### Estilizar el recorrido

En esta sección, personalizaremos nuestros trazados para resaltar las características importantes de cada paseo y restar importancia a los elementos que distraen. Para ello, identificamos las características que queremos resaltar, como dónde empezó el paseo, dónde terminó y el camino recorrido. A continuación, identificamos las características a las que queremos restar importancia, como las marcas de graduación y las etiquetas. El resultado debe ser una representación visual sencilla que comunique claramente el camino recorrido en cada paseo aleatorio.

#### Colorear los puntos

Utilizaremos un mapa de colores para mostrar el orden de los puntos en el paseo, y eliminaremos el contorno negro de cada punto para que el color de los puntos sea más claro. Para colorear los puntos según su posición en el recorrido, pasamos al argumento c una lista que contiene la posición de cada punto. Como los puntos se trazan en orden, esta lista sólo contiene los números de 0 a 4.999:

**rw\_visual.py**

--snip--  
while True:  
 # Make a random walk.  
 rw = RandomWalk()  
 rw.fill\_walk()  
  
 # Plot the points in the walk.  
 plt.style.use('classic')  
 fig, ax = plt.subplots()  
❶ point\_numbers = range(rw.num\_points)  
 ax.scatter(rw.x\_values, rw.y\_values, c=point\_numbers, cmap=plt.cm.Blues,  
 edgecolors='none', s=15)  
 ax.set\_aspect('equal')  
 plt.show()  
 --snip--

Utilizamos range() para generar una lista de números igual al número de puntos del recorrido ❶. Asignamos esta lista a point\_numbers, que utilizaremos para establecer el color de cada punto del paseo. Pasamos point\_numbers al argumento c, utilizamos el mapa de colores Blues, y luego pasamos edgecolors='none' para deshacernos del contorno negro alrededor de cada punto. El resultado es un gráfico que varía de azul claro a azul oscuro, mostrando exactamente cómo se mueve el paseo desde su punto inicial hasta su punto final. Esto se muestra en la [Figura 15](#figure15-10)-10.

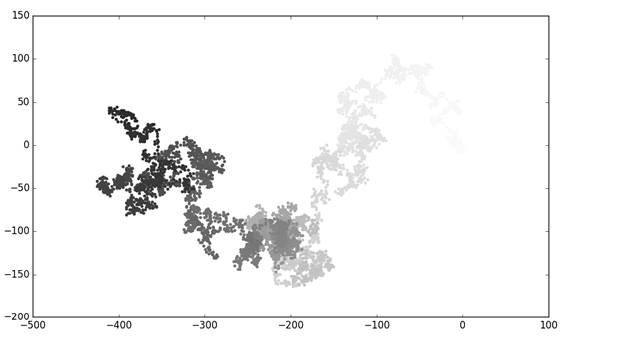


Figura15-10: Un paseo aleatorio coloreado con el mapa de colores Blues

#### Trazar los puntos inicial y final

Además de colorear los puntos para mostrar su posición a lo largo del recorrido, sería útil ver exactamente dónde empieza y acaba cada recorrido. Para ello, podemos trazar los puntos inicial y final individualmente, una vez trazada la serie principal. Haremos los puntos finales más grandes y los colorearemos de forma diferente para que destaquen:

**rw\_visual.py**

--snip--  
while True:  
 --snip--  
 ax.scatter(rw.x\_values, rw.y\_values, c=point\_numbers, cmap=plt.cm.Blues,  
 edgecolors='none', s=15)  
 ax.set\_aspect('equal')  
  
 # Emphasize the first and last points.  
 ax.scatter(0, 0, c='green', edgecolors='none', s=100)  
 ax.scatter(rw.x\_values[-1], rw.y\_values[-1], c='red', edgecolors='none',  
 s=100)  
  
 plt.show()  
 --snip--

Para mostrar el punto inicial, trazamos el punto (0, 0) en verde y en un tamaño mayor (s=100) que el resto de los puntos. Para marcar el punto final, trazamos los últimos valores *x*- y *y*- también en rojo y con un tamaño de 100. Asegúrate de insertar este código justo antes de la llamada a plt.show() para que los puntos inicial y final se dibujen encima de todos los demás puntos.

Cuando ejecutes este código, deberías ser capaz de distinguir exactamente dónde empieza y acaba cada paseo. Si estos puntos finales no destacan con suficiente claridad, ajusta su color y tamaño hasta que lo hagan.

