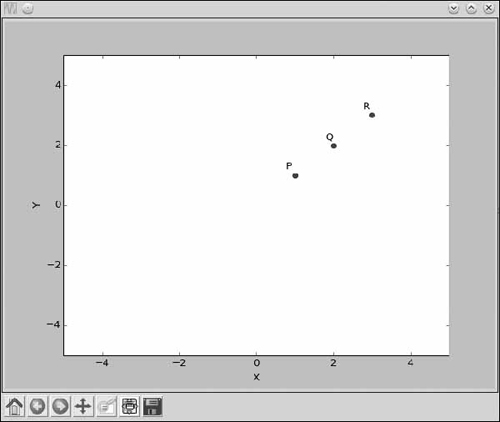
Capítulo 6: Dibujo de formas geométricas y fractales

### **Dibujar fractales**

Los fractales son patrones o formas geométricas complejas que surgen de fórmulas matemáticas sorprendentemente sencillas. Comparado con las formas geométricas, como los círculos y los rectángulos, un fractal parece irregular y sin ningún patrón o descripción obvia, pero si lo observas de cerca, verás que surgen patrones y que toda la forma está compuesta por numerosas copias de sí misma. Como los fractales implican la aplicación repetitiva de la misma *transformación geométrica* de puntos en un plano, los programas informáticos son muy adecuados para crearlos. En este capítulo aprenderemos a dibujar el helecho de Barnsley, el triángulo de Sierpiński y el conjunto de Mandelbrot (los dos últimos en los desafíos), ejemplos populares de fractales estudiados en este campo. Los fractales también abundan en la naturaleza: las costas, los árboles y los copos de nieve son ejemplos populares.

#### ***Transformaciones de puntos en un plano***

Una idea básica en la creación de fractales es la de la transformación de un punto. Dado un punto, *P(x*, *y*), en un plano *x-y*, un ejemplo de transformación es *P(x*, *y*) → *Q(x* + 1, *y* + 1), lo que significa que, tras aplicar la transformación , se crea un nuevo punto, *Q*, que está una unidad por encima y una unidad a la derecha de *P*. Si consideras entonces *Q* como punto de partida, obtendrás otro punto, *R*, que está una unidad por encima y una unidad a la derecha de *Q*. Considera que el punto de partida, *P*, es (1, 1). [La figura 6-5](ch06.html#ch6fig5) muestra el aspecto que tendrían los puntos.



*Figura 6-5: Los* puntos Q *y* R *se han obtenido aplicando una transformación al punto* P *durante dos iteraciones.*

Esta transformación es, por tanto, una regla que describe cómo se desplaza un punto en el plano *x-y*, partiendo de una posición inicial y desplazándose a un punto distinto en cada iteración. Podemos pensar en una transformación como la *trayectoria* del punto en el plano. Ahora, considera que en lugar de una regla de transformación, hay dos reglas de este tipo y que una de estas transformaciones se elige *al azar* en cada paso. Consideremos estas reglas:

Regla 1: *P* 1*(x*, *y*) → *P* 2*(x* + 1, *y* - 1)

Regla 2: *P* 1*(x*, *y*) → *P* 2 (*x* + 1, *y* + 1)

Considera que *P1*(1, 1) es el punto de partida. Si realizamos cuatro iteraciones, podríamos tener la siguiente secuencia de puntos

*P* 1 (1, 1) → *P* 2 (2, 0) (Regla 1)

*P* 2 (2, 0) → *P* 3 (3, 1) (Regla 2)

*P* 3 (3, 1) → *P* 4 (4, 2) (Regla 2)

*P* 4 (4, 2) → *P* 5 (5, 1) (Regla 1)

... y así sucesivamente.

