítulo 4](ch04.xhtml#ch04) que los grados son una forma de medir la distancia que hay que recorrer alrededor de un círculo. [La Figura 10-11](ch10.xhtml#ch10fig11) muestra ejemplos de dos arcos, en los que recorremos 90 grados y 270 grados alrededor de un círculo.



*Figura 10-11: Arcos de 90 y* 270 grados

El siguiente código dibuja varios arcos diferentes a lo largo de la página para que puedas ver lo que ocurre cuando utilizamos diferentes grados con la función create\_arc :

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=400, height=400)  
>>> canvas.pack()  
>>> canvas.create\_arc(10, 10, 200, 80, extent=45, style=ARC)  
>>> canvas.create\_arc(10, 80, 200, 160, extent=90, style=ARC)  
>>> canvas.create\_arc(10, 160, 200, 240, extent=135, style=ARC)  
>>> canvas.create\_arc(10, 240, 200, 320, extent=180, style=ARC)  
>>> canvas.create\_arc(10, 320, 200, 400, extent=359, style=ARC)

El resultado se muestra en la Figura 10-12 [.](ch10.xhtml#ch10fig12)

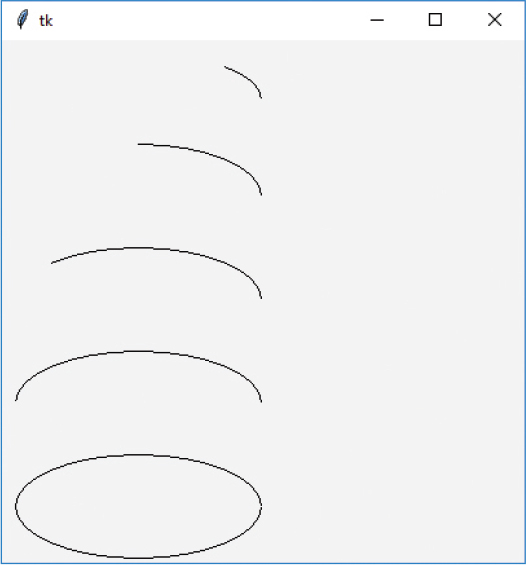


Figura 10-12*: Arcos múltiples*

**NOTA**

*Utilizamos 359 grados en el círculo final, en lugar de 360, porque tkinter considera que 360 es lo mismo que 0 grados, y no dibujaría nada.*

El parámetro style es el tipo de arco que quieres dibujar. Hay otros dos tipos de arco: cuerda y rebanada de pastel. Una *cuerda* es casi igual que el arco que ya hemos dibujado, salvo que los dos extremos se unen con una sola línea recta. Un *trozo* de tarta es exactamente lo que parece: como si cortaras un segmento de una pizza o una tarta.

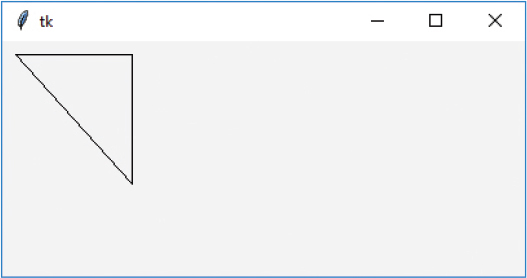
### Dibujar polígonos

Un *polígono* es cualquier forma con tres o más lados. Hay polígonos de formas regulares, como triángulos, cuadrados, rectángulos, pentágonos, hexágonos, etc., así como polígonos *irregulares* con bordes desiguales, muchos más lados y formas extrañas.

