Desconocido

### Variables y ámbito

Una variable que está dentro del cuerpo de una función no puede volver a utilizarse una vez que la función ha terminado de ejecutarse, porque sólo existe dentro de la función. En el mundo de la programación, el lugar donde se puede utilizar una variable se denomina ámbito *.* Veamos una función sencilla que utiliza un par de variables pero no tiene ningún parámetro:

>>> def variable\_test():  
 first\_variable = 10  
 second\_variable = 20  
 return first\_variable \* second\_variable

En este ejemplo, creamos la función variable\_test , que multiplica first\_variable y second\_variable y devuelve el resultado:

>>> print(variable\_test())  
200

Si llamamos a esta función utilizando print , obtenemos 200. Sin embargo, si intentamos imprimir el contenido de first\_variable (o second\_variable ,  para el caso) fuera del bloque de código de la función, obtendremos un mensaje de error:

>>> print(first\_variable)  
Traceback (most recent call last):  
 File "<pyshell#50>", line 1, in <module>  
 print(first\_variable)  
NameError: name 'first\_variable' is not defined

Si una variable se define fuera de la función, tiene un ámbito diferente. Por ejemplo, definamos una variable antes de crear nuestra función, y luego intentemos utilizarla dentro de la función:

➊ >>> another\_variable = 100  
 >>> def variable\_test2():  
 first\_variable = 10  
 second\_variable = 20  
 ➋ return first\_variable \* second\_variable \* another\_variable

En este código, aunque las variables first\_variable y second\_variable no se pueden utilizar fuera de la función, la variable another\_variable (que se creó fuera de la función ➊ ) se puede utilizar dentro de ella ➋ .

Éste es el resultado de llamar a esta función:

>>> print(variable\_test2())  
20000

Ahora, supón que construyes una nave espacial con algo económico como latas de conserva recicladas. Puedes aplanar dos latas a la semana para crear las paredes curvas de tu nave espacial, pero necesitarás unas 500 latas para terminar el fuselaje. Intentemos escribir una función que imprima el total de latas aplastadas cada semana a lo largo de un año.



Nuestra función calculará cuántas latas hemos aplastado cada semana, con el número de latas como parámetro (así será más fácil cambiar el número de latas más adelante):

>>> def spaceship\_building(cans):  
 total\_cans = 0  
 for week in range(1, 53):  
 total\_cans = total\_cans + cans  
 print(f'Week {week} = {total\_cans} cans')

En la primera línea de la función, creamos la variable total\_cans y establecemos su valor en 0. A continuación, creamos un bucle para las semanas del año y añadimos el número de latas aplastadas cada semana. Este bloque de código constituye el contenido de nuestra función, y las dos últimas líneas constituyen otro bloque del bucle for .

Probemos a introducir esa función en la Shell de Python y a llamarla con distintos valores para el número de latas, empezando por 2:

>>> spaceship\_building(2)  
Week 1 = 2 cans  
Week 2 = 4 cans  
Week 3 = 6 cans  
Week 4 = 8 cans  
Week 5 = 10 cans  
Week 6 = 12 cans  
Week 7 = 14 cans  
Week 8 = 16 cans  
...  
Week 50 = 100 cans  
Week 51 = 102 cans  
Week 52 = 104 cans  
  
>>> spaceship\_building(10)  
Week 1 = 10 cans  
Week 2 = 20 cans  
Week 3 = 30 cans  
Week 4 = 40 cans  
Week 5 = 50 cans  
...  
Week 48 = 480 cans  
Week 49 = 490 cans  
Week 50 = 500 cans  
Week 51 = 510 cans  
Week 52 = 520 cans

Esta función puede reutilizarse con distintos valores para el número de latas por semana, lo que es un poco más eficaz que volver a escribir el bucle for cada vez que quieras probarlo con distintos números  . Cuando ejecutamos spaceship\_building(10) , podemos ver que tendremos suficientes latas para construir las paredes de la nave espacial en la semana 50.

Las funciones también pueden agruparse en módulos, lo que hace que Python sea *realmente* útil, en lugar de sólo medianamente útil.

[anterior](ch07_3.html)[Subtema 4 de 8: (Ver todo)](ch07.html)[siguiente](ch07_5.html)