**PYTHON BASIC ASSIGNMENT\_23 - SUBMITTED BY SAMUEL DEVDAS**

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 function ‘func(a,b=6,c=8) is defined to take three arguments, a with no default value, b and c with with default values 6 and 8 respectively.

Then, the function call func(1,2) is going to assign a=1, b=2 and c=8 by default. The result of print(1,2,8) will output as ‘1 2 8’ in the console.

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. Similar, to question 1, The function ‘func(a,b, c=5) is defined to take three arguments, a and b with no default value and c with default value as 5.

Then, the function call func(1,c=3,b=2) is going to assign b=2 and c=3. The result of print(1,2,3) will output as ‘1 2 3’ in the console.

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

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

print(a, pargs)

>>> func(1, 2, 3)

Ans. ‘\*pargs’ used in function ‘func(a, \*pargs)’ definition acts as an alias for \*args or arbitrary arguments and ‘a’ is the first required argument.

Function call ‘func(1,2,3), will be evaluated by the ‘print(a,pargs)’ statement as: ‘a’ will take the value ‘1’ and pargs will take the values (1,2) in tuple form.

The final output 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. Ans. ‘\*kargs’ used in function ‘func(a, \*kargs)’ definition acts as an alias for \*kwargs or keyword arguments and ‘a’ is the first required argument.

Function call ‘func(a=1,c=3,b=2), will be evaluated by the ‘print(a, kargs)’ statement as: ‘a’ will take the key ‘a’ and value ‘1’ and kargs will take the values {‘c’=3,’b’=2} in dictionary form.

The final output 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. For the defined function ‘func(a, b, c=8, d=5)’,‘a’ is the first required argument. ‘b’ is the second required argument. ‘c’ and ‘d’ are default values ‘8’ and ‘5’ respectively.

Now, Function call ‘func(1,\*(5,6)), will be evaluated by the ‘print(a,b,c,d)’ statement as: ‘a’ will be assigned the value ‘1’, ‘b’ will take the value ‘5’, ‘c’ will be assigned the value ‘6’ and ‘d’ will take the default value ‘5’ from the defined function.

The final output 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. 1. def func(a, b, c):

a = 2

b[0] = 'x'

c['a'] = 'y'

Consider 1, The function ‘func(a,b,c)’ is defined and variables are declared into the function like, ‘a’ is assigned the value ‘2’, index ‘0’ of list ‘b’ is assigned the value ‘x’ (b=[‘x’]) and key ‘a’ of dictionary ‘c’ is assigned the value ‘y’(c={‘a’:’y’}).

2. l=1

m=[1]

n={'a':0}

Now, for 2.: variable ‘l’ is assigned the value ‘1’, variable ‘m’ is assigned a list with an element ‘1’ and variable ‘n’ is assigned a dictionary with a key-value pair ‘a’ and ‘0’.

3. func(l, m, n)

l, m, n

Next for 3.: function ‘func(l,m,n)’ is called, which means for func(a,b,c) the arguments values l,m and n are passed in place of variables a, b and c, ie. Variable reassignment is performed, resulting in a=1, m=[1] is passed to b, which wont change anything, ie. b=[‘x’] and lastly, n={‘a’:0} is passed to l, which also wont change anything, i.e c={‘a’:’y’}.

Finally, l,m and n are returned in output as (1, ['x'], {'a': 'y'})