**ADITYA AMIN ASSIGN : 23**

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

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

print(a, b, c)

>>> func(1, 2)

there is an indentation error in the code, as the print() statement is not indented properly, causing a syntax error. Assuming its indented properly we will get output as : 1 2 8.

Explanation:

func() is called with arguments 1 and 2.

The value 1 is passed to a, and 2 is passed to b.

Since c does not have a corresponding argument passed during the function call, it takes the default value of 8 as defined in the function definition.

The print() statement inside the function body prints the values of a, b, and c, which are 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)

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

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

print(a, pargs)

>>> func(1, 2, 3)

First function will give…

1 2 3

Explanation:

func() is called with arguments 1, c=3, and b=2.

The value 1 is passed to a, 3 is passed to c, and 2 is passed to b.

The print() statement inside the function body prints the values of a, b, and c, which are 1, 2, and 3 respectively.

Second function will give…

1 (2, 3)

Explanation:

func() is called with arguments 1, 2, and 3.

1 is passed to a, and 2 and 3 are packed into a tuple and passed to \*pargs as variable-length arguments.

The print() statement inside the function body prints the values of a and pargs, where pargs is a tuple containing 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)

It will give following output…

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

Explanation:

func() is called with keyword arguments a=1, c=3, and b=2.

The value 1 is passed to a.

The remaining keyword arguments c=3 and b=2 are packed into a dictionary and passed to \*\*kargs as keyword arguments.

The print() statement inside the function body prints the values of a and kargs, where kargs is a dictionary containing the keyword arguments and their corresponding values {'c': 3, '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))

This will give following Output :

1 5 6 5

Explanation:

func() is called with positional arguments 1 and \*(5, 6).

1 is passed to a.

\*(5, 6) unpacks the tuple (5, 6) into positional arguments 5 and 6, which are passed to b and c respectively.

Since c has a default value of 8 and is not provided as an argument, the default value 8 is used.

Since d has a default value of 5 and is not provided as an argument, the default value 5 is used.

The print() statement inside the function body prints the values of a, b, c, and d, which are 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

This will generate following output :

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

Explanation:

func() is called with arguments l, m, and n which have values 1, [1], and {'a': 0} respectively.

Inside the function body, the local variable a is assigned the value 2, but this does not affect the value of l because integers are immutable in Python, and changes to local variables do not affect the original objects.

The list b is modified by setting its first element to 'x' using the index 0. Since lists are mutable in Python, this change affects the original list m as well, and now m becomes ['x'].

The dictionary c is modified by setting the value of its key 'a' to 'y'. Since dictionaries are mutable in Python, this change affects the original dictionary n as well, and now n becomes {'a': 'y'}.

Finally, after the function call, the print() statement prints the values of l, m, and n, which are 1, ['x'], and {'a': 'y'} respectively, reflecting the changes made inside the function.