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

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

print(a, b, c)

>>> func(1, 2)

(1,2,8) because a’s value is taken from its postion in func when called while b’s value is updated from 6 to the value given in its position I.e 2 and c ‘s is already assigned inside the function and not assigned while calling. So c will retained its value .

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)

# (1,2,3) , because a’s value is assigned as 1 when called, b is assigned as 2 when called and c’s value is update from 5 to 3 when called.

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

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

print(a, pargs)

>>> func(1, 2, 3)

1(2,3) because a’s value is assigned as 1 when called while \*pargs can take as much input as possible

4. What does this code print, and why?

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

print(a, kargs)

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

1 {'c': 3, 'b': 2} because a’s values is assigned to 1 when called while \*\* kargs can take as much input of key value pair as possible

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))

1 5 6 5 because a’s value is assigned as 1 when called while b,c is iterated and given value as 5,6 using \*(5,6) and d is given as d in the function

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

(1, ['x'], {'a': 'y'})

The value of l will be equal to 1 because first it will be assigned to 2 as a=2 but then it will be updated to 1 as l=1 . The value of m will be [‘x] because firstly m[0]=[‘x’] means in the list m at “0” index the value will be x. so m=[1] will be updated as m=[‘x’]. The value of n will be {“a”:”y”} because n[‘a’]=’y’ means n is a dictionary and at ‘a’ index the value is ‘y’ and further n={‘a’:0} means at a the value will be updated as y.