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

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

print(a, b, c)

>>> func(1, 2)

Answer: It will print value of a =1 , b=2 and c= 8 as for a we are simply passing value of 1, for b we are passing the value of 2 and are overwriting the value of 6. And c is already defined as 8 in the actual parameters of the function

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)

Answer: Here it will simply print a =1 ,b=2 and c=3 as a has value directly and b ,c have values assigned from function call whatever their order of formal parameter is.

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

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

print(a, pargs)

>>> func(1, 2, 3)

Answer: it will (1,(2,3)) as it stores a=1 and a tuple of values in \*pargs as 2,3.

4. What does this code print, and why?

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

print(a, kargs)

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

Answer: This will print 1,{c:3,b:2} as a will store the value of a and \*\*kwargs stores the value in key:value pair manner like in dictionary.

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))

Answer: This will print a =1, b = 5 , c=6 and again d=5 as a will take 1 directly, b and c will have their value overwritten directly as 5 and 6 and d is given value of 5 in actual parameters of 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

Answer: It will simply print values of a,b and c defined inside the function as we are overwriting the values inside the function if we use a print method inside the func() function but we are not having a print() and so we will not print anything.