▶ 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.)

3.3.2. Más sobre la sentencia print

Las cadenas pueden usarse también para mostrar textos por pantalla en cualquier momento a través de sentencias **print**.

volumen_esfera_13.py — volumen_esfera.py

```
1 from math import pi
2
3 print 'Programa␣para␣el␣cálculo␣del␣volumen␣de␣una␣esfera.'
4
5 radio = float(raw_input('Dame␣el␣radio:␣'))
6 volumen = 4.0 / 3.0 * pi * radio ** 3
7
8 print volumen
9 print 'Gracias␣por␣utilizar␣este␣programa.'
```

Cuando ejecutes este programa, fíjate en que las cadenas que se muestran con **print** no aparecen entrecomilladas. El usuario del programa no está interesado en saber que le estamos mostrando datos del tipo cadena: sólo le interesa el texto de dichas cadenas. Mucho mejor, pues, no mostrarle las comillas.

Una sentencia **print** puede mostrar más de un resultado en una misma línea: basta con separar con comas todos los valores que deseamos mostrar. Cada una de las comas *se traduce en un espacio de separación*. El siguiente programa:

volumen_esfera_14.py — volumen_esfera.py

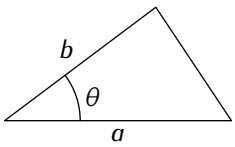
```
1 from math import pi
2
3 print 'Programa␣para␣el␣cálculo␣del␣volumen␣de␣una␣esfera.'
4
5 radio = float(raw_input('Dame␣el␣radio␣(en␣metros):␣'))
6 volumen =  4.0/3.0 * pi * radio ** 3
7
8 print 'Volumen␣de␣la␣esfera:', volumen, 'metros␣cúbicos'
```

hace que se muestre el texto «`Volumen de la esfera:`», seguido del valor de la variable *volumen* y acabado con «`metros cúbicos`». Observa que los elementos del último **print** se separan entre sí por espacios en blanco:

```
Programa para el cálculo del volumen de una esfera.
Dame el radio (en metros): 2
El volumen de la esfera es de 33.5103216383 metros cúbicos
```

EJERCICIOS

▶ 38 El área A de un triángulo se puede calcular a partir del valor de dos de sus lados, a y b, y del ángulo θ que éstos forman entre sí con la fórmula $A = \frac{1}{2}ab\sin(\theta)$. Diseña un programa que pida al usuario el valor de los dos lados (en metros), el ángulo que estos forman (en grados), y muestre el valor del área.



(Ten en cuenta que la función *sin* de Python trabaja en radianes, así que el ángulo que leas en grados deberás pasarlo a radianes sabiendo que π radianes son 180 grados. Prueba que has hecho bien el programa introduciendo los siguientes datos: $a = 1$, $b = 2$, $\theta = 30$; el resultado es 0.5.)

▶ 39 Haz un programa que pida al usuario una cantidad de euros, una tasa de interés y un número de años. Muestra por pantalla en cuánto se habrá convertido el capital inicial transcurridos esos años si cada año se aplica la tasa de interés introducida.

Recuerda que un capital de C euros a un interés del x por cien durante n años se convierten en $C \cdot (1 + x/100)^n$ euros.

(Prueba tu programa sabiendo que una cantidad de 10 000 € al 4.5% de interés anual se convierte en 24 117.14 € al cabo de 20 años.)

▶ 40 Haz un programa que pida el nombre de una persona y lo muestre en pantalla repetido 1000 veces, pero dejando un espacio de separación entre aparición y aparición del nombre. (Utiliza los operadores de concatenación y repetición.)

Por lo visto hasta el momento, cada **print** empieza a imprimir en una nueva línea. Podemos evitarlo si el *anterior* **print** finaliza en una coma. Fíjate en este programa:

volumen_esfera.py

```
1 from math import pi
2
3 print 'Programa para el cálculo del volumen de una esfera.'
4
5 radio = float(raw_input('Dame el radio (en metros): '))
6 volumen =  4.0/3.0 * pi * radio ** 3
7
8 print 'Volumen de la esfera:',
9 print volumen, 'metros cúbicos'
```

La penúltima línea es una sentencia **print** que finaliza en una coma. Si ejecutamos el programa obtendremos un resultado absolutamente equivalente al de la versión anterior:

```
Programa para el cálculo del volumen de una esfera.
Dame el radio (en metros): 2
El volumen de la esfera es de 33.5103216383 metros cúbicos
```

Ocurre que cada **print** imprime, en principio, un carácter especial denominado «nueva línea», que hace que el cursor (la posición en la que se escribe la salida por pantalla en cada instante) se desplace a la siguiente línea. Si **print** finaliza en una coma, Python no imprime el carácter «nueva línea», así que el cursor no se desplaza a la siguiente línea. El siguiente **print**, pues, imprimirá inmediatamente a continuación, en la misma línea.

