Funciones (2)

En esta lección se tratan conceptos relativos a las funciones. Antes de leer esta lección se recomienda leer la lección Funciones (1):

Argumentos y devolución de valores

Las funciones en Python admiten argumentos en su llamada y permiten devolver valores. Estas posiblidades permiten crear funciones más útiles y fácilmente reutilizables.

En este apartado se muestran estos conceptos mediante cuatro ejemplos. En ellos, no se pretende encontrar la mejor solución al problema planteado, sino simplemente introducir los conceptos de argumentos y devolución de valores.

Aunque las funciones en Python pueden acceder a cualquier variable del programa declarándolas como variables globales o no locales, se necesita saber el nombre de las variables, como muestra el ejemplo siguiente:

Ejemplo de función 1

```
def escribe_media():
    media = (a + b) / 2
    print(f"La media de {a} y {b} es:
{media}")
    return

a = 3
b = 5
escribe_media()
print("Programa terminado")
```

```
La media de 3 y 5 es: 4.0
Programa terminado
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

El problema de una función de este tipo es que es muy difícil de reutilizar en otros programas o incluso en el mismo programa, ya que sólo es capaz de hacer la media de las variables "a" y "b". Si en el programa no se utilizan esos nombres de variables, la función no funcionaría.

Para evitar ese problema, las funciones admiten argumentos, es decir, permiten que se les envíen valores con los que trabajar. De esa manera, las funciones se pueden reutilizar más fácilmente, como muestra el ejemplo siguiente:

Ejemplo de función 2

```
def escribe_media(x, y):
    media = (x + y) / 2
    print(f"La media de {x} y {y} es:
{media}")
    return

a = 3
b = 5
escribe_media(a, b)
print("Programa terminado")
```

```
La media de 3 y 5 es: 4.0
Programa terminado
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Esta función puede ser utilizada con más flexibilidad que la del ejemplo anterior, puesto que el programador puede elegir a qué valores aplicar la función.

Pero esta función tiene todavía un inconveniente. Como las variables locales de una función son inaccesibles desde los niveles superiores, el programa principal no puede utilizar la variable "media" calculada por la función y tiene que ser la función la que escriba el valor. Pero eso complica la reutilización de la función, porque aunque es probable que en otro programa nos interese una función que calcule la media, es más difícil que nos interese que escriba el valor nada más calcularlo.

Para evitar ese problema, las funciones pueden devolver valores, es decir, pueden enviar valores al programa o función de nivel superior. De esa manera, las funciones se pueden reutilizar más fácilmente, como muestra el ejemplo siguiente:

Ejemplo de función 3

```
def calcula_media(x, y):
    resultado = (x + y) / 2
    return resultado

a = 3
b = 5
media = calcula_media(a, b)
print(f"La media de {a} y {b} es:
{media}")
print("Programa terminado")
```

```
La media de 3 y 5 es: 4.0
Programa terminado
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Esta función puede ser utilizada con más flexibilidad que la del ejemplo anterior, puesto que el programador puede elegir qué hacer con el valor calculado por la función.

Pero esta función tiene todavía un inconveniente y es que sólo calcula la media de dos valores. Sería más interesante que la función calculara la media de cualquier cantidad de valores.

Para evitar ese problema, las funciones pueden admitir una cantidad indeterminada de valores, como muestra el ejemplo siguiente:

Ejemplo de función 4

```
def calcula_media(*args):
    total = 0
    for i in args:
        total += i
    resultado = total / len(args)
    return resultado

a, b, c = 3, 5, 10
media = calcula_media(a, b, c)
print(f"La media de {a}, {b} y {c} es:
{media}")
print("Programa terminado")
```

```
La media de 3, 5 y 10 es: 6.0
Programa terminado
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Esta última función puede ser utilizada con más flexibilidad que las anteriores, puesto que el programador puede elegir de cuántos valores hacer la media y qué hacer con el valor calculado por la función.

