Capítulo 1: Trabajar con números

### **Obtener la entrada del usuario**

Cuando empecemos a escribir programas, nos será útil disponer de una forma agradable y sencilla de aceptar la entrada del usuario mediante la función input(). De esta forma, podemos escribir programas que pidan al usuario que introduzca un número, realice operaciones específicas con ese número y, a continuación, muestre los resultados de las operaciones. Veámoslo en acción:

➊ >>> a = input()  
➋ 1

En ➊, llamamos a la función input(), que espera a que escribas algo, como se muestra en ➋, y pulses INTRO. La información introducida se almacena en a:

>>> a  
➌ '1'

Observa las comillas simples alrededor del 1 en ➌. La función input() devuelve la entrada como una *cadena*. En Python, una cadena es cualquier conjunto de caracteres entre dos comillas. Cuando quieras crear una cadena, puedes utilizar comillas simples o dobles:

>>> s1 = 'a string'  
>>> s2 = "a string"

Aquí, tanto s1 como s2 se refieren a la misma cadena.

Aunque los únicos caracteres de una cadena sean números, Python no tratará esa cadena como un número a menos que nos deshagamos de esas comillas. Así que antes de poder realizar cualquier operación matemática con la entrada, tendremos que convertirla al tipo de número correcto. Una cadena puede convertirse en un número entero o de coma flotante utilizando la función int() o float(), respectivamente:

>>> a = '1'  
>>> int(a) + 1  
2  
>>> float(a) + 1  
2.0

Son las mismas funciones int() y float() que vimos antes, pero esta vez en lugar de convertir la entrada de un tipo de número a otro, toman una cadena como entrada ('1') y devuelven un número (2 o 2.0). Sin embargo, es importante tener en cuenta que la función int() no puede convertir una cadena que contenga un decimal de coma flotante en un número entero. Si tomas una cadena que contenga un número de coma flotante (como '2.5' o incluso '2.0') e introduces esa cadena en la función int(), obtendrás un mensaje de error:

>>> int('2.0')  
Traceback (most recent call last):  
  
File "<pyshell#26>", line 1, in  
int('2.0')  
ValueError: invalid literal for int() with base 10: '2.0'

Éste es un ejemplo de *excepción:*la forma que tiene *Python*de decirte que no puede seguir ejecutando tu programa debido a un error. En este caso, la excepción es del tipo ValueError. (Para un rápido repaso de las excepciones, consulta [el Apéndice B](app02.html#app02).)

Del mismo modo, cuando proporcionas un número fraccionario como 3/4 como entrada, Python no puede convertirlo en un número de coma flotante o entero equivalente. Una vez más, se lanza una excepción ValueError:

>>> a = float(input())  
3/4  
Traceback (most recent call last):  
File "<pyshell#25>", line 1, in  
a=float(input())  
ValueError: could not convert string to float: '3/4'

Puede resultarte útil realizar la conversión en un bloque try...except para poder *manejar* esta excepción y alertar al usuario de que el programa se ha encontrado con una entrada no válida. A continuación veremos los bloques try...except.

#### ***Manejar excepciones y entradas no válidas***

Si no estás familiarizado con try...except, la idea básica es la siguiente: si ejecutas una o más sentencias en un bloque try...except y se produce un error durante la ejecución, tu programa no se bloqueará e imprimirá un Traceback. En su lugar, la ejecución se transfiere al bloque except, donde puedes realizar una operación apropiada, por ejemplo, imprimir un mensaje de error útil o intentar otra cosa.

Así es como realizarías la conversión anterior en un bloque try...except e imprimirías un mensaje de error útil si la entrada no es válida:

>>> try:  
a = float(input('Enter a number: '))  
except ValueError:  
print('You entered an invalid number')

Ten en cuenta que tenemos que especificar el tipo de excepción que queremos manejar. Aquí, queremos manejar la excepción ValueError, así que la especificamos como except ValueError.

