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 output will be 1 2 8. The function func() is called with two arguments, so the first two parameters a and b take the values of these arguments, and c takes its 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)
Ans:- The output will be 1 2 3. The function func() is called with keyword arguments, which are matched with the corresponding parameters in the function definition.

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

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

print(a, pargs)

>>> func(1, 2, 3)

Ans:- The output will be 1 (2, 3). The *pargs parameter in the function definition collects additional positional arguments into a tuple.

4. What does this code print, and why?

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

print(a, kargs)

>>> func(a=1, c=3, b=2)
Ans:- The output will be 1 {'c': 3, 'b': 2}. The **kargs parameter in the function definition collects additional keyword arguments into a dictionary.

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 output will be 1 5 6 5. The function func() is called with an argument and a tuple of arguments. The tuple is unpacked and its elements are passed as additional arguments to the function.

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 output will be (1, ['x'], {'a': 'y'}). In the function func(), the parameters a, b, and c are assigned new values. However, since a is an immutable integer object and b and c are mutable list and dictionary objects respectively, only b and c are changed in the caller’s scope.