Las funciones pueden devolver varios valores simultáneamente, como muestra el siguiente ejemplo:

Ejemplo de función 5

```
def calcula_media_desviacion(*args):
    total = 0
    for i in args:
        total += i
    media = total / len(args)
    total = 0
    for i in args:
        total += (i - media) ** 2
    desviacion = (total / len(args)) **
0.5
    return media, desviacion

a, b, c, d = 3, 5, 10, 12
```

```
Datos: 3 5 10 12
Media: 7.5
Desviación típica: 3.640054944640259
Programa terminado
```

```
media, desviacion_tipica =
calcula_media_desviacion(a, b, c, d)
print(f"Datos: {a} {b} {c} {d}")
print(f"Media: {media}")
print(f"Desviación típica:
{desviacion_tipica}")
print("Programa terminado")
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Conflictos entre nombres de parámetros y nombre de variables globales

En Python no se producen conflictos entre los nombres de los parámetros y los nombres de las variables globales. Es decir, el nombre de un parámetro puede coincidir o no con el de una variable global, pero Python no los confunde: en el ámbito de la función el parámetro hace siempre referencia al dato recibido y no a la variable global y los programas producen el mismo resultado.

Eso nos facilita reutilizar funciones en otros programas sin tener que preocuparnos por este detalle.

Aunque, como siempre, dependiendo de si a la función se le envía como parámetro un objeto mutable o inmutable, la función podrá modificar o no al objeto. En los dos siguientes ejemplos, el parámetro de la función ("b") se llama igual que una de las dos variables del programa principal. En los dos ejemplos se llama dos veces a la función, enviando cada vez una de las dos variables ("a" y "b").

- **Ejemplo de conflicto entre nombre de parámetro y nombre de variable global. Objeto mutable**

  Como en este caso las variables son listas (objetos mutables), la función modifica la lista que se envía como argumento: primero se modifica la lista "a" y a continuación la lista "b". La lista modificada no depende del nombre del parámetro en la función (que es "b"), sino de la variable enviada como argumento ("a" o "b").

```
def cambia(b):
    b += [5]
    return

a, b = [3], [4]
print(f"Al principio        : a = {a} b
= {b}")
cambia(a)
print(f"Después de cambia(a): a = {a} b
= {b}")
cambia(b)
print(f"Después de cambia(b): a = {a} b
```

```
Al principio        : a = [3] b = [4]
Después de cambia(a): a = [3, 5] b = [4]
Después de cambia(b): a = [3, 5] b =
[4, 5]
Programa terminado
```

```
= {b}")
print("Programa terminado")
```

- **Ejemplo de conflicto entre nombre de parámetro y nombre de variable global. Objeto inmutable**

  Como en este caso las variables son números enteros (objetos inmutables), la función no puede modificar los números que se envían como argumentos, ni la variable "a" ni la variable "b".

```
def cambia(b):
    b += 1
    return

a, b = 3, 4
print(f"Al principio        : a = {a} b
= {b}")
cambia(a)
print(f"Después de cambia(a): a = {a} b
= {b}")
cambia(b)
print(f"Después de cambia(b): a = {a} b
= {b}")
print("Programa terminado")
```

```
Al principio        : a = 3 b = 4
Después de cambia(a): a = 3 b = 4
Después de cambia(b): a = 3 b = 4
Programa terminado
```

Paso por valor o paso por referencia

En los lenguajes en los que las variables son "cajas" en las que se guardan valores, cuando se envía una variable como argumento en una llamada a una función suelen existir dos posiblidades:

- paso por valor: se envía simplemente el valor de la variable, en cuyo caso la función no puede modificar la variable, pues la función sólo conoce su valor, pero no la variable que lo almacenaba.
- paso por referencia: se envía la dirección de memoria de la variable, en cuyo caso la función sí que puede modificar la variable.

En Python no se hace ni una cosa ni otra. En Python cuando se envía una variable como argumento en una llamada a una función lo que se envía es la referencia al objeto al que hace referencia la variable. Dependiendo de si el objeto es mutable o inmutable, la función podrá modificar o no el objeto.

- En el ejemplo siguiente, la variable enviada a la función es una variable que hace referencia a un objeto inmutable.

Ejemplo de paso de variable (objeto inmutable)

```
def aumenta(x):
    print(id(x))
    x += 1
    print(id(x))
    return x
```

```
505894336 505894352
505894336
505894336
505894352
4
```

```
a = 3
print(id(3), id(4))
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

```
3
505894336
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

- En el ejemplo siguiente, la variable enviada a la función es una variable que hace referencia a un objeto mutable.

Ejemplo de paso de variable (objeto mutable)

```
def aumenta(x):
    print(id(x))
    x += [1]
    print(id(x))
    return x

a = [3]
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

```
45688512
45688512
45688512
[3, 1]
[3, 1]
45688512
```

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

Última modificación de esta página: 4 de abril de 2017