Al dibujar polígonos con tkinter , tienes que proporcionar las coordenadas de cada punto del polígono. Podemos dibujar un triángulo con el siguiente código:

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=400, height=400)  
>>> canvas.pack()  
>>> canvas.create\_polygon(10, 10, 100, 10, 100, 110, fill='',   
 outline='black')

Este ejemplo dibuja un triángulo comenzando con las *coordenadas* *x e* *y* (10, 10), luego moviéndose a través de ( 100, 10 ), y terminando en ( 100, 110 ). Establecemos el color de relleno en nada (una cadena vacía), para que no se coloree el triángulo, y el contorno se establece en ’black’ , para que se dibuje con una línea negra. Debería tener el aspecto de [la Figura 10-13 .](ch10.xhtml#ch10fig13)



*Figura 10-13: Dibujar un triángulo*

Podemos añadir un polígono irregular utilizando este código:

canvas.create\_polygon(200, 10, 240, 30, 120, 100, 140, 120, fill='',  
outline='black')

Este código comienza con las coordenadas ( 200, 10 ), se desplaza a ( 240, 30 ), luego a ( 120, 100 ), y finalmente a ( 140, 120 ). El módulo tkinter une automáticamente la línea de vuelta a la primera coordenada. El resultado se muestra en la [Figura 10-14 .](ch10.xhtml#ch10fig14)

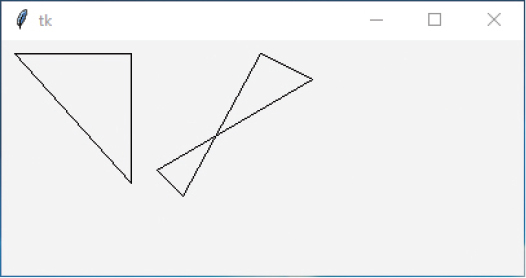


Figura 10-14*: Polígono irregular*

### Mostrar texto

Además de dibujar formas, también puedes escribir en el lienzo utilizando la función create\_text . Esta función sólo toma dos coordenadas -las *posiciones* *x e* *y* del texto- junto con un parámetro con nombre para el texto a mostrar. En el código siguiente, creamos nuestro lienzo como antes y luego mostramos una frase situada en las coordenadas ( 150, 100 ). Guarda este código como *texto.py :*

from tkinter import \*  
tk = Tk()  
canvas = Canvas(tk, width=400, height=400)  
canvas.pack()  
canvas.create\_text(150, 100, text='There once was a man from Toulouse,')

La función create\_text puede tomar otros parámetros útiles, como un color de relleno del texto. En el código siguiente, llamamos a la función create \_text con las coordenadas ( 130, 120 ), el texto que queremos mostrar y un color de relleno rojo:

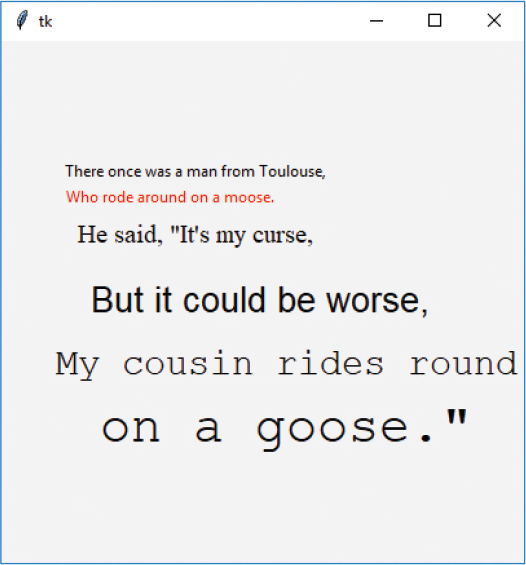
canvas.create\_text(130, 120, text='Who rode around on a moose.', fill='red')

También puedes especificar la *fuente* , o el tipo de letra utilizado para el texto mostrado, como una tupla con el nombre de la fuente y el tamaño del texto. Por ejemplo, la tupla para la fuente *Times* de tamaño 20 es ( 'Times', 20 ). En el código siguiente, mostramos el texto utilizando la fuente *Times* de tamaño 15, la fuente *Helvetica* de tamaño 20 y la fuente *Courier* de tamaños 22 y 30:



canvas.create\_text(150, 150, text='He said, "It\'s my curse,', font=('Times', 15))  
canvas.create\_text(200, 200, text='But it could be worse,', font=('Helvetica', 20))  
canvas.create\_text(220, 250, text='My cousin rides round', font=('Courier', 22))  
canvas.create\_text(220, 300, text='on a goose."', font=('Courier', 30))

[La Figura 10-15](ch10.xhtml#ch10fig15) muestra el resultado de estas funciones utilizando las tres fuentes especificadas en cinco tamaños diferentes.

