Capítulo 2: Visualizar datos con gráficos

### **Crear gráficos con Matplotlib**

Utilizaremos matplotlib para crear gráficos con Python. Matplotlib es un *paquete* de Python, lo que significa que es una colección de módulos con funcionalidades relacionadas. En este caso, los módulos sirven para trazar números y hacer gráficos. Matplotlib no viene incorporado con la biblioteca estándar de Python, así que tendrás que instalarlo. Las instrucciones de instalación se encuentran en [el Apéndice A](app01.html#app01). Una vez que lo tengas instalado, inicia un intérprete de comandos de Python. Como se explica en las instrucciones de instalación, puedes seguir utilizando el shell IDLE o utilizar el shell incorporado de Python.

Ahora estamos preparados para crear nuestro primer gráfico. Empezaremos con un gráfico sencillo con sólo tres puntos: (1, 2), (2, 4) y (3, 6). Para crear esta gráfica, primero haremos dos listas de números: una que almacene los valores de las coordenadas *x* de estos puntos y otra que almacene las *coordenadas y*. Las dos sentencias siguientes hacen exactamente eso, creando las dos listas x\_numbers y y\_numbers:

>>> x\_numbers = [1, 2, 3]  
>>> y\_numbers = [2, 4, 6]

A partir de aquí, podemos crear el gráfico:

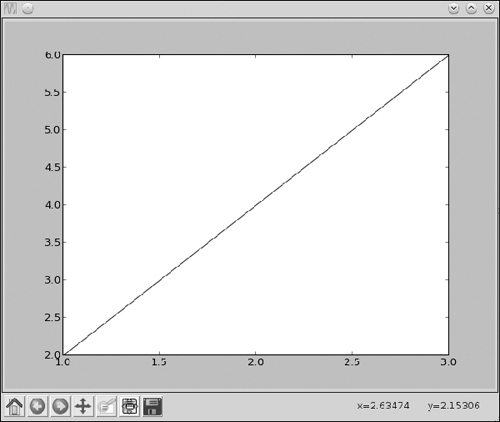
>>> from pylab import plot, show  
>>> plot(x\_numbers, y\_numbers)  
[<matplotlib.lines.Line2D object at 0x7f83ac60df10>]

En la primera línea, importamos las funciones plot() y show() del módulo pylab, que forma parte del paquete matplotlib. A continuación, llamamos a la función plot() en la segunda línea. El primer argumento de la función plot() es la lista de números que queremos representar en el eje *x*, y el segundo argumento es la lista correspondiente de números que queremos representar en el *eje y*. La función plot() devuelve un objeto o, más exactamente, una lista que contiene un objeto. Este objeto contiene la información sobre el gráfico que le hemos pedido a Python que cree. En este punto, puedes añadir más información, como un título, a la gráfica, o puedes simplemente mostrar la gráfica tal y como es. Por ahora sólo mostraremos el gráfico.

La función plot() sólo crea el gráfico. Para visualizarla realmente, tenemos que llamar a la función show():

>>> show()

Deberías ver la gráfica en una ventana de matplotlib, como se muestra en [la Figura 2-3](ch02.html#ch2fig3). (La ventana de visualización puede tener un aspecto diferente según tu sistema operativo, pero la gráfica debería ser la misma).



*Figura 2-3: Gráfica de una recta que pasa por los puntos (1, 2), (2, 4) y (3, 6)*

Observa que, en lugar de partir del origen (0, 0), el *eje x* parte del número 1 y el *eje y* parte del número 2. Estos son los números más bajos de cada uno de los ejes. Estos son los números más bajos de cada una de las dos listas. Además, puedes ver incrementos marcados en cada uno de los ejes (como 2,5, 3,0, 3,5, etc., en el *eje y*). En "[Personalizar gráficos](ch02.html#ch02lev2sec05)", en [la página 41](ch02.html#page_41), aprenderemos a controlar esos aspectos del gráfico, junto con la forma de añadir etiquetas a los ejes y un título para el gráfico.

