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

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

print(a, b, c)

>>> func(1, 2)

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)

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

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

print(a, pargs)

>>> func(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)

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))

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.

```

1 2 8

```

In this function call, `func(1, 2)`, the argument `a` is assigned the value 1, `b` is assigned the value 2, and `c` retains its default value, which is 8. Since only two arguments are provided, `b` takes the second provided value, and `c` retains its default value.

2.

```

1 2 3

```

In this function call, `func(1, c=3, b=2)`, all arguments are explicitly provided. So, `a` is assigned the value 1, `b` is assigned the value 2 (overriding the default value), and `c` is assigned the value 3.

3.

```

1 (2, 3)

```

This function definition has a parameter `\*pargs`, which collects any additional positional arguments into a tuple. When calling `func(1, 2, 3)`, `a` is assigned 1, and the remaining arguments (2 and 3) are collected into `pargs` as a tuple.

4.

```

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

In this function call, `func(a=1, c=3, b=2)`, `a` is explicitly assigned the value 1. The remaining keyword arguments (`b=2` and `c=3`) are collected into `kargs` as a dictionary.

5.

```

1 5 6 8 5

```

Here, `\*(5, 6)` unpacks the tuple `(5, 6)` into individual arguments. So, effectively, it's like calling `func(1, 5, 6)`, and `c` retains its default value, which is 8, and `d` retains its default value, which is 5.

6.

(1, ['x'], {'a': 'y'})

Inside the function `func`, `a` is reassigned to 2, `b[0]` (the first element of the list `m`) is modified to `'x'`, and the value associated with the key `'a'` in the dictionary `n` is changed to `'y'`. After calling `func(l, m, n)`, the values of `l`, `m`, and `n` outside the function are printed. `l` remains unchanged (immutable type), `m` reflects the change (mutable type), and `n` also reflects the change (mutable type).