La regla de transformación se elige al azar, y cada regla tiene la misma probabilidad de ser seleccionada. Se elija la que se elija, los puntos avanzarán hacia la derecha, porque aumentamos la *coordenada x* en ambos casos. A medida que los puntos avanzan hacia la derecha, se mueven hacia arriba o hacia abajo, creando así una trayectoria en zigzag. El siguiente programa traza la trayectoria de un punto al someterlo a una de estas transformaciones durante un número determinado de iteraciones:

'''  
Example of selecting a transformation from two equally probable  
transformations  
'''  
import matplotlib.pyplot as plt  
import random  
  
def transformation\_1(p):  
x = p[0]  
y = p[1]  
return x + 1, y - 1  
  
def transformation\_2(p):  
x = p[0]  
y = p[1]  
return x + 1, y + 1  
  
def transform(p):  
➊ # List of transformation functions  
transformations = [transformation\_1, transformation\_2]  
# Pick a random transformation function and call it  
➋ t = random.choice(transformations)  
➌ x, y = t(p)  
return x, y  
  
def build\_trajectory(p, n):  
x = [p[0]]  
y = [p[1]]  
for i in range(n):  
p = transform(p)  
x.append(p[0])  
y.append(p[1])  
  
return x, y  
  
if \_\_name\_\_ == '\_\_main\_\_':  
# Initial point  
p = (1, 1)  
n = int(input('Enter the number of iterations: '))  
➍ x, y = build\_trajectory(p, n)  
# Plot  
➎ plt.plot(x, y)  
plt.xlabel('X')  
plt.ylabel('Y')  
plt.show()

Definimos dos funciones, transformation\_1() y transformation\_2(), correspondientes a las dos transformaciones anteriores. En la función transform(), creamos una lista con estos dos nombres de función en ➊ y utilizamos la función random.choice() para elegir una de las transformaciones de la lista en ➋. Ahora que hemos elegido la transformación a aplicar, la llamamos con el punto, *P*, y almacenamos las coordenadas del punto transformado en las etiquetas x, y ➌ y las devolvemos.

**SELECCIONAR UN ELEMENTO ALEATORIO DE UNA LISTA**

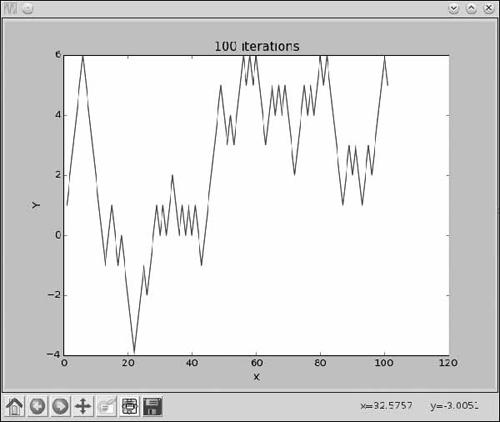
La función random.choice() que vimos en nuestro primer programa fractal se puede utilizar para seleccionar un elemento aleatorio de una lista. Cada elemento tiene *la misma* probabilidad de ser devuelto. Aquí tienes un ejemplo:

>>> import random  
>>> l = [1, 2, 3]  
>>> random.choice(l)  
3  
>>> random.choice(l)  
1  
>>> random.choice(l)  
1  
>>> random.choice(l)  
3  
>>> random.choice(l)  
3  
>>> random.choice(l)  
2

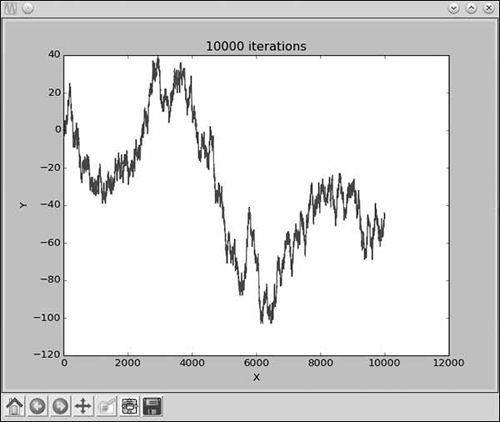
La función también funciona con tuplas y cadenas. En este último caso, devuelve un carácter aleatorio de la cadena.