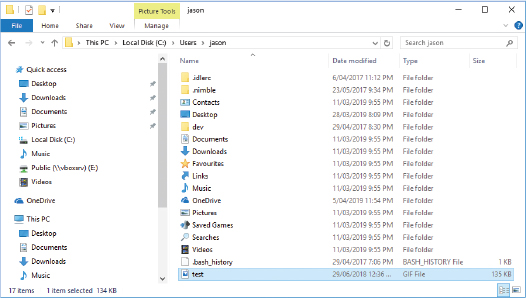


*Figura 10-15: Dibujar texto con tkinter*

### Visualizar imágenes

Para mostrar una imagen en un lienzo utilizando tkinter , primero carga la imagen y luego utiliza la función create\_image en el objeto lienzo. Cualquier imagen que cargues debe estar en una carpeta (o directorio) accesible para Python.

El mejor lugar para colocar las imágenes es en tu carpeta de inicio. En Windows es *c:* \*Users* \*<tu* nombre de usuario> ; en macOS, */Users/<tu nombre de usuario>* ; y en Ubuntu o Raspberry Pi, */home/<tu* *nombre de usuario>* . [La Figura 10-16](ch10.xhtml#ch10fig16) muestra una carpeta home en Windows.



*Figura 10-16:* Carpeta de inicio en Windows

**NOTA**

*Con tkinter , sólo puedes cargar imágenes GIF -archivos de imagen con la extensión* .gif-. *Puedes mostrar otros tipos de imágenes, como PNG* (.png) *y JPG* (.jpg), *pero tendrás que utilizar un módulo diferente como Pillow, la biblioteca de imágenes de Python (que encontrarás en* https://python-pillow.org *)* *. Si no tienes un GIF para utilizar, prueba a abrir una foto y luego guardarla como GIF. En Windows puedes hacerlo fácilmente con la aplicación Paint, pero hay muchas otras formas de convertir imágenes al formato GIF.*

Podemos visualizar una imagen llamada *prueba.gif* de la siguiente manera.

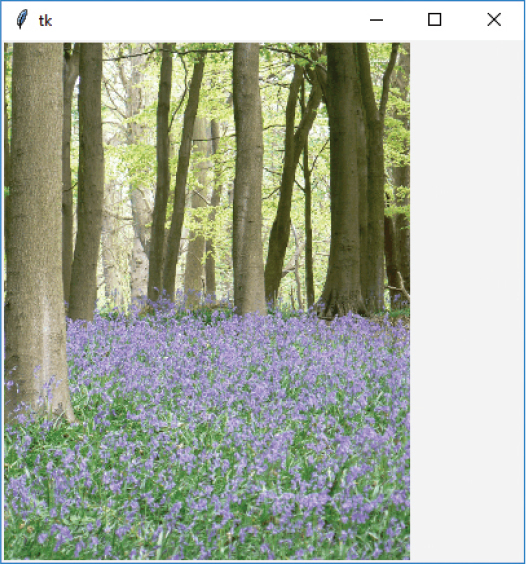
from tkinter import \*  
tk = Tk()  
canvas = Canvas(tk, width=400, height=400)  
canvas.pack()  
my\_image = PhotoImage(file='c:\\Users\\jason\\test.gif')  
canvas.create\_image(0, 0, anchor=NW, image=my\_image)

En las cuatro primeras líneas, configuramos el lienzo como en los ejemplos anteriores. En la quinta línea, la imagen se carga en la variable my\_image . Creamos PhotoImage con el nombre de archivo *c:\\* *Usuarios\jason\test.gif .* Necesitamos usar dos barras invertidas ( *\* ) en un nombre de archivo de Windows, porque la barra invertida es un carácter especial en una cadena Python (se usa para algo llamado carácter de escape -por ejemplo, *\t* es el carácter de escape que representa un tabulador, *\n* es el carácter de escape que representa una nueva línea, que ya usamos en el [Capítulo 7](ch07.xhtml#ch07) ), y dos barras invertidas son simplemente una forma de decir: "No quiero usar un carácter de escape aquí -quiero una sola barra invertida".