Ahora, cuando des una entrada no válida, como 3/4, imprime un mensaje de error útil, como se muestra en ➊:

Enter a number: 3/4  
➊ You entered an invalid number

También puedes especificar un aviso con la función input() para indicar al usuario qué tipo de entrada se espera. Por ejemplo:

>>> a = input('Input an integer: ')

El usuario verá ahora el mensaje indicándole que introduzca un número entero como entrada:

Input an integer: 1

En muchos programas de este libro, pediremos al usuario que introduzca un número como entrada, por lo que tendremos que asegurarnos de que nos ocupamos de la conversión antes de intentar realizar cualquier operación con estos números. Puedes combinar la entrada y la conversión en una sola sentencia, como se indica a continuación:

>>> a = int(input())  
1  
>>> a + 1  
2

Esto funciona muy bien si el usuario introduce un número entero. Pero como hemos visto antes, si la entrada es un número en coma flotante (incluso uno equivalente a un entero, como 1,0), se producirá un error:

>>> a = int(input())  
1.0  
Traceback (most recent call last):  
File "<pyshell#42>", line 1, in  
a=int(input())  
ValueError: invalid literal for int() with base 10: '1.0'

Para evitar este error, podríamos configurar una captura ValueError como la que vimos antes para las fracciones. De este modo, el programa detectaría los números en coma flotante, lo que no funcionaría en un programa pensado para enteros. Sin embargo, también detectaría números como 1,0 y 2,0, que Python *ve* como números de coma flotante pero que son equivalentes a los enteros y funcionarían perfectamente si se introdujeran como el tipo Python correcto.

Para evitar todo esto, utilizaremos el método is\_integer() para filtrar cualquier número con un dígito significativo después del punto decimal. (Este método sólo está definido para números de tipo float en Python; no funcionará con números que ya estén introducidos en forma de entero).

He aquí un ejemplo:

>>> 1.1.is\_integer()  
False

Aquí, llamamos al método is\_integer() para comprobar si 1,1 es un número entero, y el resultado es False porque 1,1 es realmente un número en coma flotante. Por otro lado , cuando se llama al método con 1,0 como número de coma flotante, el resultado es True:

>>> 1.0.is\_integer()  
True

Podemos utilizar is\_integer() para filtrar entradas que no sean números enteros y mantener entradas como 1,0, que se expresa como un número de coma flotante pero equivale a un número entero. Veremos cómo encajaría el método en un programa más grande un poco más adelante.

#### ***Fracciones y números complejos como entrada***

La clase Fraction que hemos conocido antes también es capaz de convertir una cadena como '3/4' en un objeto Fraction. De hecho, así es como podemos aceptar una fracción como entrada:

>>> a = Fraction(input('Enter a fraction: '))  
Enter a fraction: 3/4  
>>> a  
Fraction(3, 4)

Intenta introducir una fracción como 3/0 como entrada:

>>> a = Fraction(input('Enter a fraction: '))  
Enter a fraction: 3/0  
Traceback (most recent call last):  
File "<pyshell#2>", line 1, in  
a = Fraction(input('Enter a fraction: '))  
File "/usr/lib64/python3.3/fractions.py", line 167, in \_\_new\_\_  
raise ZeroDivisionError('Fraction(%s, 0)' % numerator)  
ZeroDivisionError: Fraction(3, 0)

El mensaje de excepción ZeroDivisionError te dice (como ya sabes) que una fracción con denominador 0 no es válida. Si piensas hacer que los usuarios introduzcan fracciones en uno de tus programas, es una buena idea atrapar siempre este tipo de excepciones. He aquí cómo puedes hacer algo así:

>>> try:  
a = Fraction(input('Enter a fraction: '))  
except ZeroDivisionError:  
print('Invalid fraction')  
  
Enter a fraction: 3/0  
Invalid fraction

Ahora, cada vez que el usuario de tu programa introduzca una fracción con 0 en el denominador, imprimirá el mensaje Invalid fraction.

Del mismo modo, la función complex() puede convertir una cadena como '2+3j' en un número complejo:

>>> z = complex(input('Enter a complex number: '))  
Enter a complex number: 2+3j  
>>> z  
(2+3j)

Si introduces la cadena como '2 + 3j' (con espacios), aparecerá un mensaje de error ValueError:

>>> z = complex(input('Enter a complex number: '))  
Enter a complex number: 2 + 3j  
Traceback (most recent call last):  
File "<pyshell#43>", line 1, in  
z = complex(input('Enter a complex number: '))  
ValueError: complex() arg is a malformed string

Es una buena idea capturar la excepción ValueError al convertir una cadena en un número complejo, como hemos hecho para otros tipos de números.

[anterior](ch01_4.html)[Subtema 5 de 8: (Ver todo)](ch01.html)[siguiente](ch01_6.html)