6.5.4. Acceso a variables globales desde funciones

Por lo dicho hasta ahora podrías pensar que en el cuerpo de una función sólo pueden utilizarse variables locales. No es cierto. Dentro de una función también puedes consultar y modificar variables globales. Eso sí, deberás «avisar» a Python de que una variable usada en el cuerpo de una función es global *antes* de usarla. Lo veremos con un ejemplo.

Vamos a diseñar un programa que gestiona una de las funciones de un cajero automático que puede entregar cantidades que son múltiplo de 10 €. En cada momento, el cajero tiene un número determinado de billetes de 50, 20 y 10 €. Utilizaremos una variable para cada tipo de billete y en ella indicaremos cuántos billetes de ese tipo nos quedan en el cajero. Cuando un cliente pida sacar una cantidad determinada de dinero, mostraremos por pantalla cuántos billetes de cada tipo le damos. Intentaremos darle siempre la menor cantidad de billetes posible. Si no es posible darle el dinero (porque no tenemos suficiente dinero en el cajero o porque la cantidad solicitada no puede darse con una combinación válida de los billetes disponibles) informaremos al usuario.

Inicialmente supondremos que el cajero está cargado con 100 billetes de cada tipo:

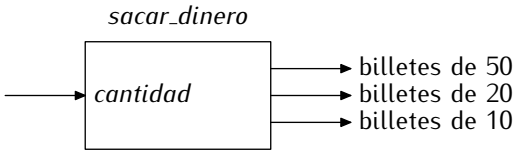
cajero.py

```
1 carga50 = 100
2 carga20 = 100
3 carga10 = 100
```

Diseñaremos ahora una función que, ante una petición de dinero, muestre por pantalla los billetes de cada tipo que se entregan. La función devolverá una lista con el número de billetes de 50, 20 y 10 € si se pudo dar el dinero, y la lista `[0, 0, 0]` en caso contrario. Intentémoslo.

cajero.py

```
1  carga50 = 100
2  carga20 = 100
3  carga10 = 100
4
5  def sacar_dinero(cantidad):
6      de50 = cantidad / 50
7      cantidad = cantidad % 50
8      de20 = cantidad / 20
9      cantidad = cantidad % 20
10     de10 = cantidad / 10
11     return [de50, de20, de10]
```



¿Entiendes las fórmulas utilizadas para calcular el número de billetes de cada tipo? Estúdialas con calma antes de seguir.

En principio, ya está. Bueno, no; hemos de restar los billetes que le damos al usuario de las variables *carga*50, *carga*20 y *carga*10, pues el cajero ya no los tiene disponibles para futuras extracciones de dinero:

cajero_5.py — cajero.py

```
1  carga50 = 100
2  carga20 = 100
3  carga10 = 100
4
5  def sacar_dinero(cantidad):
6      de50 = cantidad / 50
7      cantidad = cantidad % 50
8      de20 = cantidad / 20
9      cantidad = cantidad % 20
10     de10 = cantidad / 10
11     carga50 = carga50 - de50
12     carga20 = carga20 - de20
13     carga10 = carga10 - de10
14     return [de50, de20, de10]
```

Probemos el programa añadiendo, momentáneamente, un programa principal:

cajero_1.py — cajero.py

```
19 c = int(raw_input('Cantidad a extraer: '))
20 print sacar_dinero(c)
```

¿Qué ocurrirá con el acceso a *carga*50, *carga*20 y *carga*10? Puede que Python las tome por variables locales, en cuyo caso, no habremos conseguido el objetivo de actualizar la cantidad de billetes disponibles de cada tipo. Lo que ocurre es peor aún: al ejecutar el programa obtenemos un error.

```
$ python cajero.py↲
Cantidad a extraer: 70
Traceback (most recent call last):
  File "cajero.py", line 17, in ?
    print sacar_dinero(c)
  File "cajero.py", line 11, in sacar_dinero
    carga50 = carga50 - de50
UnboundLocalError: local variable 'carga50' referenced before assignment
```