3.3.3. Salida con formato

Con la sentencia **print** podemos controlar hasta cierto punto la apariencia de la salida. Pero no tenemos un control total:

- Cada coma en la sentencia **print** hace que aparezca un espacio en blanco en la pantalla. ¿Y si no deseamos que aparezca ese espacio en blanco?

- Cada número ocupa tantas «casillas» de la pantalla como caracteres tiene. Por ejemplo, el número 2 ocupa una casilla, y el número 2000, cuatro. ¿Y si queremos que todos los números ocupen el mismo número de casillas?

Python nos permite controlar con absoluta precisión la salida por pantalla. Para ello hemos de aprender un nuevo uso del operador **%**. Estudia detenidamente este programa:

potencias_1.py — potencias.py

```
numero = int(raw_input('Dame un número: '))

print '%d elevado a %d es %d' % (numero, 2, numero ** 2)
print '%d elevado a %d es %d' % (numero, 3, numero ** 3)
print '%d elevado a %d es %d' % (numero, 4, numero ** 4)
print '%d elevado a %d es %d' % (numero, 5, numero ** 5)
```

Cada una de las cuatro últimas líneas presenta este aspecto:

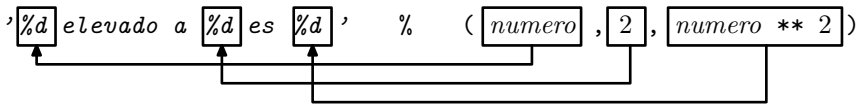
print *cadena* **%** (*valor* , *valor* , *valor*)

La *cadena* es especial, pues tiene unas marcas de la forma **%d**. ¿Tienen algún significado? Después de la cadena aparece el operador **%**, que hemos visto en el tema anterior como operador de enteros o flotantes, pero que aquí combina una cadena (a su izquierda) con una serie de valores (a su derecha). ¿Qué hace en este caso?

Para entender qué hacen las cuatro últimas líneas, ejecutemos el programa:

```
Dame un número: 3
3 elevado a 2 es 9
3 elevado a 3 es 27
3 elevado a 4 es 81
3 elevado a 5 es 243
```

Cada *marca de formato* **%d** en la cadena **'%d elevado a %d es %d'** ha sido sustituida por un número entero. El fragmento **%d** significa «aquí va un número entero». ¿Qué número? El que resulta de evaluar cada una de las tres expresiones que aparecen separadas por comas y entre paréntesis a la derecha del operador **%**.



No sólo podemos usar el operador **%** en cadenas que vamos a imprimir con **print**: el resultado es una cadena y se puede manipular como cualquier otra:

```
>>> x = 2 ↵
>>> print 'número %d y número %d' % (1, x) ↵
número 1 y número 2
>>> a = 'número %d y número %d' % (1, x) ↵
>>> a ↵
'número 1 y número 2'
>>> print ('número %d y número %d' % (1, x)).upper() ↵
NÚMERO 1 Y NÚMERO 2
```

EJERCICIOS

▶ 41 ¿Qué mostrará por pantalla este programa?

```
print '%d' % 1
print '%d %d' % (1, 2)
print '%d%d' % (1, 2)
print '%d, %d' % (1, 2)
print 1, 2
print '%d 2' % 1
```

▶ 42 Un alumno inquieto ha experimentado con las marcas de formato y el método *upper* y ha obtenido un resultado sorprendente:

```
>>> print ('número %d y número %d' % (1, 2)).upper() ↲
NÚMERO 1 Y NÚMERO 2
>>> print 'número %d y número %d'.upper() % (1, 2) ↲
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: unsupported format character 'D' (0x44) at index 8
```

¿Qué crees que ha pasado?

