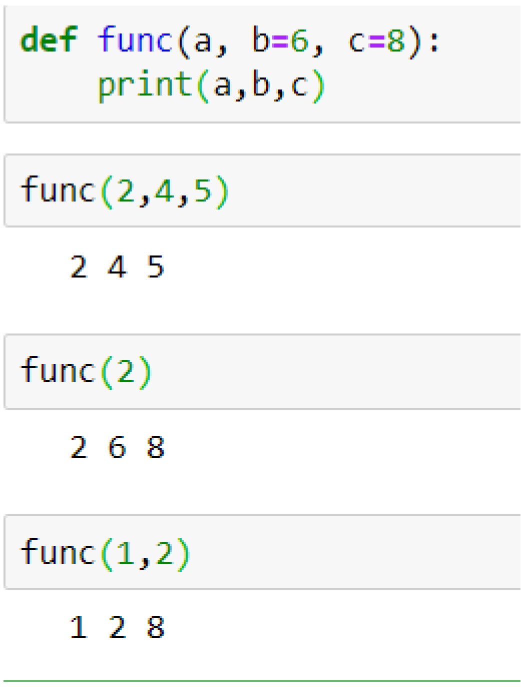
1. What is the result of the code, and why?

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

print(a, b, c)

>>> func(1, 2)



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

```
func(2,4,5)
```

```
2 4 5
```

```
func(2)
```

```
2 6 8
```

```
func(1,2)
```

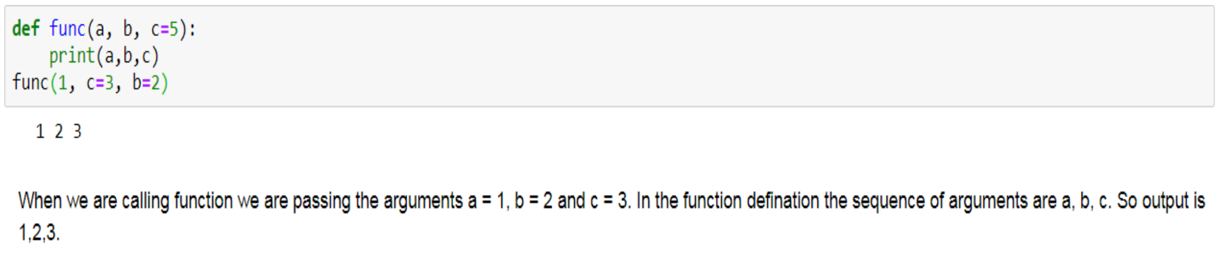
```
1 2 8
```

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)



```
def func(a, b, c=5):
    print(a,b,c)
func(1, c=3, b=2)
```

```
1 2 3
```

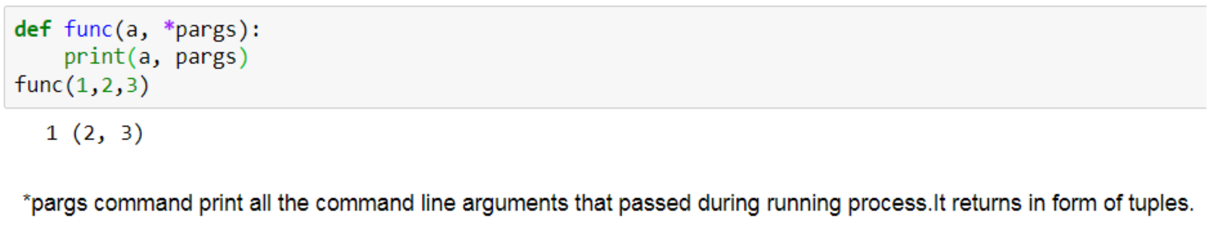
When we are calling function we are passing the arguments a = 1, b = 2 and c = 3. In the function defination the sequence of arguments are a, b, c. So output is 1,2,3.

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

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

print(a, pargs)

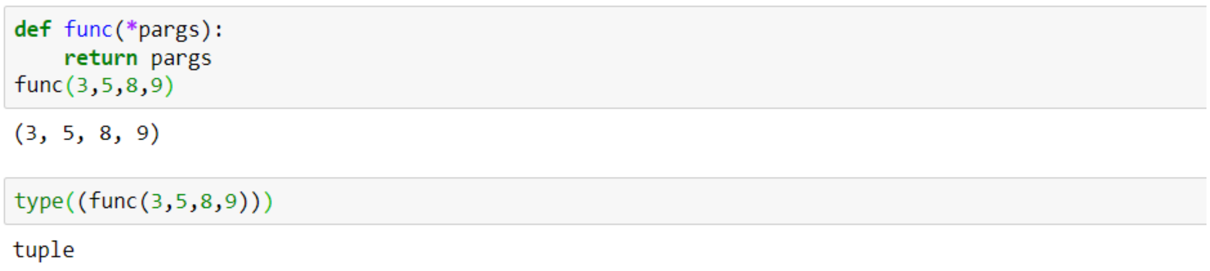
>>> func(1, 2, 3)



```
def func(a, *pargs):
    print(a, pargs)
func(1,2,3)
```

```
1 (2, 3)
```

