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**

1 2 8

func(1,2) will be considered as argument to a and b since default value is given in function defnititon it will be considered for c value

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**

1 2 3

Order of the parameter doesn’t matter when we are explicitly mentioning them ordere should be followed only when giving without mentioning

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

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

print(a, pargs)

>>> func(1, 2, 3)

**ANS**

1 (2,3)

The special syntax *\*args* in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list.

4. What does this code print, and why?

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

print(a, kargs)

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

**ANS**

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

The special syntax *\*\*kwargs* in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name *kwargs* with the double star. The reason is because the double star allows us to pass through keyword arguments (and any number of them).

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**

1 5 6 5

By adding \* in parameter list it can be used as argument in the order mentioned in the function definition since d had a default value mentioned in the definition

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, ['x'], {'a': 'y'})