(Nota: Aunque experimentar conlleva el riesgo de equivocarse, no podemos enfatizar suficientemente cuán importante es para que asimiles las explicaciones. Probarlo todo, cometer errores, reflexionar sobre ellos y corregirlos es uno de los mejores ejercicios imaginables.)

Vamos a modificar ligeramente el programa:

potencias.py

```
numero = int(raw_input('Dame un número: '))

print '%d elevado a %d es %4d' % (numero, 2, numero ** 2)
print '%d elevado a %d es %4d' % (numero, 3, numero ** 3)
print '%d elevado a %d es %4d' % (numero, 4, numero ** 4)
print '%d elevado a %d es %4d' % (numero, 5, numero ** 5)
```

El tercer `%d` de cada línea ha sido sustituido por un `%4d`. Veamos qué ocurre al ejecutar el nuevo programa:

```
Dame un número: 3
3 elevado a 2 es    9
3 elevado a 3 es   27
3 elevado a 4 es   81
3 elevado a 5 es  243
```

Los números enteros que ocupan la tercera posición aparecen alineados a la derecha. El fragmento `%4d` significa «aquí va un entero que representaré ocupando 4 casillas». Si el número entero tiene 4 o menos dígitos, Python lo representa dejando delante de él los espacios en blanco que sea menester para que ocupe exactamente 4 espacios. Si tiene más de 4 dígitos, no podrá cumplir con la exigencia impuesta, pero seguirá representando el número entero correctamente.

Hagamos la prueba. Ejecutemos de nuevo el mismo programa, pero introduciendo otro número:

```
Dame un número: 7
7 elevado a 2 es   49
7 elevado a 3 es  343
7 elevado a 4 es 2401
7 elevado a 5 es 16807
```

¿Ves? El último número tiene cinco dígitos, así que «se sale» por el margen derecho.

Las cadenas con marcas de formato como **%d** se denominan *cadenas con formato* y el operador **%** a cuya izquierda hay una cadena con formato es el *operador de formato*.

Las cadenas con formato son especialmente útiles para representar adecuadamente números flotantes. Fíjate en el siguiente programa:

area_con_formato.py

```
1 from math import pi
2
3 radio = float(raw_input('Dame el radio: '))
4 area = pi*radio**2
5
6 print 'El área de un círculo de radio %f es %f' % (radio, area)
7 print 'El área de un círculo de radio %6.3f es %6.3f' % (radio, area)
```

Ejecutemos el programa:

```
Dame el radio: 2
El área de un círculo de radio 2.000000 es 12.566371
El área de un círculo de radio  2.000 es 12.566
```

Observa: la marca **%f** indica que ahí aparecerá un flotante. Podemos meter un número con decimales entre el **%** y la **f**. ¿Qué significa? Indica cuántas casillas deseamos que ocupe el flotante (parte entera del número entre la **%** y la **f**) y, de ellas, cuántas queremos que ocupen los números decimales (parte decimal del mismo número).

Hemos visto que hay marcas de formato para enteros y flotantes. También hay una marca para cadenas: **%s**. El siguiente programa lee el nombre de una persona y la saluda:

saluda_3.py — saluda.py

```
1 nombre = raw_input('Tu nombre: ')
2 print 'Hola, %s.' % (nombre)
```

Probemos el programa:

```
Tu nombre: Juan
Hola, Juan.
```

¡Ah! Los paréntesis en el argumento de la derecha del operador de formato son opcionales si sólo se le pasa un valor. Esta nueva versión del programa es equivalente:

saluda_4.py — saluda.py

```
1 nombre = raw_input('Tu nombre: ')
2 print 'Hola, %s.' % nombre
```

EJERCICIOS

▶ 43 ¿Qué pequeña diferencia hay entre el programa **saluda.py** y este otro cuando los ejecutamos?

saluda2.py

```
1 nombre = raw_input('Tu nombre: ')
2 print 'Hola,', nombre, '.'
```

▶ 44 La marca **%s** puede representar cadenas con un número fijo de casillas. A la vista de cómo se podía expresar esta característica en la marca de enteros **%d**, ¿sabrías como indicar que deseamos representar una cadena que ocupa 10 casillas?