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 : The code provided defines a function func() with three parameters: a, b, and c. The parameters b and c have default values of 6 and 8, respectively.

When the function is called using **func(1, 2)**, the value **1** is passed for the parameter **a**, and the value **2** is passed for the parameter **b**. Since no value is explicitly passed for the parameter **c**, it takes its default value of **8**.

The **print(a, b, c)** statement inside the function will print the values of **a**, **b**, and **c**. In this case, it will output:

1 2 8

The value of **a** is **1** because it is assigned the value passed as the first argument **1**. The value of **b** is **2** because it is assigned the value passed as the second argument **2**. The value of **c** is **8** because it takes its default value since no argument was passed for it.

Therefore, the result of the code will be:

1 2 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 : The code provided defines a function func() with three parameters: a, b, and c. The parameter c has a default value of 5.

When the function is called using **func(1, c=3, b=2)**, the value **1** is passed for the parameter **a**, the value **2** is passed for the parameter **b**, and the value **3** is passed for the parameter **c** explicitly using keyword argument syntax.

The **print(a, b, c)** statement inside the function will print the values of **a**, **b**, and **c**. In this case, it will output:

1 2 3

The value of **a** is **1** because it is assigned the value passed as the first argument **1**. The value of **b** is **2** because it is explicitly passed as the keyword argument **b=2**. The value of **c** is **3** because it is explicitly passed as the keyword argument **c=3**, overriding its default value of **5**.

Therefore, the result of the code will be:

1 2 3

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

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

print(a, pargs)

>>> func(1, 2, 3)

Ans : The code provided defines a function func() with a parameter a and \*pargs, which is a variable-length parameter or "args" parameter. This parameter allows the function to accept any number of positional arguments, which will be packed into a tuple called pargs.

When the function is called using **func(1, 2, 3)**, the value **1** is passed for the parameter **a**, and the values **2** and **3** are passed as additional positional arguments.

The **print(a, pargs)** statement inside the function will print the value of **a** and the tuple **pargs**, which contains the additional positional arguments. In this case, it will output:

1 (2, 3)

The value of **a** is **1** because it is assigned the value passed as the first argument **1**. The **pargs** tuple contains the additional positional arguments **2** and **3**.

Therefore, the result of the code will be:

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)

Ans : The code provided defines a function func() with a parameter a and \*\*kargs, which is a variable-length parameter or "kwargs" parameter. This parameter allows the function to accept any number of keyword arguments, which will be packed into a dictionary called kargs.

When the function is called using **func(a=1, c=3, b=2)**, the value **1** is passed for the parameter **a**, and the keyword arguments **c=3** and **b=2** are passed.

The **print(a, kargs)** statement inside the function will print the value of **a** and the dictionary **kargs**, which contains the keyword arguments. In this case, it will output:

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

The value of **a** is **1** because it is explicitly passed as the keyword argument **a=1**. The **kargs** dictionary contains the keyword arguments **c=3** and **b=2**, where the keys are the parameter names and the values are the respective argument values.

Therefore, the result of the code will be:

1 {'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))

Ans : The code you provided defines a function func() with four parameters: a, b, c, and d. The parameters c and d have default values of 8 and 5, respectively.

When the function is called using **func(1, \*(5, 6))**, the value **1** is passed for the parameter **a**, and the tuple **(5, 6)** is unpacked using the **\*** operator to provide the values for the parameters **b** and **c**, respectively.

The **print(a, b, c, d)** statement inside the function will print the values of **a**, **b**, **c**, and **d**. In this case, it will output:

1 5 6 5

The value of **a** is **1** because it is assigned the value passed as the first argument **1**. The value of **b** is **5** because it is assigned the first element of the tuple **(5, 6)**. The value of **c** is **6** because it is assigned the second element of the tuple **(5, 6)**. The value of **d** is **5** because it takes its default value of **5** since no argument was passed for it.

Therefore, the result of the code will be:

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 code you provided defines a function func() that takes three parameters: a, b, and c. Inside the function, it assigns new values to a, modifies the first element of b, and adds a new key-value pair to the dictionary c.

After defining the function, the code assigns values to variables **l**, **m**, and **n**, which are then passed as arguments to the **func()** function when it is called.

Here is a step-by-step breakdown of the code execution:

1. **l=1** assigns the value **1** to the variable **l**.
2. **m=[1]** assigns a list **[1]** to the variable **m**.
3. **n={'a':0}** assigns a dictionary **{'a': 0}** to the variable **n**.
4. **func(l, m, n)** calls the **func()** function with the values of **l**, **m**, and **n** as arguments.
   * Inside the function, **a** is assigned the value **2**, which doesn't affect the original value of **l**.
   * **b[0] = 'x'** modifies the first element of the list **m** to **'x'**.
   * **c['a'] = 'y'** adds a new key-value pair **'a': 'y'** to the dictionary **n**.
5. Finally, **l, m, n** is printed, which outputs the current values of **l**, **m**, and **n** after the function call.

Therefore, the result of the code will be:

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

The value of **l** remains **1** because **a = 2** inside the function doesn't affect the original value of **l**. The value of **m** becomes **['x']** because the first element of the list **m** is modified inside the function. The value of **n** becomes **{'a': 'y'}** because a new key-value pair **'a': 'y'** is added to the dictionary **n** inside the function.