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: This funtion is taking a positional argument and 2 keyward argument. When function call m=is made, parameter passed

are a=1,b=2. When the function is executed , parameter c=8 will be taken by default as its a keyword argument.

solurion is = 1,2,8

In [5]:

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

print(a, b, c)

func(1, 2)

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: When we make function call, order will be positional argument and then keywords arguments. we can pass the keyword arguments in any order we want.

Solution is 1,2,3

In [6]:

**def** func(a, b, c**=**5):

print(a, b, c)

func(1, c**=**3, b**=**2)

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 return type of \*args parameter is tuple, where as \*\*kargs will be dictionary

solution is = 1,(2,3)

In [7]:

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

print(a, pargs)

func(1, 2, 3)

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 return type of \*\*kargs is dictionary

solution is = 1,{'c':3,'b':2}

In [8]:

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

print(a, kargs)

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

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: '\*' is the unpacking operator and are operators that unpack the values from iterable objects in Python. The single

asterisk operator \* can be used on any iterable that Python provides, while the double asterisk operator \*\* can only

be used on dictionaries. In the example the value \*(5,6) will be unpacked and will be assigned to b and c and passed

as arguments, d =5 will taken by defaults are keyword arguments.

Solution 1,5,6,5

In [18]:

**def** func(a, b, c**=**8, d**=**5):

print(a, b, c, d)

func(1, **\***(5, 6))

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: Here in the code, the list and dict are passed as argument, and those are mutable. Here the list l and parametr b point

to the same list in the memeory location where as dict n and c point to the same memory location. Any updates to this

list will update in the memory location

l = 1 , integer values, immutable, m is list, mutable, n is dict, mutable.

output will be = 1,['x'],{'a':'y'}

In [27]:

**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

Out[27]:

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

In [28]:

*## similar example*

**def** func1(x, l1):

x **=** 5

l1**.**append("nonsense")

y **=** 10

list1 **=** ["meaning"]

func1(y, list1)

print(y)

print(list1)

10

['meaning', 'nonsense']