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

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

print(a, b, c)

>>> func(1, 2)

Ans1

Output

1, 2, and 8

This is because the function func takes three parameters: a, b, and c. However, the default values for b and c are specified in the function definition, so if those parameters are not passed in when the function is called, they will take on their default values.

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)

Ans2

Output

1, 2, and 3

This is because when calling the function func, the values for parameters a, b, and c are passed in as 1, 2, and 3 respectively.

Even though the parameter c has a default value of 5, its value is explicitly set to 3 when calling the function, so it takes that value instead of its default value. The order of the named arguments does not matter in this case, as they are explicitly assigned values based on their names.

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

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

print(a, pargs)

>>> func(1, 2, 3)

Ans3

Output

1 and (2, 3)

This is because when calling the function func, the first argument is passed in as 1, and the remaining arguments (2 and 3) are collected into a tuple called pargs due to the use of the single asterisk (\*) before the parameter name.

Inside the function, the values of a and pargs are printed, where a is the first argument and pargs is the tuple containing the remaining arguments.

4. What does this code print, and why?

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

print(a, kargs)

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

Ans4

Output

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

This is because when calling the function func, the arguments are passed in as keyword arguments with the syntax key=value. The first argument a is set to 1 using this syntax, and the remaining keyword arguments b and c are collected into a dictionary called kargs due to the use of the double asterisks (\*\*).

Inside the function, the values of a and kargs are printed, where a is the value of the first argument and kargs is the dictionary containing the remaining keyword arguments.

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))

Ans5

Output

1, 5, 6, and 5

This is because when calling the function func, the first argument a is set to 1, the second argument b is set to 5, and the third argument c is set to 6 using the syntax \*(5, 6), which unpacks the tuple (5, 6) into individual arguments.

Since no value is passed for the parameter d, it takes on its default value of 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)

Ans6

Output

TypeError with the message unhashable type: 'list'

This is because the function func takes three arguments a, b, and c, and modifies their values inside the function. Specifically, a is set to 2, the first element of the list b is set to 'x', and the value associated with the key 'a' in the dictionary c is set to 'y'.

Before calling the function, the variables l, m, and n are set to the values 1, [1], and {'a': 0} respectively. When the function is called with these variables as arguments, the following modifications are made:

a is set to 2, but since l is an integer and integers are immutable, the value of l is not affected by this change.

The first element of b (which is the list [1]) is set to 'x', so m is modified to be ['x'].

The value associated with the key 'a' in c is set to 'y', so n is modified to be {'a': 'y'}.