Si guardaste tu imagen en el escritorio, debes crear el PhotoImage con esa carpeta, así:

my\_image = PhotoImage(file='C:\\Users\\JoeSmith\\Desktop\\test.gif')

Una vez cargada la imagen en la variable, canvas .create\_image(0, 0, anchor=NW, image=my\_image) la muestra utilizando la función create\_image . Las coordenadas ( 0, 0 ) son dónde se mostrará la imagen, y anchor=NW (con NW que significa *noroeste* ) indica a la función que utilice el borde superior izquierdo de la imagen como punto de partida al dibujar; de lo contrario, utilizará por defecto el centro de la imagen como punto de partida. El último parámetro con nombre, image , señala la variable de la imagen cargada. Tu pantalla debería tener un aspecto similar al de [la Figura 10-17 .](ch10.xhtml#ch10fig17)



*Figura 10-17: Visualización de una imagen*

### Crear una animación básica

Ya hemos visto cómo crear imágenes estáticas que no se mueven. Ahora nos centraremos en la creación de animaciones.

La animación no es necesariamente una especialidad del módulo tkinter , pero puede manejar lo básico. Por ejemplo, podemos crear un triángulo relleno y hacer que se mueva por la pantalla utilizando este código:

>>> import time  
>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=400, height=200)  
>>> canvas.pack()  
>>> canvas.create\_polygon(10, 10, 10, 60, 50, 35)  
>>> for x in range(1, 61):  
 canvas.move(1, 5, 0)  
 tk.update()  
 time.sleep(0.05)

Cuando ejecutes este código, el triángulo empezará a moverse por la pantalla hasta el final de su recorrido, como en la [Figura 10-18 .](ch10.xhtml#ch10fig18)

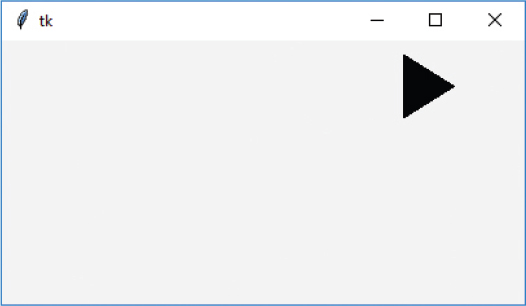


Figura 10-18*: Triángulo en* movimiento

Como antes, hemos utilizado las tres primeras líneas después de importar tkinter para realizar la configuración básica para visualizar un lienzo. Creamos el triángulo con la llamada a la función canvas.create\_polygon(10, 10, 10, 60, 50, 35) .

**NOTA**

*Cuando introduzcas esta línea, se imprimirá un número en la pantalla. Se trata de un identificador para el polígono. Podemos utilizarlo para referirnos a la forma más adelante, como se describe en el siguiente ejemplo.*

A continuación, creamos un sencillo bucle for para contar de 1 a 61, comenzando por for x in range(1, 61) .

El bloque de código dentro del bucle mueve el triángulo por la pantalla. La función canvas.move moverá cualquier objeto dibujado añadiendo valores a sus coordenadas x y y . Por ejemplo, con canvas.move(1, 5, 0) , movemos el objeto con ID 1 (el identificador del triángulo -ver la Nota anterior) 5 píxeles a lo ancho y 0 píxeles a lo bajo. Para volver a moverlo, podríamos utilizar la llamada a la función canvas.move(1, -5, 0) .



La función tk.update() obliga a tkinter a actualizar la pantalla (redibujarla). Si no utilizáramos update , tkinter esperaría a que terminara el bucle antes de mover el triángulo, lo que significa que lo verías saltar a la última posición, en lugar de moverse suavemente por el lienzo. La última línea del bucle, time.sleep(0.05) , indica a Python que espere una vigésima de segundo (0,05 segundos) antes de continuar.

