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

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

print(a, b, c)

>>> func(1, 2)

Answer: 1 2 8

The function assigns the new values by replacing them.Here c takes as default value.

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)

Answer: 1 2 3

The function assigns the new values by replacing them.Here a,b and c takes as given value.

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

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

print(a, pargs)

>>> func(1, 2, 3)

Answer: 1(2,3)

As a is exluded in the function command it combines (2,3) to add more flexibility to functions.

4. What does this code print, and why?

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

print(a, kargs)

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

Answer: 1 {'c': 3, 'b': 2}  
This is because the function `func` takes an argument `a` and any additional keyword arguments (\*\*kargs). When calling the function with `func(a=1, c=3, b=2)`, `a` is assigned the value 1 and the keyword arguments `c` and `b` are assigned the values 3 and 2, respectively. The function then prints the value of `a` and the keyword arguments `kargs`, which is a dictionary containing the additional keyword arguments passed to the function.

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))

Answer: 1 5 6 5

It generates the output as \* command is used and function assigns the values.

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

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

m[1] is ['x'] here it is taking b[0] = 'x' values in the second position that is 1.