*pargs command print all the command line arguments that passed during running process.It returns in form of tuples.



```
def func(*pargs):
    return pargs
func(3,5,8,9)
```

```
(3, 5, 8, 9)
```

```
type((func(3,5,8,9)))
```

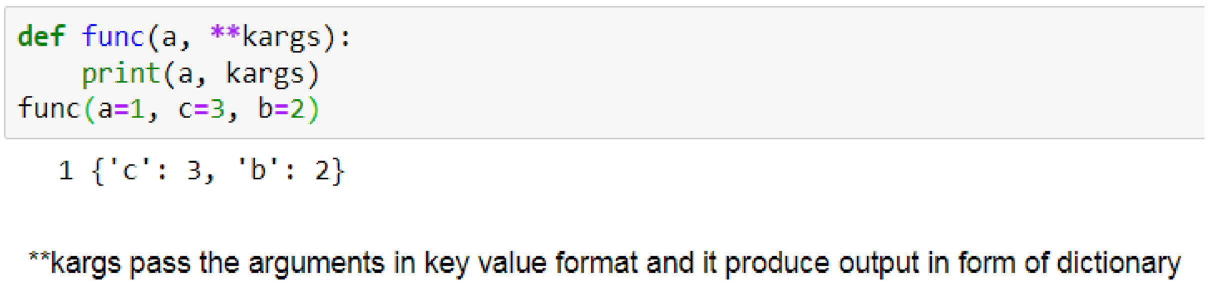
```
tuple
```

4. What does this code print, and why?

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

print(a, kargs)

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



```
def func(a, **kargs):
    print(a, kargs)
func(a=1, c=3, b=2)
```

```
1 {'c': 3, 'b': 2}
```

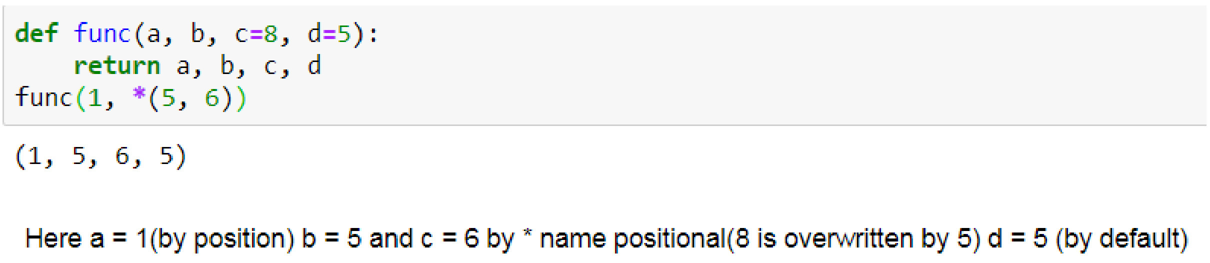
**kargs pass the arguments in key value format and it produce output in form of dictionary

---

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))



```
def func(a, b, c=8, d=5):
    return a, b, c, d
func(1, *(5, 6))
```

```
(1, 5, 6, 5)
```

Here a = 1(by position) b = 5 and c = 6 by * name positional(8 is overwritten by 5) d = 5 (by default)

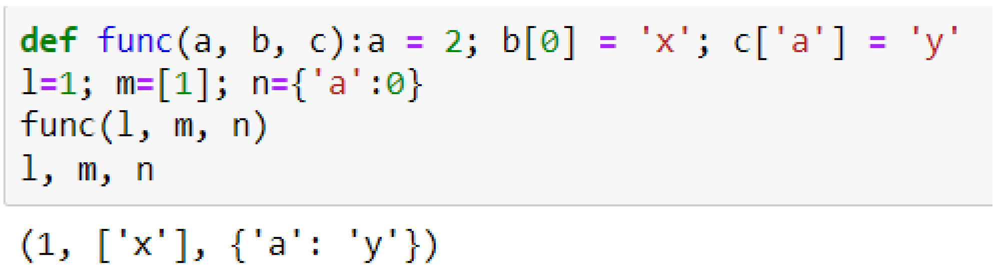
6. what is the result of this, and explain?

>>> 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
```



```
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'})
```

l=1, m = [1] = [b[0]] = ['x'], n = {'a' : 0} = {'a' : 'y'}