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- result - 1 ,2, 8

This is because when the function func is called with arguments (1, 2), 1 is assigned to a, 2 is assigned to b, and the default value 8 is assigned to c. Therefore, when the function prints the values of a, b, and c, it prints 1, 2, and 8 respectively.

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)

Ans - Result is 1, 2, 3

This is because when calling the function func(1, c=3, b=2), the value 1 is assigned to a, 2 is explicitly assigned to b, and 3 is explicitly assigned to c. Therefore, when the function prints the values of a, b, and c, it prints 1, 2, and 3 respectively.

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

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

print(a, pargs)

>>> func(1, 2, 3)

Ans- Result is 1, (2, 3)

This is because the function **func** has a parameter **a** and a variable-length argument parameter **pargs** defined with **\*pargs**. When the function **func(1, 2, 3)** is called, **1** is assigned to **a**, and the remaining arguments **2** and **3** are packed into a tuple assigned to **pargs**. Therefore, when the function prints the values of **a** and **pargs**, it prints **1** and **(2, 3)** respectively.

4. What does this code print, and why?

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

print(a, kargs)

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

Ans – Result is 1, {'c': 3, 'b': 2}

This is because the function **func** has a parameter **a** and a variable-length keyword argument parameter **kargs** defined with **\*\*kargs**. When the function **func(a=1, c=3, b=2)** is called, **1** is assigned to **a**, and the keyword arguments **c=3** and **b=2** are packed into a dictionary assigned to **kargs**. Therefore, when the function prints the values of **a** and **kargs**, it prints **1** and **{'c': 3, 'b': 2}** respectively.

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 – Result is 1,5,6,5

When the function **func(1, \*(5, 6))** is called, **1** is assigned to **a**, and the tuple **(5, 6)** is unpacked into **b** and **c** because they are positional parameters. Therefore, **5** is assigned to **b** and **6** is assigned to **c**. Since **d** has a default value of **5**, it remains unchanged. Thus, the function prints **1 5 6 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

Ans - The result of the code is: (1, ['x'], {'a': 'y'})

When the function **func(l, m, n)** is called, the values of **l**, **m**, and **n** are passed as arguments to the function. Inside the function, **a** is reassigned to **2**, but since **a** is a local variable, it doesn't affect the value of **l** outside the function.

However, **b** is a mutable object (a list), so when it is modified inside the function (**b[0] = 'x'**), the change affects the original list **m** outside the function.

Similarly, **c** is a mutable object (a dictionary), so when it is modified inside the function (**c['a'] = 'y'**), the change affects the original dictionary **n** outside the function.

Therefore, after calling the function, **l** remains **1**, **m** is modified to **['x']**, and **n** is modified to **{'a': 'y'}**.