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:**

**By default, values b and c are 6 and 8. When parameters were passed to the function, second parameter overrides default value 6.**

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:**

**Function’s default parameter for c was overridden by passed argument c=3. We can pass argument to the function parameters either in order or without order. When there is no order, we have to specify which parameter gets which argument.**

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:**

**\*pargs will collect arguments and put them into a tuple.**

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:**

**\*\*kargs takes keyword arguments and puts them into dictionary.**

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:**

**\*(5,6) gets passed to b and c parameters. C parameter gets overridden with the new value d parameter remains default.**

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:**

**Numbers, strings and tuples are not mutable, that’s why first argument is not overwritten. Lists and dictionaries are mutable. When they are passed to a function, they reference to the original, not a copy, thus original one gets modified.**