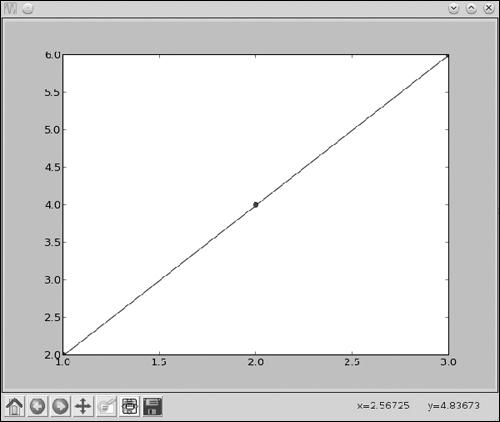
Observarás en el intérprete de comandos interactivo que no puedes introducir más sentencias hasta que cierres la ventana de matplotlib. Cierra la ventana del gráfico para poder seguir programando.

#### ***Marcar puntos en tu gráfico***

Si quieres que el gráfico marque los puntos que has suministrado para su trazado, puedes utilizar un argumento de palabra clave adicional al llamar a la función plot():

>>> plot(x\_numbers, y\_numbers, marker='o')

Introduciendo marker='o', le decimos a Python que marque cada punto de nuestras listas con un pequeño punto parecido a una *o*. Cuando vuelvas a introducir show(), verás que cada punto se marca con un punto (ver [Figura 2-4](ch02.html#ch2fig4)).

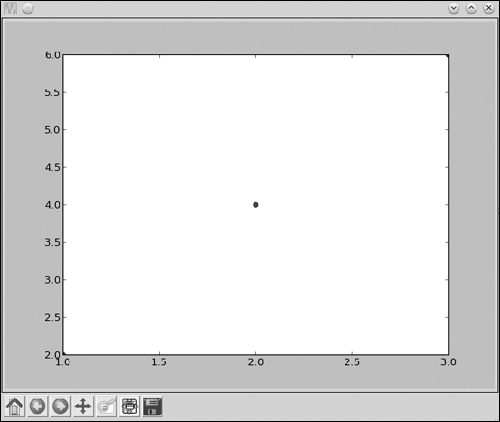


*Figura 2-4: Gráfico que muestra una recta que pasa por los puntos (1, 2), (2, 4) y (3, 6) con los puntos marcados con un punto*

El marcador en (2, 4) es fácilmente visible, mientras que los demás están ocultos en las esquinas del gráfico. Puedes elegir entre varias opciones de marker, incluyendo 'o', '\*', 'x' y '+'. El uso de marker= incluye una línea que conecta los puntos (es la opción por defecto). También puedes hacer un gráfico que marque sólo los puntos que has especificado, sin ninguna línea que los conecte, omitiendo marker=:

>>> plot(x\_numbers, y\_numbers, 'o')  
[<matplotlib.lines.Line2D object at 0x7f2549bc0bd0>]

Aquí, 'o' indica que cada punto debe marcarse con un punto, pero no debe haber ninguna línea que conecte los puntos. Llama a la función show() para mostrar el gráfico, que debería parecerse al que se muestra en la [Figura 2-5](ch02.html#ch2fig5).



*Figura 2-5: Gráfica que muestra los puntos (1, 2), (2, 4) y (3, 6)*

Como puedes ver, ahora sólo se muestran los puntos en la gráfica, sin ninguna línea que los conecte. Como en el gráfico anterior, el primer y el último punto apenas son visibles, pero pronto veremos cómo cambiar eso.

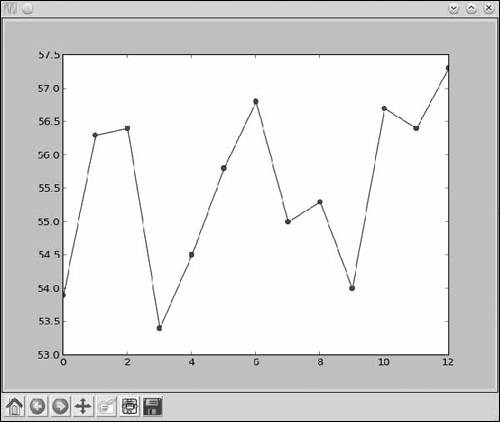
#### ***Gráfico de la temperatura media anual en Nueva York***

