INF1002 Programming Fundamentals – Lab3

Topics:

1. Functional Abstraction
2. Functions and modules
3. Higher Order Functions

Warmup exercises:

The following lab assignment requires the use of all topics discussed so far in the module. You may wish to practice some of the concepts with simple exercises before attempting the lab assignment. You are not required to include these exercises in your submission, though it may help you in the quiz.

1. Define one function to calculate the division of given two numbers, return the result:

   ____

   ____

   ____

2. Define one function to print out all the element for one given list – the input argument is one list, and nothing needs to be returned.

   ____

   ____

   ____

3. Evaluate the following code and figure out the output and explain why?

```
def printMax(a, b):
    if a > b:
        print(a, 'is maximum')
    elseif a == b:
        print(a, 'is equal to', b)
    else:
        print(b, 'is maximum')
```

   Evaluate the following expressions

printMax(3,4)	printMax(3)
printMax(3,3)	printMax(3,4,5)
printMax(4,4)	printMax('charlie', 'hello')

4. Evaluate what is the output of the following code and understand why

```
def say(message, times = 2):
    print(message * times)

say('Hello')
say('World', 5)
```

   ____

   ____

   ____

5. Evaluate the following code and understand why

```
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)

func(3, 7)
func(25, c = 24)
func(c = 50, a = 100)
```

6. Evaluate what is the output of the following program and explain why?

```
def func(a, b, names):
     a = a+10
     b = b+20
     names[0] = 12
     names[1] = 18
     return a,b

x,y=10,30
fruits=['apple', 'orange', 'banana']
num1, num2=func(x,y,fruits)
print num1, num2
for fruit in fruits:
     print fruit
```

7. According to Lab2 Task 3, write one function to decide whether one year is a leap year or not. If it is one leap year, return true. Otherwise, return false.

8. The following examples of writing higher order functions may help you better understand the basics covered by the lecture. Please evaluate the following code, and explain and understand why the output is that:

```
>>> def func(x):
        return x+5

>>> func(20)
```

Ans:

```
>>> sorted(abs, [100,-800,400,-200,50])
```

Ans: If there is an error, then fix it.

a)

```
def printWelcome():
            return 'Welcome:'
def messager(func,str1):
            print func(),str1

messager(printWelcome, 'Python')
```

b)

```
def increment(x):
     return x+100
def double(x):
     return x*2
def getBonus(func, salary):
     bonus = 1000
     if func(salary) > 5000:
          return func(salary)+bonus*2
     else:
          return func(salary)+bonus

print (getBonus(increment, 3000))
print (getBonus(double, 3000))
print (getBonus(increment, 6000))
print (getBonus(double, 6000))
```

Lab Assignment:

To help you better practice, you need to perform a set of tasks in one auto-grading system, Gradescope in LMS/xSiTe. Gradescope will provide you immediate feedback of your program, such that you will know the issue of your program.

In this lab, you need to finish three tasks in Gradescope system. Gradescope system will provide you immediate feedback about the correctness of your program. You can view the feedback of your system by clicking the failed case. Gradescope adopts a test cases-based approach to check your program. Your score is depending on the number of test cases that you program can pass. Note that to train you to have a good programming practice, you must write your program strictly according to the requirement of the tasks, including your input and output format. If there is any difference (even one space), your program will fail on the test cases. SO, TRAIN YOURSELF TO BE AN EXACT THINKER!

To help you practice, you are allowed to do multiple attempts for each task. Enjoy your learning!

Note:

1. **It is noted that usage of '*quit()*', '*exit()*' and '*sys.exit()*' functions to quit/exit a conditional statement or function in Gradescope submission results in test case failure.**
2. **Use '*return*' or other means.**

Task 1: Developing Your Own Math Module

Task Description:

Develop your own math module (**myMath**) including eight basic functions.

Function 1: add(x, y)	→ return the summation of x and y
Function 2: subtraction(x, y)	→ return the subtraction between x and y
Function 3: evenNum(x)	→ return the number of even numbers in the given list
Function 4: maximum(x)	→ return the maximum value from the given list x
Function 5: minimum(x)	→ return the minimum value from the given list x
Function 6: absolute(x)	→ return the absolute value of one number x
Function 7: sumTotal(x)	→ return the summation of all the elements for a given list x
Function 8: clear(x)	→ this function sets all the elements into 0 for a given list x

Write one program (**myMain**) to import the module you have created to implement the following functions.

