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

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

print(a, b, c)

>>> func(1, 2)

Sol:-

1 2 8

Explanation: In the function definition, `b` and `c` have default values of 6 and 8, respectively. When the function is called with arguments `1` and `2`, `a` is assigned `1`, `b` is assigned `2`, and `c` retains its default value of `8`. The `print` statement in the function then outputs `1`, `2`, and `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)

Sol:-

The output of the code will be: `1 2 3`

This is because in the function call, the values 1 and 2 are assigned to the parameters `a` and `b` respectively, while the value 3 is assigned to the parameter `c` through the keyword argument `c=3`. The order of the keyword arguments does not matter as long as they are properly assigned to their corresponding parameter.

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

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

print(a, pargs)

>>> func(1, 2, 3)

Sol:-

1, (2, 3)

In this example, the function `func` is defined with two parameters: `a` and `\*pargs`. The second parameter, `\*pargs`, uses the `\*` operator to accept an arbitrary number of positional arguments, which will be packed into a tuple.

When `func(1, 2, 3)` is called, `a` is assigned the value 1, and the remaining arguments, 2 and 3, are packed into a tuple and assigned to `pargs`. The print statement then prints out the value of `a` and `pargs`, resulting in `1, (2, 3)`.

4. What does this code print, and why?

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

print(a, kargs)

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

Sol:-

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

In this example, the function `func()` takes one mandatory argument (`a`) and any number of keyword arguments (specified with `\*\*kargs`). When we call the function, we specify the value of `a` with the keyword argument `a=1`, and we also provide two more keyword arguments `c=3` and `b=2`. These additional keyword arguments are packed into a dictionary, which is then printed as `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))

Sol:-

1 5 6 5

Here, the positional arguments 5 and 6 are passed to the function using the `\*` operator. The first passed value is assigned to the variable `b` as it is the second formal argument of the function. The second passed value is assigned to the variable `c` as it is the third formal argument of the function. Since no value was passed for the parameter `d`, the default value of 5 is used. The value of the parameter `a` is not passed using the `\*` operator so it takes its default value 1.

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

Sol:-

(1, ['x'], {'a': 'y'})

Explanation:

The function func() takes three arguments a, b, and c.

When the function is called with func(l, m, n), the variable l is passed as a normal argument, whereas m and n are passed as references.

Inside the function, a is set to 2, which does not affect the original value of l since it is an immutable integer.

The first element of b (which is a list) is changed to 'x'. Since m was passed as a reference, this modification is reflected in the original list m.

The value of the key 'a' in dictionary c is changed to 'y'. Similarly, since n was passed as a reference, this modification is reflected in the original dictionary n.