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

Explanation: The function func has default values for b and c. When func(1, 2) is called, a is assigned 1, b is assigned 2 (overriding the default), and c retains its default value of 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)

**Ans**:

Output: 1 2 3

Explanation: The function func has default value 5 for c. When calling func(1, c=3, b=2), a is assigned 1, b is assigned 2, and c is explicitly set to 3, overriding its default.

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)

Explanation: The \*pargs syntax allows the function to accept a variable number of positional arguments. In this case, func(1, 2, 3) assigns 1 to a and collects the remaining arguments into a tuple, resulting in (2, 3) for pargs.

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}

Explanation: The \*\*kargs syntax allows the function to accept a variable number of keyword arguments. In this case, func(a=1, c=3, b=2) assigns 1 to a, and the remaining keyword arguments are collected into a dictionary, resulting in {'c': 3, 'b': 2} for 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))

**Ans**:

Output: 1 5 6 5

Explanation: The \*(5, 6) syntax unpacks the tuple (5, 6) into individual arguments. So, func(1, \*(5, 6)) is equivalent to func(1, 5, 6), and values are assigned to a, b, c accordingly. D = 5 as default.

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

Explanation: The function func modifies the arguments a, b, and c. l remains unchanged, m is modified in place (list is mutable), and n is modified by adding a key-value pair. Therefore, print(l, m, n) displays the modified values.