Conceptos avanzados sobre funciones

Tipos de argumentos: posicionales o keyword

Tenemos dos tipos de parámetros: los posiciónales donde el parámetro real debe coincidir en posición con el parámetro formal:

```
>>> def sumar(n1,n2):
...   return n1+n2
...
>>> sumar(5,7)
12
>>> sumar(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sumar() missing 1 required positional argument: 'n2'
```

Además podemos tener parámetros con valores por defecto:

```
>>> def operar(n1,n2,operador='+',respuesta='El resultado es '):
...   if operador=="+":
...     return respuesta+str(n1+n2)
...   elif operador=="-":
...     return respuesta+str(n1-n2)
...   else:
...     return "Error"
...
>>> operar(5,7)
'El resultado es 12'
>>> operar(5,7,"-")
'El resultado es -2'
>>> operar(5,7,"-","La resta es ")
'La resta es -2'
```

Los parámetros keyword son aquellos donde se indican el nombre del parámetro formal y su valor, por lo tanto no es necesario que tengan la misma posición. Al definir una función o al llamarla, hay que indicar primero los argumentos posicionales y a continuación los argumentos con valor por defecto (keyword).

```
>>> operar(5,7) # dos parámetros posicionales
>>> operar(n1=4,n2=6)   # dos parámetros keyword
>>> operar(4,6,respuesta="La suma es")  # dos parámetros posicionales y uno keyword
>>> operar(4,6,respuesta="La resta es",operador="-")    # dos parámetros posicionales y dos keyword
```

Parámetro *

Un parámetro `*` entre los parámetros formales de una función, nos obliga a indicar los parámetros reales posteriores como keyword:

```
>>> def sumar(n1,n2,*,op="+"):
...   if op=="+":
...     return n1+n2
...   elif op=="-":
...     return n1-n2
...   else:
...     return "error"
...
>>> sumar(2,3)
5
>>> sumar(2,3,"-")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sumar() takes 2 positional arguments but 3 were given
>>> sumar(2,3,op="-")
-1
```

Argumentos arbitrarios (*args y **kwargs)

Para indicar un número indefinido de argumentos posicionales al definir una función, utilizamos el símbolo `*`:

```
>>> def sumar(n,*args):
...   resultado=n
...   for i in args:
...     resultado+=i
...   return resultado
...
>>> sumar(2)
2
>>> sumar(2,3,4)
9
```

Para indicar un número indefinido de argumentos keyword al definir una función, utilizamos el símbolo `**`:

```
>>> def saludar(nombre="pepe",**kwargs):
...   cadena=nombre
...   for valor in kwargs.values():
...    cadena=cadena+" "+valor
...   return "Hola "+cadena
...
>>> saludar()
'Hola pepe'
>>> saludar("juan")
'Hola juan'
>>> saludar(nombre="juan",nombre2="pepe")
'Hola juan pepe'
>>> saludar(nombre="juan",nombre2="pepe",nombre3="maria")
'Hola juan maria pepe'
```

Por lo tanto podríamos tener definiciones de funciones del tipo:

```
>>> def f()
>>> def f(a,b=1)
>>> def f(a,*args,b=1)
>>> def f(*args,b=1)
>>> def f(*args,b=1,*kwargs)
>>> def f(*args,*kwargs)
>>> def f(*args)
>>> def f(*kwargs)
```

Desempaquetar argumentos: pasar listas y diccionarios

En caso contrario es cuando tenemos que pasar parámetros que lo tenemos guardados en una lista o en un diccionario.

Para pasar listas utilizamos el símbolo `*`:

```
>>> lista=[1,2,3]
>>> sumar(*lista)
6
>>> sumar(2,*lista)
8
>>> sumar(2,3,*lista)
11
```

Podemos tener parámetros keyword guardados en un diccionario, para enviar un diccionario utilizamos el símbolo `**`:

```
>>> datos={"nombre":"jose","nombre2":"pepe","nombre3":"maria"}
>>> saludar(**datos)
'Hola jose maria pepe'
```

Devolver múltiples resultados

La instrucción `return` puede devolver cualquier tipo de resultados, por lo tanto es fácil devolver múltiples datos guardados en una lista o en un diccionario. Veamos un ejemplo en que devolvemos los datos en una tupla:

```
>>> def operar(n1,n2):
...   return (n1+n2,n1-n2,n1*n2)

>>> suma,resta,producto = operar(5,2)
>>> suma
7
>>> resta
3
>>> producto
10
```