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

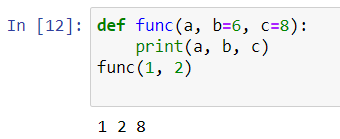
>>> def func(a, b=6, c=8):

print(a, b, c)

>>> func(1, 2)

🡪 The result of the above code will be- 1 2 8.

This is so because in the definition of function func() we have declared default values of variable b and c as 6 and 8 respectively. Thus when we call the function func() and pass the values 1,2 in it the func() will take this value as a = 1 and b = 2, while executing the func() it will print values of a,b,c as 1, 2, 8 respectively as no value for variable c is provided when calling func(), therefore it will take default value as 8.



2. What is the result of this code, and why?

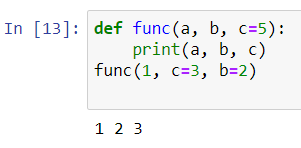
>>> def func(a, b, c=5):

print(a, b, c)

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

🡪 The result of the above code will be- 1 2 3.

This is so because in the definition of function func() we have declared default value of variable c as 5. But when we call the function func() we have passed the values 1, c=3, b=2 in it the func() will take this value as a = 1, c=3 and b = 2, while executing the func() it will print values of a,b,c as 1, 2, 3 respectively as default value for variable c is overwritten while calling func().



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

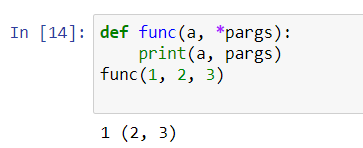
>>> def func(a, \*pargs):

print(a, pargs)

>>> func(1, 2, 3)

🡪 The result of the above code will be- 1 (2, 3).

Because in func() definition we have used \*arg(arguments) which allows us to taken any no of variables as input in function definition, here in the func() definition we are printing a and \*parg. Thus when func() is called and passed 1, 2, 3 as values in it, the result will be 1, (2, 3) where (2, 3) is a tuple due to \*pargs.



4. What does this code print, and why?

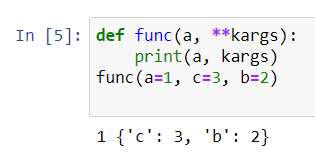
>>> def func(a, \*\*kargs):

print(a, kargs)

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

🡪 The result of the above code will be- 1 {‘c’ : 3, ‘b’: 2}.

Because in func() definition we have used \*\*karg(keyword-arguments) which allows us to taken any no of key-value pair(as in dictionary) variables as input in function definition, here in the func() definition we are printing a and \*\*karg. Thus when func() is called and passed a=1,c=3,b=2 as values in it, the result will be 1 {‘c’ : 3, ‘b’: 2} where {‘c’ : 3, ‘b’: 2) is a dictionary due to \*\*kargs.



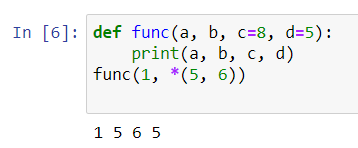
5. What gets printed by this, and explain?

>>> def func(a, b, c=8, d=5): print(a, b, c, d)

>>> func(1, \*(5, 6))

🡪 The result of the above code will be- 1 5 6 5.

Because the function definition has default values set for variable c=8 and d=5. But when the function func(1, \*(5,6)) is called value 1 and a \*(5,6) tuple is passed, so variable a,b,c are assigned values 1,5,6 respectively and the variable d is taking the default value as defined in function definition.



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

🡪 The result of the above code will be- (1, [‘x’], {‘a’: ‘y’}).

This is because inside the function we are mutating the list and dictionary passed through variable m and n respectively. So after the function is called with passing l, m, n as parameters and after this operation when we call the variables l, m, n we find m and n values are changed i.e. m and n are mutated even though l, m, n are global variables (i.e. declared outside the function func()).

