## 3.3. Entrada/salida

Los programas que hemos visto en la sección anterior adolecen de un serio inconveniente: cada vez que quieras obtener resultados para unos datos diferentes deberás editar el fichero de texto que contiene el programa.

Por ejemplo, el siguiente programa calcula el volumen de una esfera a partir de su radio, que es de un metro:

volumen_esfera_8.py — volumen_esfera.py

```
1 from math import pi
2
3 radio = 1
4 volumen = 4.0 / 3.0 * pi * radio ** 3
5
6 print volumen
```

Aquí tienes el resultado de ejecutar el programa:

```
$ python volumen_esfera.py↲
4.18879020479
```

Si deseas calcular ahora el volumen de una esfera de 3 metros de radio, debes editar el fichero que contiene el programa, yendo a la tercera línea y cambiándola para que el programa pase a ser éste:

### *Ejecución implícita del intérprete*

No es necesario llamar explícitamente al intérprete de Python para ejecutar los programas. En Unix existe un convenio que permite llamar al intérprete automáticamente: si la primera línea del fichero de texto empieza con los caracteres **#!**, se asume que, a continuación, aparece la ruta en la que encontrar el intérprete que deseamos utilizar para ejecutar el fichero.

Si, por ejemplo, nuestro intérprete Python está en **/usr/local/bin/python**, el siguiente fichero:

miprograma_5.py — miprograma.py

```
#! /usr/local/bin/python

from math import pi

radio = 1
perimetro = 2 * pi * radio

print perimetro
```

además de contener el programa, permitiría invocar automáticamente al intérprete de Python. O casi. Nos faltaría un último paso: dar *permiso de ejecución* al fichero. Si deseas dar permiso de ejecución has de utilizar la orden Unix **chmod**. Por ejemplo,

```
$ chmod u+x miprograma.py ↲
```

da permiso de ejecución al usuario propietario del fichero. A partir de ahora, para ejecutar el programa sólo tendremos que escribir el nombre del fichero:

```
$ miprograma.py ↲
6.28318530718
```

Si quieres practicar, genera ficheros ejecutables para los programas de los últimos tres ejercicios.

Ten en cuenta que, a veces, este procedimiento falla. En diferentes sistemas puede que Python esté instalado en directorios diferentes. Puede resultar más práctico sustituir la primera línea por esta otra:

miprograma_6.py — miprograma.py

```
#! /usr/bin/env python

from math import pi

radio = 1
perimetro = 2 * pi * radio

print perimetro
```

El programa **env** (que *debería* estar en /**usr**/**bin** en cualquier sistema) se encarga de «buscar» al programa **python**.

volumen_esfera_9.py — volumen_esfera.py

```
from math import pi

radio = 3
volumen = 4.0 / 3.0 * pi * radio ** 3

print volumen
```

Ahora podemos ejecutar el programa:

```
$ python volumen_esfera.py↲
113.097335529
```

Y si ahora quieres calcular el volumen para otro radio, vuelta a empezar: abre el fichero con el editor de texto, ve a la tercera línea, modifica el valor del radio y guarda el fichero. No es el colmo de la comodidad.

### 3.3.1. Lectura de datos de teclado

Vamos a aprender a hacer que nuestro programa, cuando se ejecute, pida el valor del radio para el que vamos a efectuar los cálculos *sin necesidad de editar el fichero de programa*.

Hay una función predefinida, *raw_input* (en inglés significa «entrada en bruto»), que hace lo siguiente: detiene la ejecución del programa y espera a que el usuario escriba un texto (el valor del radio, por ejemplo) y pulse la tecla de retorno de carro; en ese momento prosigue la ejecución y la función devuelve *una cadena* con el texto que tecleó el usuario.

Si deseas que el radio sea un valor flotante, debes transformar la cadena devuelta por *raw_input* en un dato de tipo flotante llamando a la función *float*. La función *float* recibirá como argumento la cadena que devuelve *raw_input* y proporcionará un número en coma flotante. (Recuerda, para cuando lo necesites, que existe otra función de conversión, *int*, que devuelve un entero en lugar de un flotante.) Por otra parte, *raw_input* es una función y, por tanto, el uso de los paréntesis que siguen a su nombre es obligatorio, incluso cuando no tenga argumentos.

