**Assignment23**

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

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

print(a, b, c)

>>> func(1, 2)

1. The result of the code will be: 1 2 8

The function func is defined with three parameters: a, b, and c, where b and c have default values of 6 and 8, respectively.

When func(1, 2) is called, the arguments are passed to the parameters a and b, respectively. Since the argument for parameter c is not provided, it takes its default value.

So, the function call func(1, 2) prints the values of a, b, and c as 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)

1. The result of the code will be:1 2 3

The function func is defined with three parameters: a, b, and c, where c has a default value of 5.

When func(1, c=3, b=2) is called, the arguments 1, c=3, and b=2 are passed to the parameters a, b, and c, respectively. The arguments are passed by keyword, so the order in which they are specified does not matter.

So, the function call func(1, c=3, b=2) prints the values of a, b, and c as 1, 2, and 3, respectively. The value of c is overridden by the keyword argument c=3.

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

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

print(a, pargs)

>>> func(1, 2, 3)

1. The result is :: 1 (2, 3)

The function func is defined with two parameters: a and \*pargs. The \*pargs parameter is a variable-length argument list, also known as "arbitrary positional arguments" or "args". It allows the function to accept any number of additional positional arguments after the first parameter a.

When func(1, 2, 3) is called, the argument 1 is passed to parameter a, and the remaining arguments 2 and 3 are collected into the tuple pargs because of the \* before pargs.

So, the function call func(1, 2, 3) prints 1 as the value of parameter a, and (2, 3) as the value of pargs, which is a tuple containing the additional positional arguments 2 and 3.

4. What does this code print, and why?

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

print(a, kargs)

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

1. The result will be: 1 {'c': 3, 'b': 2}

The function func is defined with two parameters: a and \*\*kargs. The \*\*kargs parameter is a variable-length keyword argument list, also known as "arbitrary keyword arguments" or "kwargs". It allows the function to accept any number of additional keyword arguments after the first parameter a, which are collected into a dictionary.

When func(a=1, c=3, b=2) is called, the argument 1 is passed to parameter a, and the keyword arguments c=3 and b=2 are collected into the dictionary kargs.

So, the function call func(a=1, c=3, b=2) prints 1 as the value of parameter a, and {'c': 3, 'b': 2} as the value of kargs, which is a dictionary containing the additional keyword arguments c=3 and b=2.

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))

The result of the code will be:

Copy code

1 5 6 5

Explanation:

The function func is defined with four parameters: a, b, c, and d, where c has a default value of 8 and d has a default value of 5.

When func(1, \*(5, 6)) is called, the arguments 1, 5, and 6 are passed to the parameters a, b, and c, respectively. Since there are no more arguments to fill in for d, it takes its default value.

So, the function call func(1, \*(5, 6)) prints the values of a, b, c, and d as 1, 5, 6, and 5, respectively.

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

1. The result of the code will be:

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

Explanation:

Three variables l, m, and n are initialized with values 1, [1], and {'a': 0} respectively.

The function func is defined with three parameters a, b, and c.

Inside func, the value of a is set to 2, b[0] (the first element of list m) is set to 'x', and c['a'] (the value associated with key 'a' in dictionary n) is set to 'y'.

When func(l, m, n) is called, the variables l, m, and n are passed as arguments to the parameters a, b, and c respectively.

Inside func, the operations a = 2, b[0] = 'x', and c['a'] = 'y' modify the values of the passed variables l, m, and n indirectly.

After the function call, the values of l, m, and n are printed. l remains unchanged (1), m is modified with its first element replaced with 'x', and n is modified with its value associated with key 'a' changed to 'y'. Therefore, the output is (1, ['x'], {'a': 'y'}).