#### Limpiar los ejes

Eliminemos los ejes de este gráfico para que no distraigan de la trayectoria de cada paseo. He aquí cómo ocultar los ejes:

**rw\_visual.py**

--snip--  
while True:  
 --snip--  
 ax.scatter(rw.x\_values[-1], rw.y\_values[-1], c='red', edgecolors='none',  
 s=100)  
  
 # Remove the axes.  
 ax.get\_xaxis().set\_visible(False)  
 ax.get\_yaxis().set\_visible(False)  
  
 plt.show()  
 --snip--

Para modificar los ejes, utilizamos los métodos ax.get\_xaxis() y ax.get\_yaxis() para obtener cada eje, y luego encadenamos el método set\_visible() para hacer invisible cada eje. A medida que sigas trabajando con visualizaciones, verás con frecuencia este encadenamiento de métodos para personalizar distintos aspectos de una visualización.

Ejecuta ahora *rw\_visual.py*; deberías ver una serie de gráficos sin ejes.

#### Añadir puntos de trazado

Aumentemos el número de puntos, para tener más datos con los que trabajar. Para ello, aumentamos el valor de num\_points cuando hagamos una instancia de RandomWalk y ajustamos el tamaño de cada punto al dibujar el gráfico:

**rw\_visual.py**

--snip--  
while True:  
 # Make a random walk.  
 rw = RandomWalk(50\_000)  
 rw.fill\_walk()  
  
 # Plot the points in the walk.  
 plt.style.use('classic')  
 fig, ax = plt.subplots()  
 point\_numbers = range(rw.num\_points)  
 ax.scatter(rw.x\_values, rw.y\_values, c=point\_numbers, cmap=plt.cm.Blues,  
 edgecolors='none', s=1)  
 --snip--

Este ejemplo crea un paseo aleatorio con 50.000 puntos y traza cada punto con el tamaño s=1. El paseo resultante es difuso y con forma de nube, como se muestra en la [Figura 15-11](#figure15-11). ¡Hemos creado una obra de arte a partir de un simple gráfico de dispersión!

Experimenta con este código para ver cuánto puedes aumentar el número de puntos de un recorrido antes de que tu sistema empiece a ralentizarse significativamente o el gráfico pierda su atractivo visual.

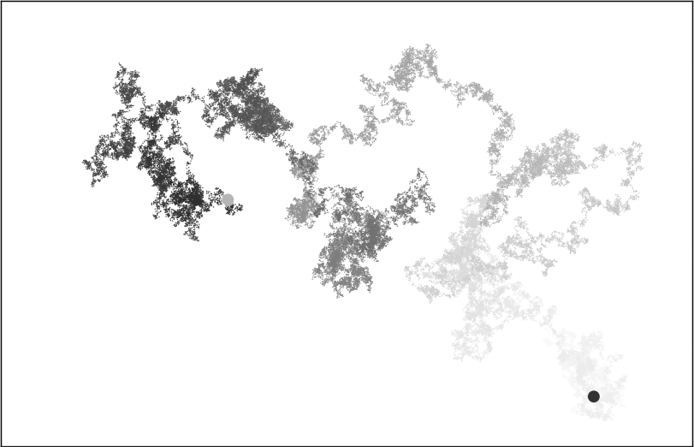


Figura 15-11: Un paseo con 50.000

puntos

#### Alterar el tamaño para llenar la pantalla

Una visualización es mucho más eficaz a la hora de comunicar patrones en los datos si se ajusta bien a la pantalla. Para que la ventana de trazado se ajuste mejor a tu pantalla, puedes ajustar el tamaño de la salida de Matplotlib. Esto se hace en la llamada subplots():

fig, ax = plt.subplots(figsize=(15, 9))

Al crear un gráfico, puedes pasar a subplots() un argumento figsize, que establece el tamaño de la figura. El parámetro figsize toma una tupla que indica a Matplotlib las dimensiones de la ventana de trazado en pulgadas.

Matplotlib asume que la resolución de tu pantalla es de 100 píxeles por pulgada; si este código no te da un tamaño de trazado exacto, ajusta los números según sea necesario. O, si conoces la resolución de tu sistema, puedes pasar a subplots() la resolución utilizando el parámetro dpi:

fig, ax = plt.subplots(figsize=(10, 6), dpi=128)

Esto debería ayudarte a hacer un uso más eficiente del espacio disponible en tu pantalla.

[anterior](c15_20.html)[Subtema 21 de 35: (Ver todo)](c15.html)[siguiente](c15_22.html)