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

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

print(a, b, c)

>>> func(1, 2)

A1. Output: 1 2 8

Value for ‘a’ was assigned in the function call and the default value for ‘b’ got replaced with the function call as well.

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)

A2. Output: 1 2 3

Values for all parameters were taken from the function call. If the parameter is mentioned before assigning it a value, then the order of the parameters doesn’t matter.

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

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

print(a, pargs)

>>> func(1, 2, 3)

A3. Output: 1 (2, 3)

Value of a is defined in the in the first argument. Anything else passed into this function will be taken as a part of pargs.

4. What does this code print, and why?

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

print(a, kargs)

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

A4. Output: 1 {'c': 3, 'b': 2}

Value of a is defined in the in the first argument. Rest of the arguments passed will be read as a dictionary as a part of kargs. \*\* is used to pass multiple arguments in dictionary format.

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))

A5. Output: 1 5 6 5

Values for a, b, c are taken from the function call. As we did not assign a value to d during function call, it takes the default 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

A6. (1, ['x'], {'a': 'y'})

Value for ‘l’ comes from the value assignment step before applying the function

Value of m and n comes from the substitution actions performed inside the function.