Name :- Deepawali . B. Mhaisagar
Assignment no 23

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
The result of the code will be `1 2 8`.

Here's an explanation of the code:

1. The `func` function is defined with three parameters: `a`, `b`, and `c`. The parameter `b` has a default value of `6`, and the parameter `c` has a default value of `8`.
2. When calling the `func` function with the arguments `1` and `2` (`func(1, 2)`), the argument `1` is assigned to the parameter `a`, and the argument `2` is assigned to the parameter `b`. Since the argument `c` is not provided, the default value `8` for the parameter `c` is used.
3. The `print(a, b, c)` statement inside the `func` function prints the values of the parameters `a`, `b`, and `c`.
4. Therefore, when `func(1, 2)` is called, it prints `1 2 8` as the output. The value of `a` is `1`, the value of `b` is `2` (overriding the default value), and the value of `c` is `8` (using the default value).

Hence, the output of the code will be:

```
1 2 8
```

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
The result of the code will be `1 2 3`.

Here's an explanation of the code:

1. The `func` function is defined with three parameters: `a`, `b`, and `c`. The parameter `c` has a default value of `5`.
2. When calling the `func` function with the arguments `1`, `c=3`, and `b=2` (`func(1, c=3, b=2)`), the argument `1` is assigned to the parameter `a`. The argument `3` is explicitly passed for the parameter `c` using the keyword argument `c=3`, overriding

the default value. Similarly, the argument `2` is explicitly passed for the parameter `b` using the keyword argument `b=2`.
3. The `print(a, b, c)` statement inside the `func` function prints the values of the parameters `a`, `b`, and `c`.
4. Therefore, when `func(1, c=3, b=2)` is called, it prints `1 2 3` as the output. The value of `a` is `1`, the value of `b` is `2` (passed explicitly), and the value of `c` is `3` (overriding the default value).

Hence, the output of the code will be:

```
1 2 3
```

3. How about this code: what is its result, and why?
>>> def func(a, *pargs):
print(a, pargs)
>>>func(1, 2, 3)

ANS

The result of the code will be `1 (2, 3)`.

Here's an explanation of the code:

1. The `func` function is defined with two parameters: `a` and `*pargs`. The `*pargs` parameter is preceded by an asterisk (`*`), indicating that it is a variable-length argument list or "args" in Python terminology. This parameter can accept any number of positional arguments and collects them into a tuple.
2. When calling the `func` function with the arguments `1`, `2`, and `3` (`func(1, 2, 3)`), the argument `1` is assigned to the parameter `a`, and the remaining arguments `2` and `3` are collected into the `pargs` tuple.
3. The `print(a, pargs)` statement inside the `func` function prints the values of the parameters `a` and `pargs`.
4. Therefore, when `func(1, 2, 3)` is called, it prints `1 (2, 3)` as the output. The value of `a` is `1`, and the value of `pargs` is the tuple `(2, 3)` containing the remaining arguments passed to the function.

Hence, the output of the code will be:

```
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)

ANS

The code will print `1 {'c': 3, 'b': 2}`.

Here's an explanation of the code:

1. The `func` function is defined with two parameters: `a` and `**kargs`. The `**kargs` parameter is preceded by double asterisks (`**`), indicating that it is a variable-length keyword argument dictionary or "kwargs" in Python terminology. This parameter can accept any number of keyword arguments and collects them into a dictionary.
2. When calling the `func` function with the keyword arguments `a=1`, `c=3`, and `b=2` (`func(a=1, c=3, b=2)`), the value `1` is assigned to the parameter `a`, and the remaining keyword arguments `c=3` and `b=2` are collected into the `kargs` dictionary.
3. The `print(a, kargs)` statement inside the `func` function prints the values of the parameters `a` and `kargs`.
4. Therefore, when `func(a=1, c=3, b=2)` is called, it prints `1 {'c': 3, 'b': 2}` as the output. The value of `a` is `1`, and the value of `kargs` is the dictionary `{'c': 3, 'b': 2}` containing the keyword arguments passed to the function.

Hence, the output of the code will be:

```
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))

ANS

The code will print `1 5 6 5`.

Here's an explanation of the code:

1. The `func` function is defined with four parameters: `a`, `b`, `c`, and `d`. The parameters `c` and `d` have default values of `8` and `5` respectively.
2. When calling the `func` function with the arguments `1` and `*(5, 6)` (`func(1, *(5, 6))`), the argument `1` is assigned to the parameter `a`. The tuple `(5, 6)` is unpacked using the asterisk `*`, which means that its elements are treated as individual arguments. Therefore, the element `5` is assigned to the parameter `b`, and the element `6` is assigned to the parameter `c`.

3. Since the argument `d` is not provided, the default value `5` for the parameter `d` is used.
4. The `print(a, b, c, d)` statement inside the `func` function prints the values of the parameters `a`, `b`, `c`, and `d`.
5. Therefore, when `func(1, *(5, 6))` is called, it prints `1 5 6 5` as the output. The value of `a` is `1`, the value of `b` is `5` (assigned from the unpacked tuple), the value of `c` is `6` (assigned from the unpacked tuple), and the value of `d` is `5` (using the default value).

Hence, the output of the code will be:

```
1 5 6 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
The result of the code will be `1, ['x'], {'a': 'y'}`.

Here's an explanation of the code:

1. The `func` function is defined with three parameters: `a`, `b`, and `c`.
2. Inside the `func` function, the variable `a` is reassigned the value `2`. This modification is local to the function and does not affect the variables outside the function.
3. The list `b` is modified by assigning `'x'` to its first element `b[0]`. This change modifies the list in place, so the modification is visible outside the function.
4. The dictionary `c` is modified by assigning `'y'` to its key `'a'` (`c['a'] = 'y'`). This modification is also visible outside the function.
5. After defining the `func` function, three variables are initialized: `l` with the value `1`, `m` as a list with the value `[1]`, and `n` as a dictionary with the key `'a'` and value `0`.
6. Then, the `func(l, m, n)` function is called, passing the variables `l`, `m`, and `n` as arguments. This causes the modifications explained in step 3 and step 4 to occur.
7. Finally, the values of `l`, `m`, and `n` are printed using `l, m, n`.

Therefore, the output of the code will be:

```
1, ['x'], {'a': 'y'}
```

The value of `l` remains `1` since the reassignment within the function does not affect the variable outside the function. The list `m` is modified to `['x']` because lists are mutable and changes made to them are visible outside the function. The dictionary `n` is modified to `{'a': 'y'}` because dictionaries are also mutable and changes made to them are visible outside the function.