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 line func() takes two values, that is 1 and 2 respectively, but the func() is defined to take 3 values, as in func() we pass only two values, the third value comes from the default value, that is c = 8.

```
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 line func() takes three values, that is 1 , c= 3 snd b = 2 respectively,the func() is defined to take 3 values, as in func() we pass othree values, the default values gets overwritten, and we get the newly passed values.

Ans :

```
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 special syntax *pargs 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.

What *pargs allows you to do is take in more arguments than the number of formal arguments that you previously defined. With* pargs, any number of extra arguments can be tacked on to your current formal parameters (including zero extra arguments).

Ans:

```
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 special syntax **kargs in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name kwargs with the double star. The reason is because the double star allows us to pass through keyword arguments (and any number of them).

Ans:

```
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:

```
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:

]:

```
def func(a, b, c):
    a = 2; b[0] = 'x'; c['a'] = 'y'
l=1; m=[1]; n={'a':0}
func(l, m, n)
```