El error es del tipo *UnboundLocalError* (que podemos traducir por «error de variable local no ligada») y nos indica que hubo un problema al tratar de acceder a *carga*50, pues es una variable *local* que no tiene valor asignado previamente. Pero, ¡*carga*50 debería ser una variable global, no local, y además sí se le asignó un valor: en la línea 1 asignamos a *carga*50 el valor 100! ¿Por qué se confunde? Python utiliza una regla simple para decidir si una variable usada en una función es local o global: si se le asigna un valor, es local; si no, es global. Las variables *carga*50, *carga*20 y *carga*10 aparecen en la parte izquierda de una asignación, así que Python supone que son variables locales. Y si son locales, no están inicializadas cuando se evalúa la parte derecha de la asignación. Hay una forma de evitar que Python se equivoque en situaciones como ésta: declarar explícitamente que esas variables son globales. Fíjate en la línea 6:

cajero_6.py — cajero.py

```
1  carga50 = 100
2  carga20 = 100
3  carga10 = 100
4
5  def sacar_dinero(cantidad):
6      global carga50, carga20, carga10
7      de50 = cantidad / 50
8      cantidad = cantidad % 50
9      de20 = cantidad / 20
10     cantidad = cantidad % 20
11     de10 = cantidad / 10
12     carga50 = carga50 - de50
13     carga20 = carga20 - de20
14     carga10 = carga10 - de10
15     return [de50, de20, de10]
16
17 c = int(raw_input('Cantidad a extraer: '))
18 print sacar_dinero(c)
```

```
$ python cajero.py↲
Cantidad a extraer: 70↲
[1, 1, 0]
```

¡Perfecto! Hagamos una prueba más:

```
$ python cajero.py↲
Cantidad a extraer: 7000↲
[140, 0, 0]
```

¿No ves nada raro? ¡La función ha dicho que nos han de dar 140 billetes de 50 €, cuando sólo hay 100! Hemos de refinar la función y hacer que nos dé la cantidad solicitada sólo cuando dispone de suficiente efectivo:

cajero_7.py — cajero.py

```
1  carga50 = 100
2  carga20 = 100
```

```
3  carga10 = 100
4
5  def sacar_dinero(cantidad):
6      global carga50, carga20, carga10
7      if cantidad <= 50 * carga50 + 20 * carga20 + 10 * carga10:
8          de50 = cantidad / 50
9          cantidad = cantidad % 50
10         de20 = cantidad / 20
11         cantidad = cantidad % 20
12         de10 = cantidad / 10
13         carga50 = carga50 - de50
14         carga20 = carga20 - de20
15         carga10 = carga10 - de10
16         return [de50, de20, de10]
17     else:
18         return [0, 0, 0]
19
20 c = int(raw_input('Cantidad␣a␣extraer:␣'))
21 print sacar_dinero(c)
```

La línea 7 se encarga de averiguar si hay suficiente dinero en el cajero. Si no lo hay, la función finaliza inmediatamente devolviendo la lista `[0, 0, 0]`. ¿Funcionará ahora?

```
$ python cajero.py↲
Cantidad a extraer: 7000↲
[140, 0, 0]
```

¡No! Sigue funcionando mal. ¡Claro!, hay $50 \times 100 + 20 \times 100 + 10 \times 100 = 8000$ € en el cajero y hemos pedido 7000 €. Lo que deberíamos controlar no (sólo) es que haya suficiente dinero, sino que haya suficiente cantidad de billetes de cada tipo:

cajero_8.py — cajero.py

```
1  carga50 = 100
2  carga20 = 100
3  carga10 = 100
4
5  def sacar_dinero(cantidad):
6      global carga50, carga20, carga10
7      if cantidad <= 50 * carga50 + 20 * carga20 + 10 * carga10:
8          de50 = cantidad / 50
9          cantidad = cantidad % 50
10         if de50 >= carga50: # Si no hay suficientes billetes de 50
11             cantidad = cantidad + (de50 - carga50) * 50
12             de50 = carga50
13         de20 = cantidad / 20
14         cantidad = cantidad % 20
15         if de20 >= carga20: # y no hay suficientes billetes de 20
16             cantidad = cantidad + (de20 - carga20) * 20
17             de20 = carga20
18         de10 = cantidad / 10
19         cantidad = cantidad % 10
20         if de10 >= carga10: # y no hay suficientes billetes de 10
21             cantidad = cantidad + (de10 - carga10) * 10
22             de10 = carga10
23         # Si todo ha ido bien, la cantidad que resta por entregar es nula:
24         if cantidad == 0:
25             # Así que hacemos efectiva la extracción
26             carga50 = carga50 - de50
27             carga20 = carga20 - de20
```

```
28          carga10 = carga10 - de10
29          return [de50, de20, de10]
30      else: # Y si no, devolvemos la lista con tres ceros:
31          return [0, 0, 0]
32    else:
33      return [0, 0, 0]
34
35  c = int(raw_input('Cantidad a extraer: '))
36  print sacar_dinero(c)
```