Cuando ejecutas el programa, te pide el número de iteraciones, n-es decir, el número de veces que se aplicaría la transformación. A continuación, llama a la función build\_trajectory() con n y el punto inicial, *P*, que se fija en (1, 1) ➍. La función build\_trajectory() llama repetidamente a la función transform() n veces, utilizando dos listas, x y y, para almacenar la *coordenada x* y la *coordenada y* de todos los puntos transformados. Por último, devuelve las dos listas, que luego se trazan ➎.

[Las figuras 6-6](ch06.html#ch6fig6) y [6-7](ch06.html#ch6fig7) muestran la trayectoria del punto durante 100 y 10.000 iteraciones, respectivamente. El movimiento en zigzag es bastante evidente en ambas figuras. Esta trayectoria en zigzag suele denominarse *paseo aleatorio sobre una línea*.



*Figura 6-6: La trayectoria en zigzag trazada por el punto (1, 1) cuando se somete a una u otra de las dos transformaciones de forma aleatoria durante 100 iteraciones*



*Figura 6-7: La trayectoria en zigzag trazada por el punto (1, 1) cuando se somete a una u otra de las dos transformaciones aleatoriamente durante 10.000 iteraciones.*

Este ejemplo demuestra una idea básica en la creación de fractales: partir de un punto inicial y aplicar una transformación a ese punto repetidamente. A continuación, veremos un ejemplo de aplicación de las mismas ideas para dibujar el helecho de *Barnsley*.

#### ***Dibujar el helecho de Barnsley***

El matemático británico Michael Barnsley describió cómo crear estructuras parecidas a helechos utilizando aplicaciones repetidas de una transformación simple sobre un punto (ver [Figura 6](ch06.html#ch6fig8)-8).



*Figura 6-8: Helechos en forma de* [*dama2*](footnote.html#fn04)

Propuso los siguientes pasos para crear estructuras parecidas a helechos: empieza con el punto (0, 0) y selecciona *al azar* una de las siguientes transformaciones con la *probabilidad* asignada:

**Transformación 1** (probabilidad 0,85):

*xn+1* = 0,*85xn* + 0,*04yn*

*yn+1* = -0,*04yn* + 0,*85yn* + 1,6

**Transformación 2** (probabilidad 0,07):

*xn+1* = 0,*2xn* - 0,*26yn*

*yn+1* = 0,*23yn* + 0,*22yn* + 1,6

**Transformación** 3 (0,07 de probabilidad):

*xn+1* = -0,*15xn* - 0,*28xn*

*yn+1* = 0,*26yn* + 0,*24yn* + 0,44

**Transformación** 4 (probabilidad 0,01):

*xn+1* = 0

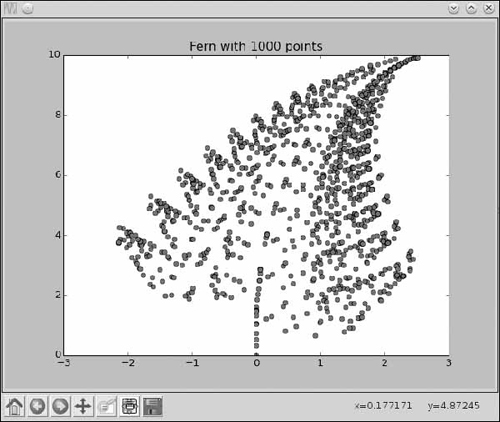
*yn+1* = 0,*16yn*

Cada una de estas transformaciones es responsable de crear una parte del helecho. La primera transformación seleccionada con la mayor probabilidad -y, por tanto, el mayor número de veces- crea el tallo y las frondas inferiores del helecho. La segunda y tercera transformaciones crean la fronda inferior de la izquierda y de la derecha, respectivamente, y la cuarta transformación crea el tallo del helecho.