Echemos un vistazo a un conjunto de datos ligeramente mayor para poder explorar más funciones de matplotlib. Las temperaturas medias anuales de la ciudad de Nueva York -medidas en Central Park, concretamente- durante los años 2000 a 2012 son las siguientes: 53,9, 56,3, 56,4, 53,4, 54,5, 55,8, 56,8, 55,0, 55,3, 54,0, 56,7, 56,4 y 57,3 grados Fahrenheit. Ahora mismo, esto parece un revoltijo aleatorio de números, pero podemos trazar este conjunto de temperaturas en un gráfico para que la subida y bajada de la temperatura media de un año a otro sea mucho más clara:

>>> nyc\_temp = [53.9, 56.3, 56.4, 53.4, 54.5, 55.8, 56.8, 55.0, 55.3, 54.0, 56.7, 56.4, 57.3]  
>>> plot(nyc\_temp, marker='o')  
[<matplotlib.lines.Line2D object at 0x7f2549d52f90>]

Almacenamos las temperaturas medias en una lista, nyc\_temp. A continuación, llamamos a la función plot() pasándole sólo esta lista (y la cadena del marcador). Cuando utilizas plot() en una sola lista, esos números se trazan automáticamente en el *eje y*. Los valores correspondientes en el *eje x* se rellenan como las posiciones de cada valor de la lista. Es decir, al primer valor de temperatura, 53,9, le corresponde un valor 0 *en el eje x*, porque está en la posición 0 de la lista (recuerda que la posición de la lista empieza a contar desde 0, no desde 1). Como resultado, los números trazados en el eje *x* son los enteros de 0 a 12, que podemos considerar como correspondientes a los 13 años de los que tenemos datos de temperatura.

Introduce show() para visualizar el gráfico, que se muestra en la [Figura 2-6](ch02.html#ch2fig6). El gráfico muestra que la temperatura media ha subido y bajado de un año a otro. Si echas un vistazo a los números que hemos trazado, en realidad no están muy alejados entre sí. Sin embargo, el gráfico hace que las variaciones parezcan bastante dramáticas. Entonces, ¿qué está pasando? La razón es que matplotlib elige el rango del *eje y* de modo que sea justo el suficiente para encerrar los datos suministrados para el trazado. Así, en este gráfico, el *eje* y empieza en 53,0 y su valor más alto es 57,5. Esto hace que incluso las pequeñas diferencias parezcan aumentadas, porque el rango del eje *y* es muy pequeño. Aprenderemos a controlar el rango de cada eje en "[Personalizar gráficos](ch02.html#ch02lev2sec05)" en [la página 41](ch02.html#page_41).



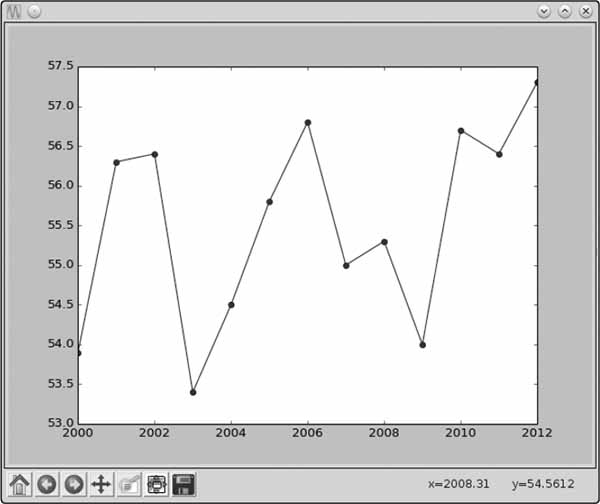
*Figura 2-6: Gráfico que muestra la temperatura media anual de la ciudad de Nueva York durante los años 2000-2012*

También puedes ver que los números del *eje y* son números de coma flotante (porque es lo que pedimos que se representara) y los del *eje x* son números enteros. Matplotlib puede manejar cualquiera de los dos.

