**Python – Assignment 23**

|  |  |
| --- | --- |
| **S. No.** | **Question / Answer** |
| 1 | What is the result of the code, and why?  >>> def func(a, b=6, c=8):  print(a, b, c)  >>> func(1, 2) |
|  | 1 2 8  value for a and b is passed to the func() as 1 and 2, while c defaults to 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) |
|  | 1 2 3  values for a, b and c are explicitly passed to the function. |
| 3 | How about this code: what is its result, and why?  >>> def func(a, \*pargs):  print(a, pargs)  >>> func(1, 2, 3) |
|  | 1 (2, 3)  pargs – is a tuple of all the arguments that are being passed to the function except the first argument. |
| 4 | What does this code print, and why?  >>> def func(a, \*\*kargs):  print(a, kargs)  >>> func(a=1, c=3, b=2) |
|  | 1 {‘c’: 3, ‘b’: 2}  kwargs – is a dictionary with key-value pairs of the parameter names and argument values. |
| 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)) |
|  | 1 5 6 5  Using \*(5,6), means to unpack the tuple to the respective arguments. Here (5, 6) is unpacked to fulfil the values for b and c, while d defaults to 5 and a is set to 1. |
| 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 |
|  | (1, [‘x’], {‘a’: ‘y’})  When a single value is passed as an argument to a function, it is a copy-by-value pass. Where as a structure/ collection is passed as an argument to a function, like a list or dictionary, it is a copy-by-reference pass. So when we set a = 2 in the function it is not reflected in the global l. While when we modify b or c, it is reflected in global m and n. |