1. What is the result of the code, and why?

>>> def func(a, b=6, c=8):

print(a, b, c)

>>> func(1, 2)

O/P- 1 2 8 . because 1 and 2 are passed to a and b by position andc is omitted in the call and defaults to 8

2. What is the result of this code, and why?

>>> def func(a, b, c=5):

print(a, b, c)

>>> func(1, c=3, b=2)

O/P 123, 1 is passed to a by position and b and c are passed 2 and3 by name (the left-to right order doesn’t matter when keyword argument are used like this)

3. How about this code: what is its result, and why?

>>> def func(a, \*pargs):

print(a, pargs)

>>> func(1, 2, 3)

**o/p 1(2,3), because 1 is passed to a and the \*pargs collects the remaining positional aruguments into a new tuple object. We can step through the extra positional arguments tuple with any iteration tool.**

4. What does this code print, and why?

>>> def func(a, \*\*kargs):

print(a, kargs)

>>> func(a=1, c=3, b=2)

**o/p 1 {'c': 3, 'b': 2}**

**1 is passed to a by name and the \*\*kargs collects the remaining keyword arguments into a dictionary. We could step through the extra keyword arguments dictionary by key with any iteration.**

5. What gets printed by this, and explain?

>>> def func(a, b, c=8, d=5): print(a, b, c, d)

>>> func(1, \*(5, 6))

**o/p 1 5 6 5 : the matches a by position, 5 and 6 match b and c by \*name positional(6 overrides c’s default) d default because to 4 was not passed a value.**

6. what is the result of this, and explain?

>>> def func(a, b, c): a = 2; b[0] = 'x'; c['a'] = 'y'

>>> l=1; m=[1]; n={'a':0}

>>> func(l, m, n)

>>> l, m, n

**o/p-(1, ['x'], {'a': 'y'})**

**The first assignment in the function doesn’t impact the caller, but the second two do because they change passed in mutable objects in place.**