Trazar la temperatura sin mostrar los años correspondientes es una forma rápida y sencilla de visualizar las variaciones entre los años. Sin embargo, si tuvieras pensado presentar este gráfico a alguien, te gustaría hacerlo más claro mostrando a qué año corresponde cada temperatura. Podemos hacerlo fácilmente creando otra lista con los años y llamando a la función plot():

>>> nyc\_temp = [53.9, 56.3, 56.4, 53.4, 54.5, 55.8, 56.8, 55.0, 55.3, 54.0, 56.7, 56.4, 57.3]  
>>> years = range(2000, 2013)  
>>> plot(years, nyc\_temp, marker='o')  
[<matplotlib.lines.Line2D object at 0x7f2549a616d0>]  
>>> show()

Utilizamos la función range() que conocimos en el [Capítulo](ch01.html#ch01) 1 para especificar los años 2000 a 2012. Ahora verás los años mostrados en el *eje x*(ver [Figura 2-7](ch02.html#ch2fig7)).



*Figura 2-7: Gráfico de la temperatura media anual de la ciudad de Nueva York, mostrando los* años en el *eje* x

#### ***Comparación de las tendencias mensuales de la temperatura en Nueva York***

Sin dejar de observar la ciudad de Nueva York, veamos cómo ha variado la temperatura media mensual a lo largo de los años. Esto nos dará la oportunidad de entender cómo trazar varias líneas en un mismo gráfico. Elegiremos tres años: 2000, 2006 y 2012. Para cada uno de estos años, trazaremos la temperatura media de los 12 meses.

En primer lugar, tenemos que crear tres listas para almacenar la temperatura (en grados Fahrenheit). Cada lista constará de 12 números correspondientes a la temperatura media de enero a diciembre de cada año:

>>> nyc\_temp\_2000 = [31.3, 37.3, 47.2, 51.0, 63.5, 71.3, 72.3, 72.7, 66.0, 57.0, 45.3, 31.1]  
>>> nyc\_temp\_2006 = [40.9, 35.7, 43.1, 55.7, 63.1, 71.0, 77.9, 75.8, 66.6, 56.2, 51.9, 43.6]  
>>> nyc\_temp\_2012 = [37.3, 40.9, 50.9, 54.8, 65.1, 71.0, 78.8, 76.7, 68.8, 58.0, 43.9, 41.5]

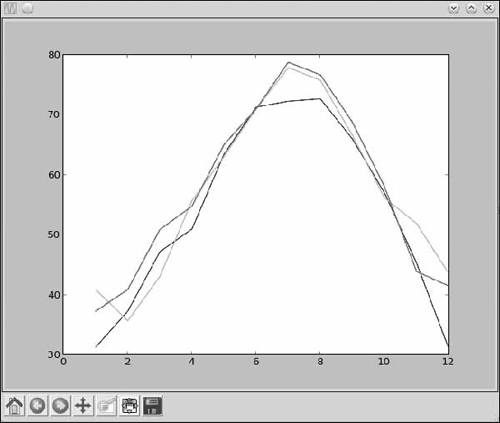
La primera lista corresponde al año 2000, y las dos siguientes a los años 2006 y 2012, respectivamente. Podríamos trazar los tres conjuntos de datos en tres gráficos diferentes, pero eso no facilitaría mucho ver cómo se compara cada año con los demás. ¡Prueba a hacerlo!

La forma más clara de comparar todas estas temperaturas es trazar los tres conjuntos de datos en un *solo* gráfico, como éste:

>>> months = range(1, 13)  
>>> plot(months, nyc\_temp\_2000, months, nyc\_temp\_2006, months, nyc\_temp\_2012)  
[<matplotlib.lines.Line2D object at 0x7f2549c1f0d0>, <matplotlib.lines.Line2D  
object at 0x7f2549a61150>, <matplotlib.lines.Line2D object at 0x7f2549c1b550>]

En primer lugar, creamos una lista (months) donde almacenamos los números 1, 2, 3, y así sucesivamente hasta el 12 utilizando la función range(). A continuación, llamamos a la función plot() con tres pares de listas. Cada par consiste en una lista de meses que se trazarán en el eje *x* y una lista de temperaturas medias mensuales (de 2000, 2006 y 2012, respectivamente) que se trazarán en el *eje y*. Hasta ahora, hemos utilizado plot() en un solo par de listas a la vez, pero en realidad puedes introducir varios pares de listas en la función plot(). Con cada lista separada por una coma, la función plot() trazará automáticamente una línea diferente para cada par.

