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

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

print(a, b, c)

>>> func(1, 2)

```
Result is 1 2 8, this will override the default assigned values.
```

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)

```
Result is 1 2 3, this will override the new argument values.
```

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

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

print(a, pargs)

>>> func(1, 2, 3)

```
Result is 1 (2 3), first value is assigned to a and later values will be assigned as tuple.
```

4. What does this code print, and why?

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

print(a, kargs)

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

```
Result is 1 {'c':3,'b':2}, first value is assigned to a and later values will be assigned as 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))

```
The results is - 1 5 6 5, it assigns sequentially
irrespective  of * and brackets
```

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

```
1 ['x'] {'a': ‘y’},
# Ans. Here in the code, the list and dict are passed as
argument, and those are mutable. Here the list l and
parameter b point
#to the same list in the memory 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.
```