Capítulo 2: Visualizar datos con gráficos

### **Retos de programación**

Aquí tienes algunos retos que se basan en lo que has aprendido en este capítulo. Puedes encontrar ejemplos de soluciones en [*http://www.nostarch.com/doingmathwithpython/*](http://www.nostarch.com/doingmathwithpython/).

#### ***#nº 1: ¿Cómo varía la temperatura durante el día?***

Si introduces un término de búsqueda como "el tiempo en Nueva York" en el buscador de Google, verás, entre otras cosas, un gráfico que muestra la temperatura en diferentes momentos del día actual. Tu tarea aquí es recrear ese gráfico.

Utilizando una ciudad de tu elección, encuentra la temperatura en diferentes momentos del día. Utiliza los datos para crear dos listas en tu programa y para crear un gráfico con la hora del día en el *eje x* y la temperatura correspondiente en el *eje y*. El gráfico debe indicarte cómo varía la temperatura con la hora del día. Prueba con otra ciudad y observa cómo se comparan las dos ciudades trazando ambas líneas en el mismo gráfico.

La hora del día puede indicarse mediante cadenas como '10:11 AM' o '09:21 PM'.

#### ***#nº 2: Exploración visual de una función cuadrática***

En el [capítulo 1](ch01.html#ch01), aprendiste a encontrar las raíces de una ecuación cuadrática, como x2 +*2x* + 1 = 0. Podemos convertir esta ecuación en una función escribiéndola como *y* = x2 +*2x* + 1. Para cualquier valor de *x*, la función cuadrática produce *algún* valor para *y*. Por ejemplo, cuando *x* = 1, *y* = 4. Aquí tienes un programa que calcula el valor de *y* para seis valores distintos de *x*:

'''  
Quadratic function calculator  
'''  
  
# Assume values of x  
➊ x\_values = [-1, 1, 2, 3, 4, 5]  
➋ for x in x\_values:  
# Calculate the value of the quadratic function  
y = x\*\*2 + 2\*x + 1  
print('x={0} y={1}'.format(x, y))

En ➊, creamos una lista con seis valores diferentes para x. El bucle for que comienza en ➋ calcula el valor de la función anterior para cada uno de estos valores y utiliza la etiqueta y para referirse a la lista de resultados. A continuación, imprimimos el valor de x y el valor correspondiente de y. Cuando ejecutes el programa, deberías ver la siguiente salida:

x=-1 y=0  
x=1 y=4  
x=2 y=9  
  
x=3 y=16  
x=4 y=25  
x=5 y=36

Observa que la primera línea de la salida es una raíz de la ecuación cuadrática porque es un valor de x que hace que la función sea igual a 0.

Tu reto de programación es mejorar este programa para crear una gráfica de la función. Intenta utilizar al menos 10 valores para x en lugar de los 6 anteriores. Calcula los valores correspondientes de y utilizando la función y luego crea una gráfica utilizando estos dos conjuntos de valores.

Una vez que hayas creado la gráfica, dedica algún tiempo a analizar cómo varía el valor de *y* con respecto a *x*. ¿La variación es lineal o no lineal?

#### ***#nº 3: Programa de comparación de trayectorias de proyectiles mejoradas***

Tu reto aquí es mejorar el programa de comparación de trayectorias de varias maneras. En primer lugar, tu programa debe imprimir el tiempo de vuelo, la distancia horizontal máxima y la distancia vertical máxima recorrida para cada una de las combinaciones de velocidad y ángulo de proyección.

La otra mejora consiste en hacer que el programa funcione con cualquier número de valores iniciales de velocidad y ángulo de proyección, suministrados por el usuario. Por ejemplo, así es como el programa debería pedir las entradas al usuario:

How many trajectories? 3  
Enter the initial velocity for trajectory 1 (m/s): 45  
Enter the angle of projection for trajectory 1 (degrees): 45  
Enter the initial velocity for trajectory 2 (m/s): 60  
Enter the angle of projection for trajectory 2 (degrees): 45  
Enter the initial velocity for trajectory(m/s) 3: 45  
Enter the angle of projection for trajectory(degrees) 3: 90

Tu programa también debería asegurarse de que las entradas erróneas se gestionan adecuadamente mediante un bloque try...except, igual que en el programa original.