La función plot() devuelve una lista de tres objetos en lugar de uno. Matplotlib considera que las tres curvas son distintas entre sí, y sabe que debe dibujarlas una encima de la otra cuando llamas a show(). Llamemos a show() para mostrar el gráfico, como se muestra en la [Figura 2-8](ch02.html#ch2fig8).



*Figura 2-8: Gráfico que muestra la temperatura media mensual de la ciudad de Nueva York durante los años 2000, 2006 y 2012*

Ahora tenemos tres gráficos en uno solo. Python elige automáticamente un color diferente para cada línea, para indicar que las líneas se han trazado a partir de conjuntos de datos diferentes.

En lugar de llamar a la función trazar con los tres pares a la vez, también podríamos llamar a la función trazar tres veces distintas, una para cada par:

>>> plot(months, nyc\_temp\_2000)  
[<matplotlib.lines.Line2D object at 0x7f1e51351810>]  
>>> plot(months, nyc\_temp\_2006)  
[<matplotlib.lines.Line2D object at 0x7f1e5ae8e390>]  
>>> plot(months, nyc\_temp\_2012)  
[<matplotlib.lines.Line2D object at 0x7f1e5136ccd0>]  
>>> show()

Matplotlib lleva la cuenta de los trazados que aún no se han mostrado. Así que, si esperamos a llamar a show() hasta después de llamar tres veces a plot(), todos los gráficos se mostrarán en el mismo gráfico.

Sin embargo, tenemos un problema, porque no tenemos ninguna pista sobre qué color corresponde a cada año. Para solucionarlo, podemos utilizar la función legend(), que nos permite añadir una leyenda al gráfico. Una *leyenda* es un pequeño recuadro que identifica lo que significan las distintas partes del gráfico. Aquí utilizaremos una leyenda para indicar a qué año corresponde cada línea de color. Para añadir la leyenda, llama primero a la función plot() como antes:

>>> plot(months, nyc\_temp\_2000, months, nyc\_temp\_2006, months, nyc\_temp\_2012)  
[<matplotlib.lines.Line2D object at 0x7f2549d6c410>, <matplotlib.lines.Line2D  
object at 0x7f2549d6c9d0>, <matplotlib.lines.Line2D object at 0x7f2549a86850>]

A continuación, importa la función legend() del módulo pylab y llámala como se indica a continuación:

>>> from pylab import legend  
>>> legend([2000, 2006, 2012])  
<matplotlib.legend.Legend object at 0x7f2549d79410>

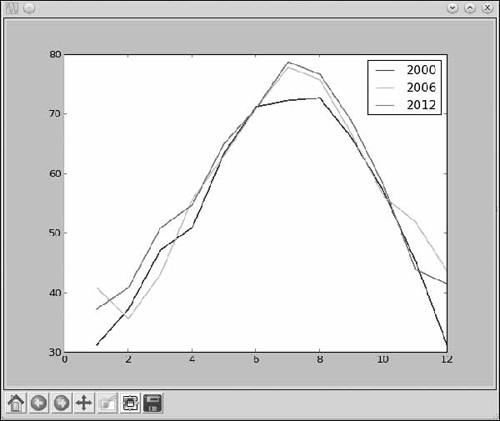
Llamamos a la función legend() con una lista de las etiquetas que queremos utilizar para identificar cada parcela del gráfico. Estas etiquetas se introducen en este orden para que coincidan con el orden de los pares de listas que se introdujeron en la función plot(). Es decir, 2000 será la etiqueta para el gráfico del primer par que introdujimos en la función plot(); 2006, para el segundo par; y 2012, para el tercero. También puedes especificar un segundo argumento a la función que especificará la posición de la leyenda. Por defecto, siempre se coloca en la parte superior derecha del gráfico. Sin embargo, puedes especificar una posición concreta, como 'lower center', 'center left', y 'upper left'. O puedes establecer la posición 'best', y la leyenda se colocará de forma que no interfiera con el gráfico.

