**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)**

**The code will output the following result:**

**Copy code**

**1 2 8**

**In the function definition, the parameter a has no default value, while the parameters b and c have default values of 6 and 8, respectively.**

**When the function is called with func(1, 2), the argument 1 is passed to the parameter a, and the argument 2 is passed to the parameter b. Since no value is passed for the parameter c, the default value of 8 is used.**

**Therefore, the output of the function call is 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)**

**The code will output the following result:**

**Copy code**

**1 2 3**

**In the function definition, the parameters a and b have no default values, while the parameter c has a default value of 5.**

**When the function is called with func(1, c=3, b=2), the argument 1 is passed to the parameter a, the argument 2 is passed to the parameter b, and the argument 3 is passed to the parameter c through a keyword argument.**

**Since the parameter c is assigned a value through the keyword argument c=3, the default value of 5 is overridden, and the value of c becomes 3. Therefore, the output of the function call is 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)**

**The code will output the following result:**

**scss**

**Copy code**

**1 (2, 3)**

**In the function definition, the parameter a has no default value, while the parameter \*pargs is a variable-length argument list, which means it can take any number of positional arguments.**

**When the function is called with func(1, 2, 3), the argument 1 is passed to the parameter a, and the arguments 2 and 3 are passed to the variable-length argument list pargs.**

**The \*pargs syntax is used to pack all the positional arguments after the first one (a) into a tuple. Therefore, the output of the function call is 1 (2, 3), where the first argument 1 is assigned to a, and the tuple (2, 3) is assigned to pargs.**

**4. What does this code print, and why?**

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

**print(a, kargs)**

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

**The code will output the following result:**

**arduino**

**Copy code**

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

**In the function definition, the parameter a has no default value, while the parameter \*\*kargs is a variable-length keyword argument list, which means it can take any number of keyword arguments.**

**When the function is called with func(a=1, c=3, b=2), the keyword arguments c=3 and b=2 are passed to the variable-length keyword argument list kargs. The argument 1 is passed to the parameter a.**

**The \*\*kargs syntax is used to pack all the keyword arguments, excluding those already assigned to other parameters, into a dictionary. Therefore, the output of the function call is 1 {'c': 3, 'b': 2}, where the first argument 1 is assigned to a, and the dictionary {'c': 3, 'b': 2} is assigned to kargs. The dictionary contains the keyword arguments c=3 and b=2 that were 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))**

**The code will output the following result:**

**Copy code**

**1 5 6 5**

**In the function definition, the parameters a and b have no default values, while the parameters c and d have default values of 8 and 5, respectively.**

**When the function is called with func(1, \*(5, 6)), the argument 1 is passed to the parameter a, and the tuple (5, 6) is unpacked and its elements 5 and 6 are passed to the parameters b and c, respectively. Since no value is passed for the parameter d, the default value of 5 is used.**

**Therefore, the output of the function call is 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**

**The code will output the following result:**

**css**

**Copy code**

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

**In the function definition, the parameters a, b, and c are assigned new values, but these changes have no effect outside the function.**

**When the function is called with func(l, m, n), the argument l is passed to the parameter a, the argument m is passed to the parameter b, and the argument n is passed to the parameter c.**

**Inside the function, the variable a is assigned a new value of 2. This has no effect outside the function, because a is a local variable and its scope is limited to the function.**

**The variable b is a mutable object (a list), so when its first element is changed to 'x' inside the function, this change affects the original list outside the function. Therefore, the value of m changes to ['x'].**

**Similarly, the variable c is a mutable object (a dictionary), so when its key 'a' is assigned a value of 'y' inside the function, this change affects the original dictionary outside the function. Therefore, the value of n changes to {'a': 'y'}.**

**After the function call, the values of l, m, and n are (1, ['x'], {'a': 'y'}), as explained above.**