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 8

In this code, we have a function func(a, b=6, c=8) that takes three arguments: a, b, and c, with default values set for b and c. When we call func(1, 2), the value of a is set to 1, b is set to 2 (the positional argument), and c takes its default value of 8. The function then prints these values, resulting in the output mentioned above.

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

In this code, we have a function func(a, b, c=5) that takes three arguments: a, b, and c, with a default value set for c. When we call func(1, c=3, b=2), a is set to 1, b is set to 2 (the keyword argument), and c is set to 3 (the keyword argument). The function then prints these values, resulting in the output mentioned above.

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)

In this code, we have a function func(a, \*pargs) that takes one required argument a and collects any additional positional arguments into a tuple pargs. When we call func(1, 2, 3), a is set to 1, and pargs collects the additional positional arguments (2, 3). The function then prints these values, resulting in the output mentioned above.

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}

In this code, we have a function func(a, \*\*kargs) that takes one required argument a and collects any additional keyword arguments into a dictionary kargs. When we call func(a=1, c=3, b=2), a is set to 1, and kargs collects the additional keyword arguments {'c': 3, 'b': 2}. The function then prints these values, resulting in the output mentioned above.

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

In this code, we have a function func(a, b, c=8, d=5) that takes four arguments: a, b, c, and d, with default values set for c and d. When we call func(1, \*(5, 6)), a is set to 1, b is set to 5 (the first value of the unpacked tuple (5, 6)), c takes its default value of 8, and d is set to 6 (the second value of the unpacked tuple). The function then prints these values, resulting in the output mentioned above.

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 : 2 ['x'] {'a': 'y'}

In this code, we have a function func(a, b, c) that takes three arguments: a, b, and c. When we call func(l, m, n), a is set to l which is 1, b is set to m which is [1], and c is set to n which is {'a': 0}. Inside the function, a is reassigned the value 2, b[0] modifies the list m to ['x'], and c['a'] modifies the dictionary n to {'a': 'y'}. After the function call, l remains 1, m is now ['x'], and n is now {'a': 'y'}. The function itself does not return anything, so the final print statement l, m, n prints the current values of l, m, and n, resulting in the output mentioned above.