Por último, llamamos a show() para mostrar el gráfico:

>>> show()

Como puedes ver en el gráfico (ver [Figura 2-9](ch02.html#ch2fig9)), ahora hay un cuadro de leyenda en la esquina superior derecha. Nos indica qué línea representa la temperatura media mensual del año 2000, qué línea representa el año 2006 y qué línea representa el año 2012.

Observando el gráfico, puedes concluir dos hechos interesantes: la temperatura más alta de los tres años se produjo en julio y alrededor de julio (correspondiente a 7 en el *eje x*), y ha ido aumentando desde el año 2000, con una subida más espectacular entre 2000 y 2006. Tener las tres líneas trazadas juntas en un gráfico hace que sea mucho más fácil ver este tipo de relaciones. Desde luego, es más claro que mirar unas largas listas de números o incluso que mirar tres líneas trazadas en tres gráficos distintos.



*Figura 2-9: Gráfico que muestra la temperatura media mensual de la ciudad de Nueva York, con una leyenda que muestra el año al que corresponde cada color*

#### ***Personalizar gráficos***

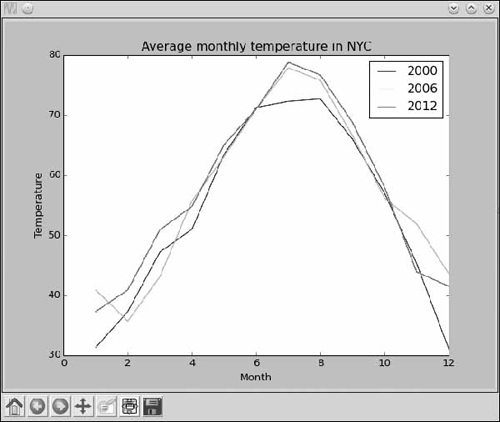
Ya hemos visto una forma de personalizar un gráfico: añadiendo una leyenda. Ahora aprenderemos otras formas de personalizar un gráfico y hacerlo más claro, añadiendo etiquetas a los ejes *x* e *y*, añadiendo un título al gráfico y controlando el rango y los pasos de los ejes.

##### **Añadir un título y etiquetas**

Podemos añadir un título a nuestro gráfico utilizando la función title() y añadir etiquetas para los *ejes* *x* e y utilizando las funciones xlabel() y ylabel(). Volvamos a crear el último gráfico y añadamos toda esta información adicional:

>>> from pylab import plot, show, title, xlabel, ylabel, legend  
>>> plot(months, nyc\_temp\_2000, months, nyc\_temp\_2006, months, nyc\_temp\_2012)  
[<matplotlib.lines.Line2D object at 0x7f2549a9e210>, <matplotlib.lines.Line2D  
object at 0x7f2549a4be90>, <matplotlib.lines.Line2D object at 0x7f2549a82090>]  
>>> title('Average monthly temperature in NYC')  
<matplotlib.text.Text object at 0x7f25499f7150>  
>>> xlabel('Month')  
<matplotlib.text.Text object at 0x7f2549d79210>  
>>> ylabel('Temperature')  
<matplotlib.text.Text object at 0x7f2549b8b2d0>  
  
>>> legend([2000, 2006, 2012])  
<matplotlib.legend.Legend object at 0x7f2549a82910>

Las tres funciones -title(), xlabel(), y ylabel()- se llaman con el texto correspondiente que queremos que aparezca en el gráfico introducido como cadenas. Al llamar a la función show() se mostrará el gráfico con toda esta información recién añadida (ver [Figura 2-10](ch02.html#ch2fig10)).



*Figura 2-10: Se han añadido al gráfico las etiquetas de los ejes y un título.*

Con los tres nuevos datos añadidos, el gráfico es más fácil de entender.

