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

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

print(a, b, c)

>>> func(1, 2)

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)

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

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

print(a, pargs)

>>> func(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)

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))

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

Solutions :

#### 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 result of the above code is 1 2 8. its because the function uses the default value of c ie 8 which is provided at the time of declaration

In [1]:

**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:** The result of the above code is 1 2 3. it is because the function will use default values only when a value for a argument is not provided and if argument name is mentioned while doing a function call, the order of arguments is also ignored by the python interpreter

In [2]:

**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 result of the code is 1 (2,3). \*pargs stands for variable length arguments. this format is used when we are not sure about the no of arguments to be passed to a function. all the values under this argument will be stored in a tuple.

In [3]:

**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 result of the above code is 1 {'c': 3, 'b': 2}. \*\*args stands for variable length keyword arguments. this format is used when we want pass key value pairs as input to a function. All these key value pairs will be stored in a dictionary

In [4]:

**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:** The output of the above is 1 5 6 5. This reason for this function not throwing an error is because, this function expects 4 arguments. the value for a is provided explicitly whereas for arguments b and c, the function will expand the \*(5,6) and consider the value of b as 5 and value of c as 6. since the default value of d is provided in function declaration d value will be 5. However it is recommended to use the feature of positional arguments at the end.

In [5]:

**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:** The output of above code is 1, ['x'], {'a': 'y'}.

1. Eventhough Python gives importance to indentation. its provides a facility to declare an entire function in one single line. where statements in a function body are sepereated by ;
2. When l,m,n are provided as inputs to the function. its modifies the values of l,m,n and sets the value of l=2 ,m=['x'] and n={'a':'y'}

In [6]:

**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[6]:

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