#### ***#nº 4: Visualizar tus gastos***

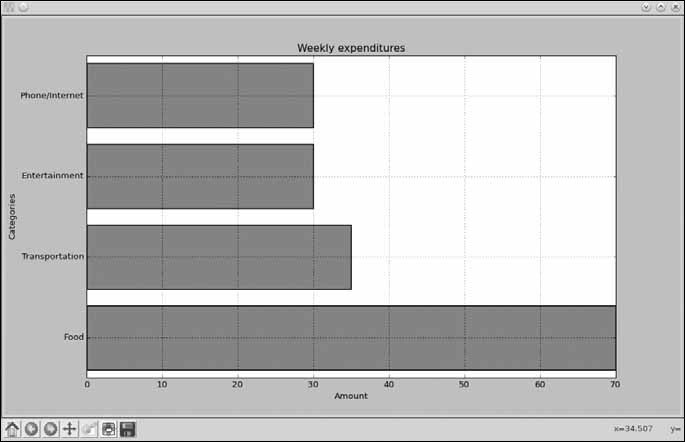
Siempre me pregunto a final de mes: "¿Adónde ha ido a parar todo ese dinero?". Estoy seguro de que no es un problema al que me enfrente sólo yo.

Para este reto, escribirás un programa que cree un gráfico de barras para comparar fácilmente los gastos semanales. El programa debe pedir primero el número de categorías para los gastos y el gasto total semanal en cada categoría, y luego debe crear el gráfico de barras que muestre estos gastos.

Aquí tienes un ejemplo de cómo debería funcionar el programa:

Enter the number of categories: 4  
Enter category: Food  
Expenditure: 70  
  
Enter category: Transportation  
Expenditure: 35  
Enter category: Entertainment  
Expenditure: 30  
Enter category: Phone/Internet  
Expenditure: 30

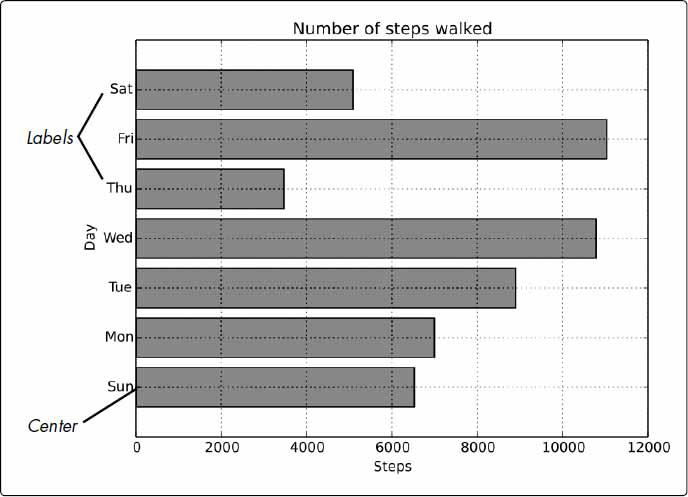
[La Figura 2-16](ch02.html#ch2fig16) muestra el gráfico de barras que se creará para comparar los gastos. Si guardas el gráfico de barras para cada semana, al final del mes podrás ver cómo han variado los gastos entre las semanas para las distintas categorías.



*Figura 2-16: Gráfico de barras que muestra los gastos por categoría durante la semana*

No hemos hablado de la creación de un gráfico de barras con matplotlib, así que probemos con un ejemplo.

Se puede crear un gráfico de barras utilizando la función barh() de matplotlib, que también está definida en el módulo pyplot. [La Figura 2-17](ch02.html#ch2fig17) muestra un gráfico de barras que ilustra el número de pasos que he dado durante la semana pasada. Los días de la semana -domingo, lunes, martes, etc.- se denominan *etiquetas*. Cada barra horizontal parte del *eje y*, y tenemos que especificar la *coordenada y* del *centro* de esta posición para cada una de las barras. La longitud de cada barra corresponde al número de pasos especificado.