##### **Personalizar los Ejes**

Hasta ahora, hemos permitido que Python determine automáticamente los números de ambos ejes basándose en los datos suministrados a la función plot(). Esto puede estar bien para la mayoría de los casos, pero a veces este rango automático no es la forma más clara de presentar los datos, como vimos en el gráfico en el que trazamos la temperatura media anual de la ciudad de Nueva York (ver [Figura 2-7](ch02.html#ch2fig7)). Allí, incluso pequeños cambios en la temperatura parecían grandes porque el rango del eje *y* elegido automáticamente era muy estrecho. Podemos ajustar el rango de los ejes utilizando la función axis(). Esta función puede utilizarse tanto para recuperar el rango actual como para establecer un nuevo rango para los ejes.

Considera, una vez más, la temperatura media anual de la ciudad de Nueva York durante los años 2000 a 2012 y crea un gráfico como hicimos anteriormente.

>>> nyc\_temp = [53.9, 56.3, 56.4, 53.4, 54.5, 55.8, 56.8, 55.0, 55.3, 54.0, 56.7, 56.4, 57.3]  
>>> plot(nyc\_temp, marker='o')  
[<matplotlib.lines.Line2D object at 0x7f3ae5b767d0>]

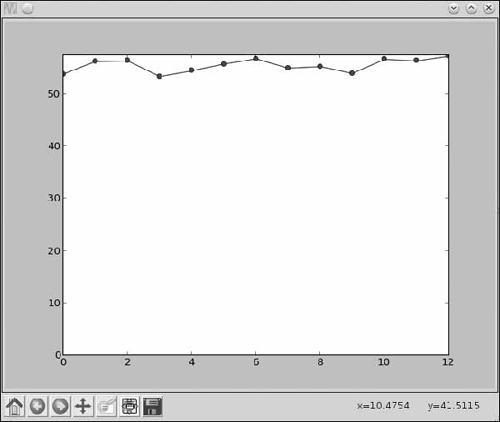
Ahora, importa la función axis() y llámala:

>>> from pylab import axis  
>>> axis()  
(0.0, 12.0, 53.0, 57.5)

La función devuelve una tupla con cuatro números correspondientes al rango para el *eje x*(0,0, 12,0) y el *eje y*(53,0, 57,5). Estos son los mismos valores de rango del gráfico que hicimos antes. Ahora, cambiemos el *eje* y para que empiece en 0 en lugar de en 53,0:

>>> axis(ymin=0)  
(0.0, 12.0, 0, 57.5)

Al llamar a la función axis() con el nuevo valor inicial del eje *y*(especificado por ymin=0) cambia el rango, y la tupla devuelta lo confirma. Si visualizas el gráfico llamando a la función show(), el *eje* y empieza en 0, y las diferencias entre los valores de los años consecutivos parecen menos drásticas (ver [Figura 2-11](ch02.html#ch2fig11)).



*Figura 2-11: Gráfico que muestra la temperatura media anual de la ciudad de Nueva York durante los años 2000-2012. El* eje y se ha*personalizado para que empiece en 0.*

Del mismo modo, puedes utilizar xmin, xmax, y ymax para establecer los valores mínimo y máximo del eje *x* y el valor máximo del *eje y*, respectivamente. Si vas a cambiar los cuatro valores, puede que te resulte más fácil llamar a la función axis() con los cuatro valores del rango introducidos como una lista, como axis([0, 10, 0, 20]). Esto establecería el rango del eje *x* en (0, 10) y el del *eje* y en (0, 20).

##### **Trazar con pyplot**

El módulo pylab es útil para crear gráficos en un intérprete de comandos interactivo, como el intérprete de comandos IDLE, como hemos hecho hasta ahora. Sin embargo, cuando se utiliza matplotlib fuera del shell IDLE -por ejemplo, como parte de un programa más amplio-, el módulo pyplot es más eficiente. No te preocupes: todos los métodos que conociste al utilizar pylab funcionarán igual con pyplot.

