Python Assignment - 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)**

=>

Output: 1 2 8

Explanation:

* The code defines a function func() with three parameters: a, b, and c.
* The parameter a is a positional parameter, meaning it needs to be provided with an argument when calling the function.
* The parameters b and c have default values of 6 and 8 respectively. If no argument is provided for these parameters, the default values will be used.
* When func(1, 2) is called, the argument 1 is assigned to the parameter a, and the argument 2 is assigned to the parameter b.
* Since no argument is provided for c, it takes its default value of 8.
* The print(a, b, c) statement within the function is executed, which prints the values of a, b, and c.
* Therefore, the output is 1 2 8, representing the values of the function's parameters a, b, and c respectively.

**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)**

=>

Output: 1 2 3

Explanation:

* The code defines a function func() with three parameters: a, b, and c.
* The parameters a and b are positional parameters, meaning they need to be provided with arguments in the order they are defined.
* The parameter c has a default value of 5. If no argument is provided for c, the default value will be used.
* When func(1, c=3, b=2) is called, the argument 1 is assigned to the parameter a, the argument 2 is explicitly assigned to the parameter b, and the argument 3 is explicitly assigned to the parameter c.
* By using keyword arguments (c=3 and b=2), we can explicitly specify which parameter each argument corresponds to, regardless of their order.
* The print(a, b, c) statement within the function is executed, which prints the values of a, b, and c.
* Therefore, the output is 1 2 3, representing the values of the function's parameters a, b, and c respectively. The explicitly assigned values 2 and 3 override the default value of c=5.

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

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

**print(a, pargs)**

**>>> func(1, 2, 3)**

=>

Output: 1 (2, 3)

Explanation:

* The code defines a function func() with two parameters: a and \*pargs.
* The parameter a is a positional parameter, meaning it needs to be provided with an argument when calling the function.
* The parameter \*pargs is preceded by an asterisk (\*), indicating that it will collect any additional positional arguments into a tuple.
* When func(1, 2, 3) is called, the argument 1 is assigned to the parameter a, and the remaining arguments 2 and 3 are collected into the pargs tuple.
* The print(a, pargs) statement within the function is executed, which prints the value of a and the tuple pargs.
* Therefore, the output is 1 (2, 3), representing the value of a and the tuple containing the additional positional 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)**

=>

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

Explanation:

* The code defines a function func() with two parameters: a and \*\*kargs.
* The parameter a is a keyword parameter, meaning it can be provided with an argument using the keyword syntax.
* The parameter \*\*kargs is preceded by two asterisks (\*\*), indicating that it will collect any additional keyword arguments into a dictionary.
* When func(a=1, c=3, b=2) is called, the argument 1 is passed to the parameter a using the keyword syntax. The keyword arguments c=3 and b=2 are collected into the kargs dictionary.
* The print(a, kargs) statement within the function is executed, which prints the value of a and the dictionary kargs.
* Therefore, the output is 1 {'c': 3, 'b': 2}, representing the value of a and the 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))**

=>

Output: 1 5 6 5

Explanation:

* The code defines a function func() with four parameters: a, b, c, and d.
* The parameters a and b are positional parameters, meaning they need to be provided with arguments in the order they are defined.
* The parameters c and d have default values of 8 and 5 respectively. If no arguments are provided for these parameters, the default values will be used.
* When func(1, \*(5, 6)) is called, the argument 1 is assigned to the parameter a.
* The \*(5, 6) syntax is used to unpack the tuple (5, 6) into separate arguments. So, 5 is assigned to the parameter b, and 6 is assigned to the parameter c.
* Since no argument is provided for d, it takes its default value of 5.
* The print(a, b, c, d) statement within the function is executed, which prints the values of a, b, c, and d.
* Therefore, the output is 1 5 6 5, representing the values of the function's parameters a, b, c, and d 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**

=>

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

Explanation:

* The code defines a function func() with three parameters: a, b, and c.
* Inside the function, the line a = 2 reassigns the local variable a to the value 2.
* The line b[0] = 'x' modifies the first element of the list b to the value 'x'. This change affects the original list since lists are mutable objects.
* The line c['a'] = 'y' modifies the value associated with the key 'a' in the dictionary c to 'y'. This change also affects the original dictionary since dictionaries are mutable objects.
* Before calling func(), the variables l, m, and n are assigned the values 1, [1], and {'a': 0} respectively.
* When func(l, m, n) is called, the arguments l, m, and n are passed to the function. The variable l is passed by value (immutable), and the variables m and n are passed by reference (mutable).
* Inside func(), the values of a, b, and c are modified as described above.
* After the function call, the values of l, m, and n are printed, resulting in the output (1, ['x'], {'a': 'y'}). The changes made to m and n within the function are reflected in the variables outside the function. However, the variable l remains unaffected since it is immutable and passed by value.