He aquí el nuevo programa:

volumen_esfera_10.py — volumen_esfera.py

```
1 from math import pi
2
3 texto_leido = raw_input()
4 radio = float(texto_leido)
5 volumen = 4.0 / 3.0 * pi * radio ** 3
6
7 print volumen
```

Esta otra versión es más breve:

volumen_esfera_11.py — volumen_esfera.py

```
1 from math import pi
2
3 radio = float(raw_input())
4 volumen = 4.0 / 3.0 * pi * radio ** 3
5
6 print volumen
```

Al ejecutar el programa desde la línea de órdenes Unix, el ordenador parece quedar bloqueado. No lo está: en realidad Python está solicitando una entrada de teclado y espera que se la proporcione el usuario. Si tecleas, por ejemplo, el número 3 y pulsas la tecla de retorno de carro, Python responde imprimiendo en pantalla el valor 113.097335529. Puedes volver a ejecutar el programa y, en lugar de teclear el número 3, teclear cualquier otro valor; Python nos responderá con el valor del volumen de la esfera para un radio igual al valor que hayas tecleado.

Pero el programa no es muy elegante, pues deja al ordenador bloqueado hasta que el usuario teclee una cantidad y no informa de qué es exactamente esa cantidad. Vamos

a hacer que el programa indique, mediante un mensaje, qué dato desea que se teclee. La función *raw_input* acepta un argumento: una *cadena* con el mensaje que debe mostrar.

Modifica el programa para que quede así:

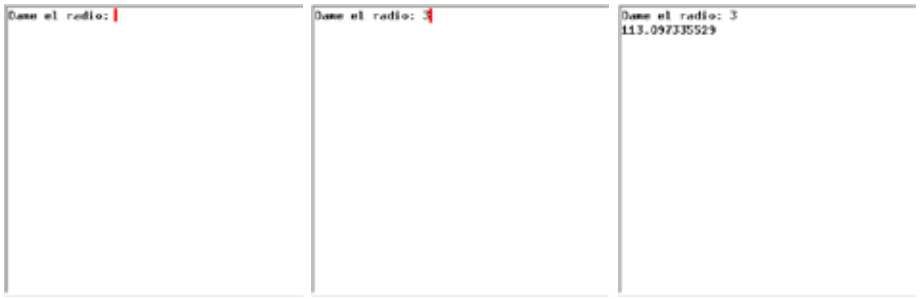
volumen_esfera_12.py — volumen_esfera.py

```
1 from math import pi
2
3 radio = float(raw_input('Dame el radio: '))
4 volumen = 4.0 / 3.0 * pi * radio ** 3
5
6 print volumen
```

Ahora, cada vez que lo ejecutes, mostrará por pantalla el mensaje «`Dame el radio:`» y detendrá su ejecución hasta que introduzcas un número y pulses el retorno de carro.

```
$ python volumen_esfera.py ↲
Dame el radio: 3
113.097335529
```

La forma de uso del programa desde PythonG es muy similar. Cuando ejecutas el programa, aparece el mensaje «`Dame el radio:`» en la consola de entrada/salida y se detiene la ejecución (figura 3.4 (a)). El usuario debe teclear el valor del radio, que va apareciendo en la propia consola (figura 3.4 (b)), y pulsar al final la tecla de retorno de carro. El resultado aparece a continuación en la consola (figura 3.4 (c)).




(a) (b) (c)

**Figura 3.4:** Entrada/salida en la consola al ejecutar el programa `volumen_esfera.py`.

## EJERCICIOS

▶ 34 Diseña un programa que pida el valor del lado de un cuadrado y muestre el valor de su perímetro y el de su área.

(Prueba que tu programa funciona correctamente con este ejemplo: si el lado vale 1.1, el perímetro será 4.4, y el área 1.21.)

▶ 35 Diseña un programa que pida el valor de los dos lados de un rectángulo y muestre el valor de su perímetro y el de su área.

(Prueba que tu programa funciona correctamente con este ejemplo: si un lado mide 1 y el otro 5, el perímetro será 12.0, y el área 5.0.)

▶ 36 Diseña un programa que pida el valor de la base y la altura de un triángulo y muestre el valor de su área.

(Prueba que tu programa funciona correctamente con este ejemplo: si la base es 10 y la altura 100, el área será 500.0.)

▶ 37 Diseña un programa que pida el valor de los tres lados de un triángulo y calcule el valor de su área y perímetro.

Recuerda que el área $A$ de un triángulo puede calcularse a partir de sus tres lados, $a$, $b$ y $c$, así: $A = \sqrt{s(s-a)(s-b)(s-c)}$, donde $s = (a+b+c)/2$.

(Prueba que tu programa funciona correctamente con este ejemplo: si los lados miden 3, 5 y 7, el perímetro será 15.0 y el área 6.49519052838.)

. . . . . .