Para hacer que el triángulo se mueva en diagonal por la pantalla, podemos modificar este código llamando a move(1, 5, 5) . Cierra el lienzo y crea un nuevo archivo ( **Archivo ▸ Nuevo Archivo** ) para el siguiente código:

import time  
from tkinter import \*  
tk = Tk()  
canvas = Canvas(tk, width=400, height=400)  
canvas.pack()  
canvas.create\_polygon(10, 10, 10, 60, 50, 35)  
for x in range(0, 60):  
 canvas.move(1, 5, 5)  
 tk.update()  
 time.sleep(0.05)

Este código difiere del original en dos aspectos:

* Hicimos la altura del lienzo 400, en lugar de 200, con canvas = Canvas(tk, width=400, height=400) .
* Añadimos 5 a las *coordenadas* *x* e *y* del triángulo con canvas.move(1, 5, 5) .

[La Figura 10-19](ch10.xhtml#ch10fig19) muestra la posición del triángulo al final del bucle, después de guardar el código y ejecutarlo.

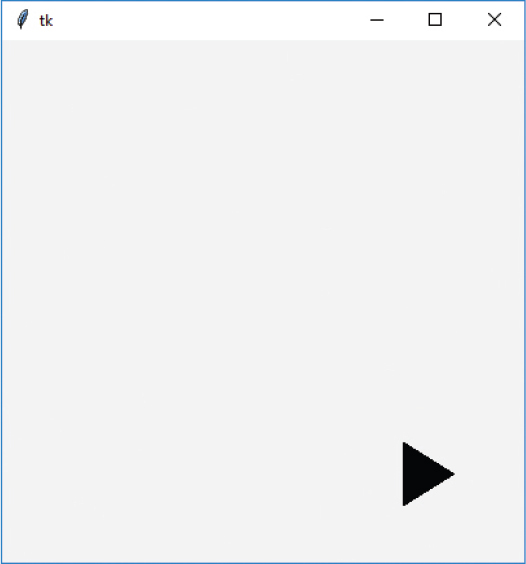


Figura 10-19*: El triángulo se ha desplazado a la parte inferior de la pantalla.*

Para mover el triángulo diagonalmente hacia arriba en la pantalla hasta su posición inicial, utiliza (-5, -5). Añade este código al final del archivo:

>>> for x in range(0, 60):  
 canvas.move(1, -5, -5)  
 tk.update()  
 time.sleep(0.05)

Después de ejecutar este código, el triángulo se moverá hacia atrás hasta donde empezó.

### Cómo hacer que un objeto reaccione ante algo

Podemos hacer que el triángulo reaccione cuando alguien pulse una tecla utilizando *eventos* *.* *Los eventos* son cosas que ocurren mientras se ejecuta un programa, como que alguien mueva el ratón, pulse una tecla o cierre una ventana. Puedes decirle a tkinter que esté atento a estos eventos y que haga algo en respuesta.

Para empezar a *manejar* eventos (hacer que Python haga algo cuando se produce un evento), primero creamos una función. La parte de la vinculación se produce cuando le decimos a tkinter que una función concreta está vinculada (o asociada) a un evento específico. En otras palabras, tkinter la llamará automáticamente para gestionar ese evento.

Por ejemplo, para hacer que el triángulo se mueva cuando pulsamos ENTER, podemos definir esta función:

def movetriangle(event):  
 canvas.move(1, 5, 0)

La función toma un único parámetro ( event ), que tkinter utiliza para enviar información a la función sobre el evento. Le decimos a tkinter que esta función debe utilizarse para un evento concreto utilizando la función bind\_all en el lienzo. El código completo tiene ahora este aspecto: introdúcelo en un nuevo archivo en IDLE y guárdalo como *moviendotriángulo.py* antes de ejecutarlo:

from tkinter import \*  
tk = Tk()  
canvas = Canvas(tk, width=400, height=400)  
canvas.pack()  
canvas.create\_polygon(10, 10, 10, 60, 50, 35)  
def movetriangle(event):  
 canvas.move(1, 5, 0)  
canvas.bind\_all('<KeyPress-Return>', movetriangle)

El primer parámetro de esta función describe el evento que queremos que tkinter vigile. En este caso, se llama <KeyPress-Return> , que es una pulsación de la tecla ENTER o RETURN. Le decimos a tkinter que la función movetriangle debe ser llamada siempre que se produzca este evento KeyPress . Ejecuta este código, haz clic en el lienzo con el ratón, y luego prueba a pulsar INTRO con el teclado.



