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 : 1 2 8

a = 1 (first argument)

b = 2 (second argument)

c = 8 (default value, as no third argument is given)

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 : 1 2 3

a = 1 (first positional argument)

b = 2 (keyword argument b=2)

c = 3 (keyword argument c=3)

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

def func(a, \*pargs):

print(a, pargs)

func(1, 2, 3)

Ans : 1 (2, 3)

a = 1 (first positional argument)

\*pargs = (2, 3) (remaining positional arguments collected 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 : 1 {'c': 3, 'b': 2}

a = 1 (from the keyword argument a=1).

kargs = {'c': 3, 'b': 2} (a dictionary containing keyword arguments c=3 and 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 : 1 5 6 5

a receives the first positional argument 1.

b receives the unpacked value 5 from the tuple (5, 6).

c receives the unpacked value 6 from the tuple (5, 6), overriding its default value of 8.

d retains its default value of 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

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

l, which is passed as a, remains unchanged outside the function because integers are immutable in Python.

m, which is passed as b, is a list, and lists are mutable. When b[0] = 'x' is executed inside func, it modifies the list m itself (not a copy of it). Therefore, when you print m after calling func, you see the modified list [x].

n, which is passed as c, is a dictionary, and dictionaries are mutable. When c['a'] = 'y' is executed inside func, it adds a new key-value pair 'a': 'y' to the dictionary n. Therefore, when you print n after calling func, you see {'a': 'y'}.