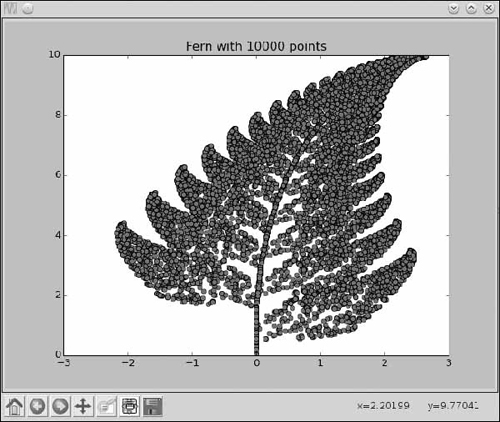
Éste es un ejemplo de selección probabilística no uniforme, sobre la que aprendimos por primera vez en [el Capítulo 5](ch05.html#ch05). El siguiente programa dibuja el helecho de Barnsley para el número de puntos especificado:

'''  
Draw a Barnsley Fern  
'''  
import random  
import matplotlib.pyplot as plt  
  
def transformation\_1(p):  
x = p[0]  
y = p[1]  
x1 = 0.85\*x + 0.04\*y  
y1 = -0.04\*x + 0.85\*y + 1.6  
return x1, y1  
  
def transformation\_2(p):  
x = p[0]  
y = p[1]  
x1 = 0.2\*x - 0.26\*y  
y1 = 0.23\*x + 0.22\*y + 1.6  
return x1, y1  
  
def transformation\_3(p):  
x = p[0]  
y = p[1]  
x1 = -0.15\*x + 0.28\*y  
y1 = 0.26\*x + 0.24\*y + 0.44  
return x1, y1  
  
def transformation\_4(p):  
x = p[0]  
y = p[1]  
x1 = 0  
y1 = 0.16\*y  
return x1, y1  
  
def get\_index(probability):  
r = random.random()  
c\_probability = 0  
sum\_probability = []  
for p in probability:  
c\_probability += p  
sum\_probability.append(c\_probability)  
for item, sp in enumerate(sum\_probability):  
if r <= sp:  
return item  
return len(probability)-1  
  
def transform(p):  
# List of transformation functions  
transformations = [transformation\_1, transformation\_2,  
transformation\_3, transformation\_4]  
➊ probability = [0.85, 0.07, 0.07, 0.01]  
# Pick a random transformation function and call it  
tindex = get\_index(probability)  
➋ t = transformations[tindex]  
x, y = t(p)  
return x, y  
  
def draw\_fern(n):  
# We start with (0, 0)  
x = [0]  
y = [0]  
  
x1, y1 = 0, 0  
for i in range(n):  
x1, y1 = transform((x1, y1))  
x.append(x1)  
y.append(y1)  
return x, y  
  
if \_\_name\_\_ == '\_\_main\_\_':  
n = int(input('Enter the number of points in the Fern: '))  
x, y = draw\_fern(n)  
# Plot the points  
plt.plot(x, y, 'o')  
plt.title('Fern with {0} points'.format(n))  
plt.show()

Cuando ejecutas este programa, te pide que especifiques el número de puntos del helecho y, a continuación, crea el helecho. Las Figuras [6-9](ch06.html#ch6fig9) y [6-10](ch06.html#ch6fig10) muestran helechos con 1.000 y 10.000 puntos, respectivamente.



*Figura 6-9: Un helecho con 1.000 puntos*



*Figura 6-10: Un helecho con 10.000 puntos*

Las cuatro reglas de transformación se definen en las funciones transformation\_1(), transformation\_2(), transformation\_3(), y transformation\_4(). La probabilidad de que se seleccione cada una se declara en una lista en ➊, y luego se selecciona una de ellas ➋ para aplicarla cada vez que la función transform() es llamada por la función draw\_fern().

El número de veces que se transforma el punto inicial (0, 0) es el mismo que el número de puntos del helecho especificados como entrada al programa.

[anterior](ch06_2.html)[Subtema 3 de 5: (Ver todo)](ch06.html)[siguiente](ch06_4.html)