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

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

print(a, b, c)

>>> func(1, 2)

The result of the code will be "1 2 8". This is because the function `func()` is defined with three parameters `a`, `b`, and `c`, where `b` and `c` have default values of 6 and 8, respectively. When the function is called with the arguments `1` and `2`, the value 1 is assigned to `a`, 2 is assigned to `b`, overriding the default value, and `c` retains its default value of 8. The function then prints the values of `a`, `b`, and `c`, resulting in "1 2 8" being printed.

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)

The result of the code will be "1 2 3". In this case, the function `func()` is defined with three parameters `a`, `b`, and `c`, where `c` has a default value of 5. When the function is called with the arguments `1`, `c=3`, and `b=2`, the value 1 is assigned to `a`, 2 is assigned to `b`, and 3 is assigned to `c`, overriding the default value. The function then prints the values of `a`, `b`, and `c`, resulting in "1 2 3" being printed.

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

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

print(a, pargs)

>>> func(1, 2, 3)

The result of the code will be "1 (2, 3)". Here, the function `func()` is defined with the parameter `a` and `*pargs`. The `*pargs` parameter allows the function to accept a variable number of positional arguments. When the function is called with the arguments `1, 2, 3`, the value 1 is assigned to `a`, and the remaining arguments 2 and 3 are collected into the tuple `pargs`. The function then prints the values of `a` and `pargs`, resulting in "1 (2, 3)" being printed.

4. What does this code print, and why?

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

print(a, kargs)

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

The code will print "1 {'c': 3, 'b': 2}". In this case, the function `func()` is defined with the parameter `a` and `**kargs`. The `**kargs` parameter allows the function to accept a variable number of keyword arguments. When the function is called with the arguments `a=1, c=3, b=2`, the value 1 is assigned to `a`, and the keyword arguments `c=3` and `b=2` are collected into the dictionary `kargs`. The function then prints the values of `a` and `kargs`, resulting in "1 {'c': 3, 'b': 2}" being printed.

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 code will print "1 5 6 5". In this scenario, the function `func()` is defined with four parameters `a`, `b`, `c`, and `d`. When the function is called with the arguments `1` and `*(5, 6)`, the value 1 is assigned to `a`, and the tuple `(5, 6)` is unpacked and assigned to `b` and `c`. The default value of `d`, which is 5, is retained. The function then prints the values of `a`, `b`, `c`, and `d`, resulting in "1 5 6 5" being printed.

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
```

The result of this code will not print anything explicitly, but it will modify the objects `l`, `m`, and `n`. In the function `func()`, the parameter `a` is assigned a new value of 2. The first element of the list `m` is modified to contain the value "x", and the value associated with the key `'a'` in the dictionary `n` is changed to "y". After calling the function with the arguments `l`, `m`, and `n`, the values of `l`, `m`, and `n` will reflect these modifications.