Bueno, parece que ya tenemos la función completa. Hagamos algunas pruebas:

```
$ python cajero.py ↲
Cantidad a extraer: 130 ↲
[2, 1, 1]
$ python cajero.py ↲
Cantidad a extraer: 7000 ↲
[100, 100, 0]
$ python cajero.py ↲
Cantidad a extraer: 9000 ↲
[0, 0, 0]
```

¡Ahora sí!

EJERCICIOS

▶ 358 Hay dos ocasiones en las que se devuelve la lista `[0, 0, 0]`. ¿Puedes modificar el programa para que sólo se devuelva esa lista explícita desde un punto del programa?

Como ya hemos diseñado y probado la función, hagamos un último esfuerzo y acabemos el programa. Eliminamos las líneas de prueba (las dos últimas) y añadimos el siguiente código:

cajero.py — cajero.py

```
⋮
35
36  # Programa principal
37  while 50*carga50 + 20*carga20 + 10*carga10 > 0:
38    peticion = int(raw_input('Cantidad que desea sacar: '))
39    [de50, de20, de10] = sacar_dinero(peticion)
40    if [de50, de20, de10] != [0, 0, 0]:
41      if de50 > 0:
42        print 'Billetes de 50 euros:', de50
43      if de20 > 0:
44        print 'Billetes de 20 euros:', de20
45      if de10 > 0:
46        print 'Billetes de 10 euros:', de10
47      print 'Gracias por usar el cajero.'
48      print
49    else:
50      print 'Lamentamos no poder atender su petición.'
51      print
52  print 'Cajero sin dinero. Avise a mantenimiento.'
```

Usemos esta versión final del programa:

```
$ python cajero.py ↲
Cantidad que desea sacar: 7000 ↲
Billetes de 50 euros: 100
Billetes de 20 euros: 100
```

```
Gracias por usar el cajero.

Cantidad que desea sacar: 500↲
Billetes de 10 euros: 50
Gracias por usar el cajero.

Cantidad que desea sacar: 600↲
Lamentamos no poder atender su petición.

Cantidad que desea sacar: 500↲
Billetes de 10 euros: 50
Gracias por usar el cajero.

Cajero sin dinero. Avise a mantenimiento.
```

Se supone que un cajero de verdad debe entregar dinero

El programa del cajero automático no parece muy útil: se limita a imprimir por pantalla el número de billetes de cada tipo que nos ha de entregar. Se supone que un cajero de verdad debe entregar dinero y no limitarse a mostrar mensajes por pantalla.

Los cajeros automáticos están gobernados por un computador. Las acciones del cajero pueden controlarse por medio de funciones especiales. Estas funciones acceden a *puertos de entrada/salida* del ordenador que se comunican con los periféricos adecuados. El aparato que entrega billetes no es más que eso, un periférico más.

Lo lógico sería disponer de un módulo, digamos *dipensador_de_billetes*, que nos diera acceso a las funciones que controlan el periférico. Una función podría, por ejemplo, entregar al usuario tantos billetes de cierto tipo como se indicara. Si dicha función se llamara *entrega*, en lugar de una sentencia como «**print** `"Billetes␣de␣50␣euros:",` *de*50», realizaríamos la llamada *entrega*`(`*de*50`,` 50`)`.

Acabaremos este apartado con una reflexión. Ten en cuenta que modificar variables globales desde una función no es una práctica de programación recomendable. La experiencia dice que sólo en contadas ocasiones está justificado que una función modifique variables globales. Se dice que modificar variables globales desde una función es un *efecto secundario* de la llamada a la función. Si cada función de un programa largo modificara libremente el valor de variables globables, tu programa sería bastante ilegible y, por tanto, difícil de ampliar o corregir en el futuro.