Capítulo 1: Trabajar con números

### **Retos de programación**

Aquí tienes algunos retos que te darán la oportunidad de practicar los conceptos de este capítulo. Cada problema puede resolverse de múltiples maneras, pero puedes encontrar ejemplos de soluciones en [*http://www.nostarch.com/doingmathwithpython/.*](http://www.nostarch.com/doingmathwithpython/)

#### ***#nº 1: Máquina expendedora par-impar***

Intenta escribir una "máquina expendedora par-impar", que tomará un número como entrada y hará dos cosas:

1. 1. Imprimir si el número es par o impar.

2. 2. Mostrar el número seguido de los 9 números pares o impares siguientes.

Si la entrada es 2, el programa debe imprimir even y después 2, 4, 6, 8, 10, 12, 14, 16, 18, 20. Del mismo modo, si la entrada es 1, el programa debe imprimir odd y después 1, 3, 5, 7, 9, 11, 13, 15, 17, 19.

Tu programa debe utilizar el método is\_integer() para mostrar un mensaje de error si la entrada es un número con cifras significativas más allá del punto decimal.

#### ***#2: Generador de tablas de multiplicar mejorado***

Nuestro generador de tablas de multiplicar es genial, pero sólo imprime los 10 primeros múltiplos. Mejora el generador para que el usuario pueda especificar tanto el número como hasta *qué* múltiplo. Por ejemplo, debería poder introducir que quiero ver una tabla con los 15 primeros múltiplos de 9.

#### ***#3: Conversor de unidades mejorado***

El programa de conversión de unidades que escribimos en este capítulo se limita a conversiones entre kilómetros y millas. Prueba a ampliar el programa para convertir entre unidades de masa (como kilogramos y libras) y entre unidades de temperatura (como Celsius y Fahrenheit).

#### ***#4: Calculadora de fracciones***

Escribe una calculadora que pueda realizar las operaciones matemáticas básicas con dos fracciones. Debe pedir al usuario dos fracciones y la operación que quiere realizar. Como adelanto, aquí tienes cómo puedes escribir el programa sólo con la operación de suma:

'''  
Fraction operations  
'''  
from fractions import Fraction  
  
def add(a, b):  
print('Result of Addition: {0}'.format(a+b))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
➊ a = Fraction(input('Enter first fraction: '))  
➋ b = Fraction(input('Enter second fraction: '))  
op = input('Operation to perform - Add, Subtract, Divide, Multiply: ')  
if op == 'Add':  
add(a,b)

Ya has visto la mayoría de los elementos de este programa. En ➊ y ➋, pedimos al usuario que introduzca las dos fracciones. A continuación, preguntamos al usuario qué operación debe realizar con las dos fracciones. Si el usuario introduce 'Add' como entrada, llamamos a la función add(), que hemos definido para hallar la suma de las dos fracciones pasadas como argumentos. La función add() realiza la operación e imprime el resultado. Por ejemplo:

Enter first fraction: 3/4  
Enter second fraction: 1/4  
Operation to perform - Add, Subtract, Divide, Multiply: Add  
Result of Addition: 1

Prueba a añadir soporte para otras operaciones como la resta, la división y la multiplicación. Por ejemplo, así es como tu programa debería poder calcular la diferencia de dos fracciones:

Enter first fraction: 3/4  
Enter second fraction: 1/4  
Operation to perform - Add, Subtract, Divide, Multiply: Subtract  
Result of Subtraction: 2/4

En el caso de la división, debes permitir que el usuario sepa si la primera fracción está dividida por la segunda fracción o viceversa.

#### ***#nº 5: Dar poder de salida al usuario***

Todos los programas que hemos escrito hasta ahora funcionan sólo para una iteración de entrada y salida. Por ejemplo, considera el programa para imprimir la tabla de multiplicar: el usuario ejecuta el programa e introduce un número; entonces el programa imprime la tabla de multiplicar y sale. Si el usuario quisiera imprimir la tabla de multiplicar de otro número, tendría que volver a ejecutar el programa.

Sería más cómodo que el usuario pudiera elegir entre salir o seguir utilizando el programa. La clave para escribir este tipo de programas es configurar un *bucle infinito*, o un bucle que no salga a menos que se le pida explícitamente que lo haga. A continuación, puedes ver un ejemplo de la disposición de un programa de este tipo:

'''  
Run until exit layout  
'''  
  
def fun():  
print('I am in an endless loop')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
➊ while True:  
fun()  
➋ answer = input('Do you want to exit? (y) for yes ')  
if answer == 'y':  
break

Definimos un bucle infinito utilizando while True en ➊. Un bucle while continúa ejecutándose a menos que la condición se evalúe como False. Como hemos elegido que la condición del bucle sea el valor constante True, seguirá ejecutándose eternamente a menos que lo interrumpamos de algún modo. Dentro del bucle, llamamos a la función fun(), que imprime la cadena I am in an endless loop. En ➋, se pregunta al usuario "¿Quieres salir?". Si el usuario introduce y como entrada, el programa sale del bucle utilizando la sentencia break (break sale del bucle más interno sin ejecutar ninguna otra sentencia en ese bucle). Si el usuario introduce cualquier otra entrada (o ninguna, sólo pulsa ENTER), el bucle while continúa la ejecución de , es decir, imprime de nuevo la cadena y continúa haciéndolo hasta que el usuario desee salir. Aquí tienes un ejemplo de ejecución del programa:

I am in an endless loop  
Do you want to exit? (y) for yes n  
I am in an endless loop  
Do you want to exit? (y) for yes n  
I am in an endless loop  
Do you want to exit? (y) for yes n  
I am in an endless loop  
Do you want to exit? (y) for yes y

Basándonos en este ejemplo, vamos a reescribir el generador de tablas de multiplicar para que siga ejecutándose hasta que el usuario desee salir. La nueva versión del programa se muestra a continuación:

'''  
Multiplication table printer with  
exit power to the user  
'''  
  
def multi\_table(a):  
  
for i in range(1, 11):  
print('{0} x {1} = {2}'.format(a, i, a\*i))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
  
while True:  
a = input('Enter a number: ')  
multi\_table(float(a))  
  
answer = input('Do you want to exit? (y) for yes ')  
if answer == 'y':  
break

Si comparas este programa con el que escribimos anteriormente, verás que el único cambio es la adición del bucle while, que incluye el indicador que pide al usuario que introduzca un número y la llamada a la función multi\_table().

Cuando ejecutes el programa, te pedirá un número e imprimirá su tabla de multiplicar, como antes. Sin embargo, posteriormente también preguntará si el usuario desea salir del programa. Si el usuario no desea salir, el programa estará listo para imprimir la tabla para otro número. Aquí tienes un ejemplo de ejecución:

Enter a number: 2  
2.000000 x 1.000000 = 2.000000  
2.000000 x 2.000000 = 4.000000  
2.000000 x 3.000000 = 6.000000  
2.000000 x 4.000000 = 8.000000  
  
2.000000 x 5.000000 = 10.000000  
2.000000 x 6.000000 = 12.000000  
2.000000 x 7.000000 = 14.000000  
2.000000 x 8.000000 = 16.000000  
2.000000 x 9.000000 = 18.000000  
2.000000 x 10.000000 = 20.000000  
Do you want to exit? (y) for yes n  
Enter a number:

Intenta reescribir algunos de los otros programas de este capítulo para que continúen ejecutándose hasta que el usuario les pida que salgan.

[anterior](ch01_7.html)[Subtema 8 de 8: (Ver todo)](ch01.html)