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

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

print(a, b, c)

>>> func(1, 2)

**Ans: Output: 1 2 12**

**Paramet b and c have default values assigned to them. If no value is passed onto these variables, the default values i.e. b=6 and c=8 will be printed.**

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)

**Ans: Output: 1 2 3**

**We have called the function with keyword arguments. In this way, we can forget about the position of values as long as we have written the right keyword argument.**

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

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

print(a, pargs)

>>> func(1, 2, 3)

**Ans: Output: 1 (2,3)**

**The first argument passed here is an integer and the second one was a variable length argument. All values after the first one will be assigned to pargs which is a tuple.**

4. What does this code print, and why?

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

print(a, kargs)

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

**Ans: Output: 1 {‘c’:3, ’b’:2}**

**\*\*kargs is a keyword variable length argument that can hold multiple key value pairs passed on to it during a function call. While calling the function, 1 is assigned to a, b and c are assigned to kargs as a key-value pair. That makes kargs a 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))

**Ans: Output: 1 5 6 5**

**The default values for c and d had already been given so if no value is passed to these variables, then their default value will be printed. During the call of function, \*(5,6) will disintegrate the individual tuple to individual values and assign 5 and 6 to c and d respectively.**

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

**Ans: Output: (1, [‘x’], {‘a’:’y’})**

**The above code is an example of calling function by value and reference. In Python, the reference of variable is passed to the called function. The reference of all variables were sent to the func() method and stored in its parameters. When we tried updating the value of each variable, the integer was an immutable value. So any changes to value of a would result in creation of another object and it’s reference will be stored in variable a. However, b and c had references to list and dictionary respectively i.e. mutable in nature. Any changes to these reference would be reflected to the same objects instead of creation of another object. That’s the reason, we were able to change value of m and n.**