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 1 2 8 is output as 1,2 are passed to a and b and c uses default value of 8 in function call

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 1 2 3 is output 1 is passed to a , 2 is passed to b and 3 is passed to c in function call

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

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

Ans 1 (2,3) is output because a is 1, and *pargs contains the tuple (2, 3) representing the additional positional arguments passed to the function

4. What does this code print, and why?

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

Ans 1 {'c': 3, 'b': 2} is output because a is 1, and **kargs contains a dictionary with the keyword arguments and their corresponding values.

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 1 5 6 5 is printed as 1 is passed to a and remaining b,c uses values 5,6 which is passed in tuple and d uses default value of 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 (1, ['x'], {'a': 'y'}) is output because In the function call func(l, m, n), the parameter a is a local variable inside the function, so modifying it doesn't affect the value of l. The parameter b is a list reference, so changes made to it will be reflected outside the function. The parameter c is a dictionary reference, so changes to it will also be visible outside the function.