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

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

print(a, b, c)

>>> func(1, 2)

Output

1 2 8

During calling func() we only passing two arguments but according to definition of function three argument is required , it leads to error if default value of variable is not specified equal 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)

Output

1 2 3

Here during calling of function , only order of argument is different , but it is clearly specified which argument is related to which variable like c comes before b but it was put first during calling .

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

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

print(a, pargs)

>>> func(1, 2, 3)

Output

1 (2, 3)

2,3 in parenthesis because return type of args is tuple.

4. What does this code print, and why?

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

print(a, kargs)

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

Output

1 {'c': 3, 'b': 2}

c and b in key and value pair form because return type of kargs is 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))

Output

1 5 6 5

args passes 5 to b , 6 to c because it works in defined continuity , and there is no argument passed for d during calling , therefore it gets it’s default value 5 .

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

Output

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

l is not changed because changes made inside the function , which will not reflect the actual change in l , because it is outside the scope , m is the list which contain only one element which is referencing to b and it means changes in b also reflect changes in m also , therefore it’s first element value changes from 1 to ‘x’ , n is referencing to c , means changes made in c also reflect changes in n , therefore value of key ‘a’ changes from 0 to ‘y’.