**Ex-3 Functions**

Ex-2와 동일한 요령으로 수행한다.

## 1. Argument-matching examples.

First, define the following six functions (either interactively or in a module file that can be imported):

def f1(a, b): print(a, b) # Normal args

def f2(a, \*b): print(a, b) # Positional var args

def f3(a, \*\*b): print(a, b) # Keyword var args

def f4(a, \*b, \*\*c): print(a, b, c) # Mixed modes

def f5(a, b=2, c=3): print(a, b, c) # Default values

def f6(a, b=2, \*c): print(a, b, c) # Defaults and positional var args

Now, test the following calls interactively..

# case a)

f1(1, 2)

f1(b=2, a=1)

# case b)

f2(1, 2, 3)

f3(1, x=2, y=3)

f4(1, 2, 3, x=2, y=3)

# case c)

f5(1)

f5(1, 4)

# case d)

f6(1)

f6(1, 3, 4)

Try to explain each result.

Your answers here.

Case a) In first line, it called the function f1 with argument 1 and 2, so print function called sucessfully. In second line, it assigned 2 and 1 in to variable b and a, and after it works, the function f1 called. So the same result came with first line.

Case b) In function f2, asterisk before argument b means multiple variable arguments. It called non-keyworded arguments. So that we can use f2 functions with 3 arguments in first line. In function f3, double asterisk means kargs(keyworded arguments) and it handles arguments like dictionary. You can see the result in line 2, it has resulted with dict. Function f4 is mixed function with non-keyworded arguments and keyworded arguments. Result in line 3 has also similar phrases with line 2.

Case c) Function f5 indicates default parameter value. In first line, there is only one parameter to call function. So it has a result like 1 2 3. But f5 has default value b=2 and c=3, so there is another parameter with those places like f5(1,4), it would have to take 1 4 3.

Case d) Function f6 is mixed function with default parameter value and non-keyworded arguments. It's result caused like f2 and f5. Second parameter b=2 works by default parameter value , third parameter \*c non-keyworded arguments which explained at case b).

## 2. Arguments.

Write a function called ```adder``` in a Python module file. The function should accept two arguments and return the sum (or concatenation) of the two. Then, call the ```adder``` function with a variety of object types (two strings, two lists, two floating points).

def adder(arg1, arg2):

return arg1+arg2

print(adder('pine','apple'))

print(adder([1,2,3,4],[5,6,7,8]))

print(adder(1.5,5.6))

## 3. Variable arguments.

Generalize the adder function you wrote in the above to compute the sum of an arbitrary number of arguments, and change the calls to pass more or fewer than two arguments.

Hints: a slice such as ```S[:0]``` returns an empty sequence of the same type as S, and the type builtin function can test types

def adder(arg1,\*arg2):

for arg in arg2:

arg1 += arg

return arg1

print(adder(1,3,5,7))

What happens if you pass in arguments of different types?

print(adder('happy','birthday','HUFS!'))

What about passing in dictionaries?

print(adder([1,2,3,4],[5,6,7,8],[9,10,11,12]))

## 4. Dictionary.

Write a function called ```add\_dict(dict1, dict2)``` that computes the union of two dictionaries. It should return a new dictionary containing all the items in both its arguments (which are assumed to be dictionaries). If the same key appears in both arguments, feel free to pick a value from either.

def add\_dict(dict1, dict2):

dict1.update(dict2)

return dict1

print(add\_dict({'apple':1000,'orange':2000,'banana':3000},{'orange':1000,'pear':1500,'pineapple':5000}))

#orange printed smaller prices because of the union of dict

What happens if you pass lists instead of dictionaries?

def add\_list(list1, list2):

list1 += list2

return list1

print(add\_list(['apple','orange','banana'],['orange','pear','pineapple']))

How could you generalize(rewrite) your function to handle this case, too? (Hint: see the ```type``` built-in function used earlier.)

## 5. Computing factorials.

N!, is computed as N\*(N-1)\*(N-2)\*...1. For instance, 6! is 6\*5\*4\*3\*2\*1, or 720. Write a recursive function ```fact1(N)``` and an iterative function ```fact2(N)```. (Note: 0! == 0)

def fact1(N):

if N == 1:

return 1

else:

return N\*fact1(N-1)

def fact2(N):

temp = 1

for i in range(1,N+1):

temp \*= i

return temp

print(fact1(6))

print(fact2(6))