Funciones y Módulos

Juan Álvarez Rubio

En matemáticas se usan distintas notaciones para definir funciones. Por ejemplo, para definir las funciones f y g se suele escribir $f(x): x^2-2x+1$ y $g(x,y): (x+y)^2$ que responden a la forma general nombre(parámetros): expresión. Una vez que las funciones están definidas, la notación para su uso o aplicación suele ser nombre(argumentos). Por ejemplo $f(3)$ entrega el resultado 4 porque el argumento 3 reemplaza al parámetro x . Por lo tanto se calcula la expresión $3^2 - 2*3 + 1$. De la misma manera $g(1,2)$ evalúa la expresión $(1+2)^2$ y entrega el valor 9. En Python, las funciones f y g se definen de la siguiente manera:

```
def f(x):  
    return x**2-2*x+1  
def g(x,y):  
    return (x+y)**2
```

Sintaxis y semántica de una función

La sintaxis o forma general de una función en Python es la siguiente:

```
def nombre(parámetros):  
    instrucciones (cero o más)  
    return expresión
```

Tanto el nombre de la función como los nombres de los parámetros (si es que existen) deben respetar las reglas del nombre de una variable. Las instrucciones pueden existir o no, pero la última debe ser una instrucción `return` que incluye una expresión para calcular el resultado de la función. Todas las instrucciones deben escribirse indentadas o alineadas en un mismo margen ubicado más adentro que el encabezamiento de la función (se recomienda que sea de al menos dos espacios). Si las instrucciones no están indentadas o alineadas correctamente se informará como un error de sintaxis.

Por otra parte, la sintaxis del uso o invocación a una función es nombre(argumentos) en que los argumentos son cero o más expresiones separadas por comas. La semántica o significado es similar a las funciones matemáticas, es decir cuando una función se usa o invoca primero se evalúan las expresiones de los argumentos y sus resultados se asignan a los parámetros. Por ejemplo, en el caso de $g(1,2)$ entonces se realizan las asignaciones $x=1$ e $y=2$. En seguida, se ejecutan las instrucciones si es que existen. Finalmente se evalúa la expresión de la instrucción `return`, por ejemplo $(x+y)**2$, y su resultado (9) se entrega o retorna en el lugar en que se usó o invocó la función. A modo de ejemplo, una función para calcular el área de un círculo puede escribirse de la siguiente manera:

```
def areaCirculo(r):  
    pi=3.14  
    return pi*r**2
```

En este caso se eligieron nombres apropiados para la función y el parámetro. En caso que la función se invoque con `areaCirculo(1)`, primero se copia el argumento 1 en el parámetro r con la asignación $r=1$. En seguida se ejecuta la asignación $pi=3.14$ y finalmente se evalúa la expresión $3.14*1**2$ y su resultado 3.14 se retorna al lugar en que se llamó o invocó la función. Cabe señalar que el parámetro r y la variable interna pi desaparecen, al momento

que la función termina al devolver el resultado, y por lo tanto no pueden ser usadas fuera de la función. El parámetro `r` y la variable `pi` se volverán a crear (y desaparecer) en caso que la función se invoque nuevamente.

Receta de Diseño de una función

Es recomendable que una función se grave en un archivo (con el mismo nombre de la función) de modo que pueda ser usada desde distintos programas. En consecuencia, las funciones deben escribirse con una buena documentación que facilite su uso posterior, eventualmente mucho tiempo después y por otras personas. Por ejemplo, la función `areaCirculo` se escribirá de la siguiente manera:

```
#areaCirculo: num -> float  
#objetivo: calcular área de un círculo de radio r  
#ejemplo: areaCirculo(1) debe entregar 3.14  
def areaCirculo(r):  
    pi=3.14  
    return pi*r**2  
assert areaCirculo(1)==3.14
```

Las líneas que comienzan con `#` corresponden a comentarios que Python ignora pero que sirven para escribir explicaciones y documentar las funciones, las instrucciones y los programas. La función se ha escrito siguiendo una “receta de diseño”, es decir, de acuerdo a un esquema que contenga información de los tipos de los parámetros y del resultado, del objetivo de la función, algún ejemplo de uso y una comprobación del resultado esperado. La forma general de la “receta de diseño” es la siguiente:

```
#nombre: tipos parámetros -> tipo resultado  
#objetivo  
#ejemplo(s)  
def nombre(parámetros):  
    ...  
#prueba(s) de ejemplo(s)  
assert nombre(argumentos)==valor
```

La primera línea de la receta de diseño (llamada contrato) especifica el nombre de la función y los tipos, tanto de los parámetros, como del resultado. Los tipos posibles son `int`, `float`, `str` o `num` (que significa que el tipo puede ser `int` o `float`). La segunda línea indica el objetivo de la función, es decir el resultado que entregará. Este objetivo podría redactarse en más de una línea. En tercer lugar, se especifican uno o más ejemplos de uso de la función y los resultados esperados. A continuación, se escribe la función siguiendo las reglas de sintaxis indicadas en la sección anterior.

La receta de diseño termina con las pruebas (testing) de la función usando una o más instrucciones `assert` que sirven para probar los ejemplos indicados y asegurar (comprobar) que la función arroja los resultados prometidos. La instrucción `assert` contiene una comparación entre el resultado que entrega una invocación a la función con el valor esperado. Al respecto, la comparación por igualdad se especifica con `==` (para diferenciarlo del signo `=` que se utiliza en la instrucción de asignación). La prueba con la instrucción `assert` ocurrirá al momento de seleccionar la opción de ejecución del archivo que contiene la función y en caso que la comparación no se cumpla se informará

como un error ("assertion error"). Por otra parte, si la condición se cumple, es decir si la comparación arroja un resultado verdadero, entonces se ejecutará la instrucción siguiente (si es que existe).