Probemos a cambiar la dirección del triángulo en función de la pulsación de distintas teclas, como las teclas de dirección. Primero tenemos que cambiar la función movetriangle por la siguiente:

def movetriangle(event):  
 if event.keysym == 'Up':  
 canvas.move(1, 0, -3)  
 elif event.keysym == 'Down':  
 canvas.move(1, 0, 3)  
 elif event.keysym == 'Left':  
 canvas.move(1, -3, 0)  
 else:  
 canvas.move(1, 3, 0)

El objeto evento pasado a movetriangle contiene varias variables. Una de estas variables, keysym (por *símbolo de tecla* ), es una cadena que contiene el valor de la tecla real pulsada. La línea if event .keysym == ’Up’ dice que si la variable keysym contiene la cadena ’Up’ , debemos llamar a canvas.move con los parámetros (1, 0, -3), como hacemos en la línea siguiente. Si keysym contiene ’Down’ , como en elif event.keysym == ’Down’ , la llamamos con los parámetros (1, 0, 3), y así sucesivamente.

Recuerda: El primer parámetro es el número identificativo de la forma dibujada en el lienzo, el segundo es el valor a añadir a la coordenada *x (* horizontal), y el tercero es el valor a añadir a la coordenada *y (* vertical).

A continuación, le decimos a tkinter que la función movetriangle se utilizará para manejar eventos de cuatro teclas diferentes (arriba, abajo, izquierda y derecha). A continuación se muestra el aspecto que debería tener ahora el código de *movingtriangle.py* :

from tkinter import \*  
tk = Tk()  
canvas = Canvas(tk, width=400, height=400)  
canvas.pack()  
canvas.create\_polygon(10, 10, 10, 60, 50, 35)  
def movetriangle(event):  
 ➊ if event.keysym == 'Up':  
 ➋ canvas.move(1, 0, -3)  
 ➌ elif event.keysym == 'Down':  
 ➍ canvas.move(1, 0, 3)  
 elif event.keysym == 'Left':  
 canvas.move(1, -3, 0)  
 ➎ else:  
 ➏ canvas.move(1, 3, 0)  
canvas.bind\_all('<KeyPress-Up>', movetriangle)  
canvas.bind\_all('<KeyPress-Down>', movetriangle)  
canvas.bind\_all('<KeyPress-Left>', movetriangle)  
canvas.bind\_all('<KeyPress-Right>', movetriangle)

En la primera línea de la función movetriangle , comprobamos si la variable keysym contiene ’Up’ ➊ . Si es así, movemos el triángulo hacia arriba utilizando la función move con los parámetros 1 , 0 , {3 ➋ . El primer parámetro es el identificador del triángulo, el segundo es la cantidad a desplazar hacia la derecha (no queremos desplazarnos horizontalmente, por lo que el valor es 0), y el tercero es la cantidad a desplazar hacia abajo (-3 píxeles).

A continuación, comprobamos si keysym contiene ’Down’ ➌ ; si es así, movemos el triángulo hacia abajo (3 píxeles) ➍ . La última comprobación es si el valor es ’Left’ ; si es así, movemos el triángulo hacia la izquierda (-3 píxeles). Si no coincide ninguno de los valores, el último else ➎ desplaza el triángulo hacia la derecha ➏ .

Ahora el triángulo debería moverse en la dirección de la tecla de flecha pulsada.

### Más formas de utilizar el identificador

Siempre que utilizamos un create\_function del lienzo, como create\_polygon o create\_rectangle , se devuelve un identificador. Este identificador se puede utilizar con otras funciones del lienzo, como hicimos anteriormente con la función move :

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=400, height=400)  
>>> canvas.pack()  
>>> canvas.create\_polygon(10, 10, 10, 60, 50, 35)  
1  
>>> canvas.move(1, 5, 0)

El problema de este ejemplo es que create\_polygon no siempre devolverá 1. Por ejemplo, si has creado otras formas, podría devolver 2, 3 o incluso 100 (dependiendo del número de formas que hayas creado). Si cambiamos el código para almacenar el valor  devuelto como una variable, y luego utilizamos la variable (en lugar de referirnos sólo al número 1), el código funcionará independientemente del número devuelto.

