## How do I write a function with output parameters (call by reference)?

Remember that arguments are passed by assignment in Python. Since assignment just creates references to objects, there's no alias between an argument name in the caller and callee, and so no call-by-reference per se. You can achieve the desired effect in a number of ways.

1. By returning a tuple of the results:

```
def func2(a, b):
    a = 'new-value'        # a and b are local names
    b = b + 1              # assigned to new objects
    return a, b            # return new values

x, y = 'old-value', 99
x, y = func2(x, y)
print(x, y)                # output: new-value 100
```

   This is almost always the clearest solution.

2. By using global variables. This isn't thread-safe, and is not recommended.

3. By passing a mutable (changeable in-place) object:

```
def func1(a):
    a[0] = 'new-value'     # 'a' references a mutable list
    a[1] = a[1] + 1        # changes a shared object

args = ['old-value', 99]
func1(args)
print(args[0], args[1])    # output: new-value 100
```

4. By passing in a dictionary that gets mutated:

```
def func3(args):
    args['a'] = 'new-value'     # args is a mutable dictionary
    args['b'] = args['b'] + 1   # change it in-place

args = {'a': 'old-value', 'b': 99}
func3(args)
print(args['a'], args['b'])
```

5. Or bundle up values in a class instance:

```
class callByRef:
    def __init__(self, **args):
        for (key, value) in args.items():
            setattr(self, key, value)

def func4(args):
```

```
    args.a = 'new-value'         # args is a mutable callByRef
    args.b = args.b + 1          # change object in-place

args = callByRef(a='old-value', b=99)
func4(args)
print(args.a, args.b)
```

There's almost never a good reason to get this complicated.

Your best choice is to return a tuple containing the multiple results.