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

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

print(a, b, c)

>>> func(1, 2)

ANSWER.

1 2 8

Explanation:

- 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 `func(1, 2)` is called, `1` is assigned to parameter `a` and `2` is assigned to parameter `b`. Since no value is provided for `c`, it takes its default value of `8`.

- Therefore, the function call `func(1, 2)` prints `1` for `a`, `2` for `b`, and `8` for `c`.

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)

ANSWER.

1 2 3

Explanation:

- The function `func()` is defined with three parameters: `a`, `b`, and `c`, where `c` has a default value of `5`.

- When `func(1, c=3, b=2)` is called, the value `1` is assigned to parameter `a`, `3` is explicitly assigned to parameter `c`, and `2` is explicitly assigned to parameter `b`.

- Therefore, the function call `func(1, c=3, b=2)` prints `1` for `a`, `2` for `b`, and `3` for `c`. The explicitly provided arguments override the default values for `b` and `c`.

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

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

print(a, pargs)

>>> func(1, 2, 3)

ANSWER.

1 (2, 3)

Explanation:

- The function `func()` is defined with a parameter `a` and a variable-length argument list `\*pargs`.

- When `func(1, 2, 3)` is called, `1` is assigned to parameter `a`, and the remaining arguments `2` and `3` are collected into a tuple assigned to `pargs`.

- Therefore, the function call `func(1, 2, 3)` prints `1` for `a` and `(2, 3)` for `pargs`, where `(2, 3)` represents a tuple containing the additional 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)

ANSWER.

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

Explanation:

- The function `func()` is defined with a parameter `a` and a variable-length keyword argument list `\*\*kargs`.

- When `func(a=1, c=3, b=2)` is called, `1` is assigned to parameter `a`, and the additional keyword arguments `c=3` and `b=2` are collected into a dictionary assigned to `kargs`.

- Therefore, the function call `func(a=1, c=3, b=2)` prints `1` for `a` and `{'c': 3, 'b': 2}` for `kargs`, where `{'c': 3, 'b': 2}` represents a dictionary containing the additional keyword arguments passed to the function.

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))

ANSWER.

1 5 6 5

Explanation:

- The function `func()` is defined with four parameters: `a`, `b`, `c`, and `d`, where `c` and `d` have default values of `8` and `5` respectively.

- When `func(1, \*(5, 6))` is called, `1` is assigned to parameter `a`, and the tuple `(5, 6)` is unpacked into arguments and assigned to `b` and `c` respectively. Since there is no value provided for `d`, it takes its default value of `5`.

- Therefore, the function call `func(1, \*(5, 6))` prints `1` for `a`, `5` for `b`, `6` for `c`, and `5` for `d`. The unpacking operation `\*(5, 6)` results in the values `5` and `6` being assigned to `b` and `c` respectively.

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

ANSWER.

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

Explanation:

- The function `func()` takes three arguments `a`, `b`, and `c`, and modifies them within the function.

- When `func(l, m, n)` is called, `l`, `m`, and `n` are passed as arguments to the function.

- Inside the function, `a` is assigned the value `2`, which doesn't affect the value of `l` outside the function because integers are immutable.

- `b[0]` is modified to `'x'`, which changes the first element of the list `m` outside the function because lists are mutable.

- `c['a']` is modified to `'y'`, which changes the value associated with the key `'a'` in the dictionary `n` outside the function because dictionaries are mutable.

- Therefore, after calling the function, the values of `l`, `m`, and `n` outside the function are `(1, ['x'], {'a': 'y'})`.