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

ANSWER

1. \*\*Code 1:\*\*

```python

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

print(a, b, c)

func(1, 2)

```

Output: `1 2 8`

Explanation: In this function call, we specify values for `a` and `b`, but not for `c`. Therefore, `c` takes its default value, which is 8.

2. \*\*Code 2:\*\*

```python

def func(a, b, c=5):

print(a, b, c)

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

```

Output: `1 2 3`

Explanation: Here, we explicitly provide values for `b` and `c`, overriding the default values. The function prints the provided values.

3. \*\*Code 3:\*\*

```python

def func(a, \*pargs):

print(a, pargs)

func(1, 2, 3)

```

Output: `1 (2, 3)`

Explanation: The `\*pargs` parameter allows the function to accept a variable number of positional arguments after `a`. In this case, the function receives `a=1` and `pargs=(2, 3)`.

4. \*\*Code 4:\*\*

```python

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

print(a, kargs)

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

```

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

Explanation: The `\*\*kargs` parameter allows the function to accept a variable number of keyword arguments. In this call, the function receives `a=1` and a dictionary `kargs` containing the keyword arguments and their values.

5. \*\*Code 5:\*\*

```python

def func(a, b, c=8, d=5):

print(a, b, c, d)

func(1, \*(5, 6))

```

Output: `1 5 6 5`

Explanation: The `\*(5, 6)` syntax is used to unpack the tuple `(5, 6)` into separate positional arguments. So, the function receives `a=1`, `b=5`, and `c=6`. The default value of `d` (5) remains unchanged.

6. \*\*Code 6:\*\*

```python

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

```

Output:

```

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

```

Explanation: In the function `func`, it modifies the variables `a`, `b`, and `c`. After the function call, `l`, `m`, and `n` are changed to `(1, ['x'], {'a': 'y'})`. This demonstrates that lists and dictionaries can be modified in-place within functions, while integers (immutable) are not changed outside the function.