>>> mytriangle = canvas.create\_polygon(10, 10, 10, 60, 50, 35)  
>>> canvas.move(mytriangle, 10, 0)

La función move nos permite mover objetos por la pantalla utilizando su identificador. Pero otras funciones del lienzo también pueden cambiar algo que hayamos dibujado. Por ejemplo, la función itemconfig puede cambiar algunos parámetros de una forma, como sus colores de relleno y contorno.

Supongamos que creamos un triángulo rojo:

>>> from tkinter import \*  
>>> tk = Tk()  
>>> canvas = Canvas(tk, width=400, height=400)  
>>> canvas.pack()  
>>> mytriangle = canvas.create\_polygon(10, 10, 10, 60, 50, 35,  
 fill='red')

Podemos cambiar el triángulo a otro color con itemconfig y utilizar el identificador como primer parámetro. El siguiente código dice: "Cambia el color de relleno del objeto identificado por el número de la variable mytriangle a azul":

>>> canvas.itemconfig(mytriangle, fill='blue')

También podríamos darle al triángulo un contorno de otro color, utilizando de nuevo el identificador como primer parámetro:

>>> canvas.itemconfig(mytriangle, outline='red')

Más adelante aprenderemos a realizar otros cambios en un dibujo, como ocultarlo y volverlo a hacer visible. La posibilidad de modificar los dibujos te resultará útil cuando empecemos a escribir juegos en el próximo capítulo.



### Lo que has aprendido

En este capítulo, has utilizado el módulo tkinter para dibujar formas geométricas simples en un lienzo, mostrar imágenes y realizar animaciones básicas. Has aprendido cómo los enlaces a eventos pueden hacer que los dibujos reaccionen cuando alguien pulsa una tecla, lo que será útil cuando empecemos a programar un juego. Aprendiste cómo las funciones create de tkinter devuelven un número identificativo, que puede utilizarse para modificar las formas una vez dibujadas, por ejemplo para moverlas por la pantalla o cambiar su color.

### Programando Puzzles

Prueba lo siguiente para seguir explorando el módulo tkinter y la animación básica. Visita http://python-for-kids.com [*para*](http://python-for-kids.com) descargar las soluciones.

#### #nº 1: Llena la pantalla de triángulos

Crea un programa utilizando tkinter para rellenar la pantalla con triángulos. A continuación, cambia el código para llenar la pantalla con triángulos de diferentes colores (rellenos).

#### #nº 2: El triángulo en movimiento

Modifica el código del triángulo en movimiento ("Crear animación básica" en [la página 159](ch10.xhtml#ch10lev1sec12) ) para que se mueva por la pantalla hacia la derecha, luego hacia abajo, después hacia la izquierda y, por último, de vuelta a su posición inicial.

#### #3: La foto en movimiento

Prueba a mostrar una foto tuya en el lienzo. Asegúrate de que sea una imagen GIF. ¿Puedes hacer que se mueva por la pantalla?

#### #4: Llena la pantalla de fotos

Coge la foto que utilizaste en el puzzle anterior y redúcela a un tamaño pequeño.

En macOS, puedes utilizar Vista Previa para cambiar el tamaño de una imagen (elige **Herramientas ▸ Ajustar tamaño** , e introduce una nueva anchura y altura. Luego, haz clic en **Archivo ▸ Exportar** para guardar con un nuevo nombre de archivo).

En Windows, puedes utilizar Paint (haz clic en el botón **Redimensionar** , elige un tamaño horizontal y vertical, y luego Archivo **▸ Guardar como** para guardar con un nuevo nombre de archivo).

En Ubuntu y Raspberry Pi, necesitarás un programa llamado GIMP (salta a la [página 203](ch13.xhtml#ch13fig02) del [Capítulo 13](ch13.xhtml#ch13) si no lo tienes instalado): selecciona **Imagen** **▸ Escalar imagen** en GIMP y, a continuación, Archivo **▸** Exportar como para guardarlo con un nuevo nombre de archivo.

Importa el módulo time y luego utiliza la función sleep (prueba con time.sleep(0.5) ) para que las fotos aparezcan más despacio.