El siguiente programa recrea el primer gráfico de este capítulo utilizando el módulo pyplot:

'''  
Simple plot using pyplot  
'''  
  
➊ import matplotlib.pyplot  
  
➋ def create\_graph():  
x\_numbers = [1, 2, 3]  
y\_numbers = [2, 4, 6]  
  
matplotlib.pyplot.plot(x\_numbers, y\_numbers)  
matplotlib.pyplot.show()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
create\_graph()

En primer lugar, importamos el módulo pyplot utilizando la sentencia import matplotlib .pyplot ➊. Esto significa que estamos importando todo el módulo pyplot del paquete matplotlib. Para hacer referencia a cualquier definición de función o clase definida en este módulo, tendrás que utilizar la sintaxis matplotlib.pyplot.itemdonde item es la función o clase que quieres utilizar.

Esto es diferente de importar una única función o clase cada vez, que es lo que hemos estado haciendo hasta ahora. Por ejemplo, en el primer capítulo importamos la clase Fraction como from fractions import Fraction. Importar un módulo entero es útil cuando vas a utilizar varias funciones de ese módulo. En lugar de importarlas individualmente, puedes importar todo el módulo de una vez y hacer referencia a las distintas funciones cuando las necesites.

En la función create\_graph() de ➋, creamos las dos listas de números que queremos trazar en el gráfico y luego pasamos las dos listas a la función plot(), del mismo modo que hicimos antes con pylab. Esta vez, sin embargo, llamamos a la función como matplotlib.pyplot.plot(), lo que significa que estamos llamando a la función plot() definida en el módulo pyplot del paquete matplotlib. A continuación, llamamos a la función show() para visualizar el gráfico. La única diferencia entre la forma de trazar los números aquí comparada con lo que hicimos antes es el mecanismo de llamada a las funciones.

Para ahorrarnos algo de tecleo, podemos importar el módulo pyplot introduciendo import matplotlib.pyplot as plt. Entonces, podemos referirnos a pyplot con la etiqueta plt en nuestros programas, en lugar de tener que escribir siempre matplotlib.pyplot:

'''  
Simple plot using pyplot  
'''  
import matplotlib.pyplot as plt  
  
def create\_graph():  
x\_numbers = [1, 2, 3]  
y\_numbers = [2, 4, 6]  
plt.plot(x\_numbers, y\_numbers)  
plt.show()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
create\_graph()

Ahora podemos llamar a las funciones anteponiéndoles el prefijo plt en lugar de matplotlib.pyplot.

A partir de ahora, para el resto de este capítulo y de este libro, utilizaremos pylab en el intérprete de comandos interactivo y pyplot en los demás casos.

#### ***Guardar los gráficos***

Si necesitas guardar tus gráficos, puedes hacerlo utilizando la función savefig(). Esta función guarda el gráfico como un archivo de imagen, que puedes utilizar en informes o presentaciones. Puedes elegir entre varios formatos de imagen, como PNG, PDF y SVG.

Aquí tienes un ejemplo:

>>> from pylab import plot, savefig  
>>> x = [1, 2, 3]  
>>> y = [2, 4, 6]  
>>> plot(x, y)  
>>> savefig('mygraph.png')

Este programa guardará el gráfico en un archivo de imagen, *mygraph.png*, en tu directorio actual. En Microsoft Windows, suele ser *C:\Python33* (donde instalaste Python). En Linux, el directorio actual suele ser tu directorio personal *(* */home/* *)*, donde es el usuario con el que has iniciado sesión. En Mac, IDLE guarda los archivos en *~/Documentos* por defecto. Si quieres guardarlo en un directorio distinto, especifica la ruta completa. Por ejemplo, para guardar la imagen en *C:\Nen* Windows como *mygraph.png*, llamarías a la función savefig() como se indica a continuación:

>>> savefig('C:\mygraph.png')

Si abres la imagen en un programa de visualización de imágenes, verás el mismo gráfico que verías llamando a la función show(). (Verás que el archivo de imagen sólo contiene el gráfico, no toda la ventana que aparece con la función show() ). Para especificar un formato de imagen diferente, simplemente nombra el archivo con la extensión adecuada. Por ejemplo, mygraph.svg creará un archivo de imagen SVG.

Otra forma de guardar una figura es utilizar el botón Guardar de la ventana que aparece cuando llamas a show().

[anterior](ch02_3.html)[Subtema 4 de 7: (Ver todo)](ch02.html)[siguiente](ch02_5.html)