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

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

**print(a, b, c)**

**>>> func(1, 2)**

The result of the code will be:

1 2 8

In this code, a function func is defined that takes three arguments: a, b, and c. b and c have default values of 6 and 8, respectively.

When the func function is called using func(1, 2), only two arguments are passed, 1 and 2. The first argument, 1, is assigned to a, and 2 is assigned to b. The value of c remains at its default value of 8.

The print function is then called with the values of a, b, and c, which prints 1 2 8 to the console.

This code demonstrates how default values can be used in function definitions to provide default values for arguments that are not passed. If an argument with a default value is not passed, the default value will be used instead.

**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)**

The result of the code will be:

1 2 3

In this code, a function func is defined that takes three arguments: a, b, and c. c has a default value of 5.

When the func function is called using func(1, c=3, b=2), the values 1, 2, and 3 are passed as arguments, but they are assigned using keyword arguments rather than positional arguments. This means that the values are assigned based on the argument names, rather than their position in the argument list.

As a result, 1 is assigned to a, 2 is assigned to b, and 3 is assigned to c. The print function is then called with the values of a, b, and c, which prints 1 2 3 to the console.

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

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

**print(a, pargs)**

**>>> func(1, 2, 3)**

The output of this code will be:

1 (2, 3)

Explanation: The "\*pargs" syntax in the function definition specifies that any additional positional arguments passed to the function will be collected into a tuple and assigned to the variable "pargs". In this case, when calling "func(1, 2, 3)" the value 1 is assigned to the variable "a", and the values 2 and 3 are collected into a tuple and assigned to the variable "pargs". Thus, when printing "a" and "pargs", we get "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)**

The code will print 1 {'c': 3, 'b': 2}.

The func function takes one mandatory argument a, and two keyword arguments \*\*kargs. The \*\*kargs syntax allows a function to take an arbitrary number of keyword arguments, which are passed to the function as a dictionary.

In this case, the function is called with the values a=1, c=3, b=2, so a takes the value 1, and kargs takes the value {'c': 3, 'b': 2}. The function then prints the values of a and 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))**

The code will print 1 {'c': 3, 'b': 2}.

The func function takes one mandatory argument a, and two keyword arguments \*\*kargs. The \*\*kargs syntax allows a function to take an arbitrary number of keyword arguments, which are passed to the function as a dictionary.

In this case, the function is called with the values a=1, c=3, b=2, so a takes the value 1, and kargs tThis code will print 1 5 6 8.

The function func takes four arguments a, b, c, and d, with c and d having default values of 8 and 5, respectively.

In the call to func, we pass 1 as the first argument a and (5, 6) as the second argument, which is unpacked as b and c. The value of d is still 5 as specified in the default value.

So the final output will be 1 5 6 8.akes the value {'c': 3, 'b': 2}. The function then prints the values of a and kargs.

**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**

The output of this code will be (1, ['x'], {'a': 'y'}).

In the function func, the argument a is assigned a new value of 2, but this does not affect the value of l outside the function. The argument b is a mutable object (a list), so when its first element is modified to 'x', this change is reflected outside the function as well. The same is true for the argument c, which is a dictionary, and its key 'a' is set to 'y'.

Therefore, after the function is called, l, m, and n are equal to 1, ['x'], and {'a': 'y'}, respectively.