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

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

print(a, b, c)

>>> func(1, 2)

**Output:** 1 2 8

**Reason:** Python allows function arguments to have default values. If the function is called without the argument, the argument gets its default value.

Here, c=8 is a default parameter. If the value of default parameter is not passed while function calling, c will take the value of 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)

**Output:**  1 2 3

**Reason:** Order of parameter in python function calling is not important.

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

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

print(a, pargs)

>>> func(1, 2, 3)

**Output:** 1, (2, 3)

**Reason:** The special syntax \***args** in function definitions in **python is** used to pass a variable number of arguments to a function. It **is** used to pass a non-key worded, variable-length argument list. The syntax **is** to use the symbol \* to take in a variable number of arguments; by convention, it **is** often used with the word **args**.

1 is a positional argument, so will be assigned to x  
2, 3 are arbitrary arguments will be assigned to args tuple

4. What does this code print, and why?

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

print(a, kargs)

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

**Output:** 1 { ‘c’ : 3, ‘b’ : 2}

**Reason:** \*\*kwargs is a dictionary of keyword arguments. The \*\* allows us to pass any number of keyword arguments. A keyword argument is basically 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))

**Output:** 1 5 6 8

**Reason:** The 1 matches x by position,  
5 and 6 match y and z by \*name positional (6 overrides z’s default),  
k defaults to 4 because it was not passed a value.

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'})

**Reason:** Initially, l =1 (integer variable), m=[] (empty list), n={‘a’: 0} (dictionary).

Inside the function func(a,b,c);

l (alias name (a){a =2}), m (alias name (b){b[0]=’x’}), n (alias name (c){c[‘a’]=0}).

However while printing ‘l’ outside the function, it retains its original value unlike dictionary and list.

In **pass by value** the function is provided with a copy of the argument object **passed** to it by the caller.

That means the original object stays intact and all changes made are to a copy of the same and stored at different memory locations.

However for dictionary or list no copy is made.