*Figura 2-17: Gráfico de barras que muestra el número de pasos caminados durante una semana*

El siguiente programa crea el gráfico de barras:

'''  
Example of drawing a horizontal bar chart  
'''  
import matplotlib.pyplot as plt  
def create\_bar\_chart(data, labels):  
# Number of bars  
num\_bars = len(data)  
# This list is the point on the y-axis where each  
# Bar is centered. Here it will be [1, 2, 3...]  
➊ positions = range(1, num\_bars+1)  
➋ plt.barh(positions, data, align='center')  
# Set the label of each bar  
plt.yticks(positions, labels)  
plt.xlabel('Steps')  
plt.ylabel('Day')  
plt.title('Number of steps walked')  
# Turns on the grid which may assist in visual estimation  
plt.grid(  
plt.show()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
# Number of steps I walked during the past week  
steps = [6534, 7000, 8900, 10786, 3467, 11045, 5095]  
# Corresponding days  
labels = ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']  
create\_bar\_chart(steps, labels)

La función create\_bar\_chart() acepta dos parámetros:data, que es una lista de números que queremos representar mediante las barras y etiquetas, y la correspondiente lista labels. Hay que especificar el centro de cada barra, y yo he elegido arbitrariamente los centros 1, 2, 3, 4, etc. utilizando la ayuda de la función range() en ➊.

A continuación, llamamos a la función barh(), pasando positions y data como los dos primeros argumentos y luego el argumento de palabra clave, align='center', en ➋. El argumento de la palabra clave especifica que las barras se centren en las posiciones del *eje y* especificadas por la lista. A continuación, establecemos las etiquetas de cada barra, las etiquetas de los ejes y el título mediante la función yticks(). También llamamos a la función grid() para activar la cuadrícula, que puede ser útil para una estimación visual del número de pasos. Por último, llamamos a la función show().

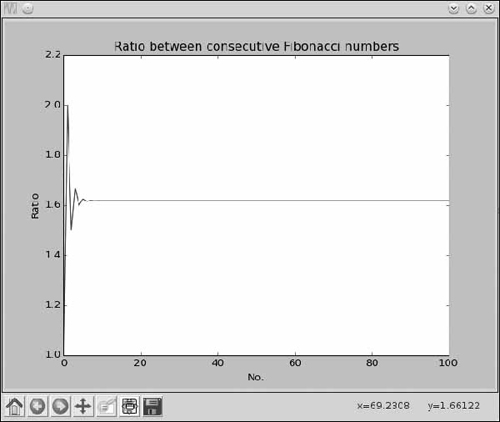
#### ***#5: Exploración de la relación entre la secuencia de Fibonacci y la proporción áurea***

La secuencia de Fibonacci (1, 1, 2, 3, 5, ...) es la serie de números en la que el *i-ésimo*número de la serie es la suma de los dos números anteriores, es decir, los números en las posiciones*(i* - 2) y*(i* - 1). Los números sucesivos de esta serie muestran una relación interesante. A medida que aumentas el número de términos de la serie, los cocientes de los pares de números consecutivos son casi iguales entre sí. Este valor se aproxima a un número especial denominado *razón* áurea. Numéricamente, la proporción áurea es el número 1,618033988 ..., y ha sido objeto de amplios estudios en música, arquitectura y naturaleza. Para este reto, escribe un programa que trace en un gráfico la relación entre números de Fibonacci consecutivos para, digamos, 100 números, lo que demostrará que los valores se aproximan a la proporción áurea.

La siguiente función, que devuelve una lista de los *n* primeros números de Fibonacci, puede resultarte útil para implementar tu solución:

def fibo(n):  
if n == 1:  
return [1]  
if n == 2:  
return [1, 1]  
# n > 2  
a = 1  
b = 1  
# First two members of the series  
series = [a, b]  
for i in range(n):  
c = a + b  
series.append(c)  
a = b  
b = c  
  
return series

La salida de tu solución debe ser un gráfico, como el que se muestra en la [Figura 2](ch02.html#ch2fig18)-18.



*Figura 2-18: La relación entre los números de Fibonacci consecutivos se aproxima a la proporción áurea.*

[anterior](ch02_6.html)[Subtema 7 de 7: (Ver todo)](ch02.html)