Módulos

Habitualmente se usan funciones que están relacionadas. Por ejemplo, además de la función que calcula el área de un círculo sería conveniente definir otra función que calcule el perímetro. En este caso se utiliza un módulo que contenga las dos funciones y que se graba en el archivo `circulo.py`:

```
#modulo circulo: en archivo circulo.py
pi=3.14
```

```
#perimetro: num -> float
#perímetro de círculo de radio r
#ej: perimetro(1) -> 6.28
def perimetro(r):
    return 2*pi*r
assert perimetro(1)==6.28
```

```
#area: num -> float
#área de círculo de radio r
#ej: area(1) -> 3.14
def area(r):
    return pi*r**2
assert area(1)==3.14
```

Las funciones del módulo se escriben siguiendo la receta de diseño. Dado que las dos funciones están dentro del módulo `circulo.py` entonces sus nombres son simplemente `perimetro` y `area`. Como las funciones calculan y entregan un resultado, entonces el objetivo se puede redactar indicando solo el valor que calculan. Por otra parte, en el ejemplo se puede utilizar la notación `->` para indicar el valor esperado. Adicionalmente, el módulo contiene una variable con una aproximación del valor de `pi`. Su ubicación permite usar la variable `pi` en las funciones del módulo y en los programas que utilicen el módulo, en otras palabras, la variable `pi` será visible dentro y fuera del módulo. La prueba del módulo se realiza en la misma forma que una función, es decir con la opción ejecutar del archivo en el menú de Python.

Un programa que use el módulo para leer el valor del radio de un círculo y escribir el valor del perímetro y del área se puede escribir de la siguiente manera:

```
import circulo
radio=float(input("radio de círculo?"))
print("área de círculo:",circulo.area(radio))
print("perímetro de círculo:",circulo.perimetro(radio))
```

La instrucción `import` incluye o deja disponible el módulo para ser usado en un programa. Para usar las funciones se debe escribir `nombreModulo.nombreFunción`, por ejemplo `circulo.area`. Para usar la variable `pi` también se tendría que escribir `circulo.pi`. Una manera más breve de escribir el programa que usa el módulo es la siguiente:

```
from circulo import *
r=float(input("radio de círculo?"))
print("área de círculo:", area(r))
print("perímetro de círculo:", perimetro(r))
```

La instrucción `from circulo import *` deja disponibles todas las funciones del módulo y pueden utilizarse solo con su nombre (lo mismo si se necesita usar la variable `pi`). Por otra parte, en este programa se ha utilizado una variable con el nombre `r` para representar el radio del círculo. El nombre coincide con el nombre del parámetro de las funciones `area` y `perimetro` pero no hay confusión ni produce ningún problema porque son distintas (una es una variable del programa y la otra un parámetro que desaparece al terminar la función).

Módulos predefinidos

Python tiene predefinidos varios módulos. Por ejemplo, el módulo `math` contiene varias funciones matemáticas de uso frecuente e incluso un valor más preciso de `pi`. Algunas de las funciones y sus resultados son las siguientes:

```
import math
math.sqrt(2)
1.4142135623730951
math.pi
3.141592653589793
angulo=90
radianes=math.pi*angulo/180
math.sin(radianes)
1.0
math.cos(radianes)
6.123233995736766e-17
```

La función `math.sqrt` (square root) calcula la raíz cuadrada de un número y parece innecesaria puesto que se podría calcular con `2**0.5`, sin embargo utiliza un algoritmo de cálculo mucho más eficiente. Por otra parte, las funciones trigonométricas (`sin`, `cos`, `tan`) necesitan los ángulos expresados en radianes. El coseno de un ángulo de 90 grados (`pi/2` en radianes) es 0, sin embargo, el resultado que se muestra corresponde al valor $6.1... \times 10^{-17}$, es decir un valor muy pequeño que es una aproximación del valor 0 que fue calculado con una serie. Otras funciones del módulo `math` se pueden averiguar en el sitio web de Python.

Por otra parte, el módulo `random` contiene funciones para generar números al azar reales y enteros: `random()` produce un número real entre 0 y 1 (de la forma $0.n^{\circ}$) y `randint(x,y)` genera un número entero entre `x` y `y`:

```
import random
random.random()
0.3504750773904878
random.randint(1,6) #simula dado
4
random.randint(0,1) #simula moneda
1
```

Si se utiliza la instrucción `from modulo import *` para los módulos predefinidos `math` y `random` entonces las funciones se pueden invocar solo con su nombre. Por otra parte, además de `input` y `print`, existen otras funciones predefinidas en Python que no pertenecen a ningún módulo y que se pueden usar sin tener que importar un módulo:

```
abs(-1)      #valor absoluto de un argumento int
1            #resultado int
abs(3.5)     #argumento float
3.5         #resultado float
max(3,4,2)   #mayor entre 2 o más argumentos
4
min(3,4,2)   #menor entre 2 o más argumentos
2
int(4.9)     #argumento float
4            #resultado int truncado (sin decimals)
int('123')  #argumento str
123         #resultado int
float(2)     #argumento int
2.0         #resultado float
float('2.5') #argumento str
2.5        #resultado float
str(123)     #argumento int
'123'       #resultado str
str(2.5)     #argumento float
'2.5'       #resultado str
```