Step 1:

Input: allows users to input multiple numbers. For simplicity, you can assume all the input numbers are integers.

Step 2:

Store all the numbers into one list

Step 3:

a. Calculate and print out the difference between the biggest and the smallest number in the list

b. Calculate and print out the summation between the biggest and the smallest numbers in the list

c. Calculate and print out the summation of all the input numbers in the list
d. Calculate and print out the number of even numbers in the list
e. If the smallest number in the list smaller than 5, set all the value to 0. Otherwise, remain the same. Print all the values in the list in the end.

Hint:

- To split one string, split() function can be used.
- For instance, str = ’1,2,3,4,10’, list1 = str.split(‘,’) would split the string and store the numbers in the list.
- Please keep in mind that each element in the list is a string type now. You must figure out one way to convert all of them into integer.

Please print your results according to this format:

print("The difference is:%d The summation is:%d The summation of all input is:%d The number of even numbers is:%d The values in the list are: %s" % (myMath.subtraction(maxNum, minNum), myMath.add(maxNum, minNum), …)

There are **two python files required for submission**: **myMain.py** and **myMath.py**.
Please submit your main code File as **myMain.py** (to be named specifically for Gradescope) and add Function File, **myMath.py**. Both files need to be in the same directory.

Running example: *(Your output should be in ONE line)*

```
C:\INF1002\Lab3\MathModule>python myMain.py 12,10,11,23,25,2

The difference is:23 The summation is:27 The summation of all input
is:83 The number of even numbers is:3 The values in the list are:
[0, 0, 0, 0, 0, 0]
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file “myMain.py” and “myMath.py” from the “Python\Lab\Lab3” xSiTe folder.
2. Add your code to the function “*def myMain()*” in myMain.py and functions in myMath.py.
3. Do not change file name or function names.
4. Drag and drop your locally tested code to Gradescope. You can submit and auto graded any number of times. Last submission counts.
5. There are 5 test cases and maximum score for this task is 5.
6. Autograder will be timed out after 10 minutes.

Task 2: Counting Letters

Task Description:

In this task, you are going to develop a letter game to count the number of letters in the given string.

Step1:

Write one function *letter_count(str)* that take in one word and returns a dictionary with the frequency counts of the various letters. Upper and lower-case characters are different characters.
Sample execution:
>>letter_count('This')
{ 'h':1, 'T':1, 'i':1, 's':1}

>>letter_count('Thisisit')
{'h':1, 'T':1, 'i':3, 's':2, 't':1}

Step 2:

Write one function *double_count(str1, str2)* that takes in two words and returns a dictionary with the frequency counts of the various letters. Upper and lower-case characters are different characters.
Sample execution:
>> double_count('This', 'isit')
{'h':1, 'T':1, 'i':3, 's':2, 't':1}

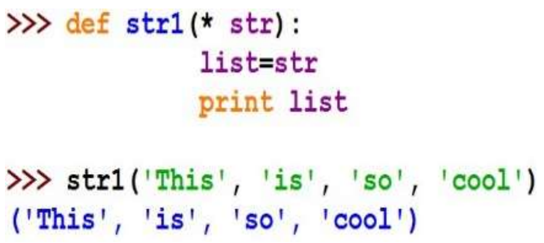
Step 3:

Write one function *various_count(*str)* that takes in any number of words and returns a dictionary with the frequency counts of the various letters. Upper and lower-case characters are different characters.
Sample execution:
>> various_count('This')
{ 'h':1, 'T':1, 'i':1, 's':1}

>> various_count('This', 'isit')
{'h':1, 'T':1, 'i':3, 's':2, 't':1}

>> various_count('This', 'is, 'it')
{'h':1, 'T':1, 'i':3, 's':2, 't':1}

HINT: In Python, using "*def func(*str): list1=str*", list1 can obtain any number of arguments and stores it as a list. You can further get each element from the list and count each word independently. You can implement another function to merge two dictionaries. Below is an example:



```
>>> def str1(* str):
            list=str
            print list

>>> str1('This', 'is', 'so', 'cool')
('This', 'is', 'so', 'cool')
```

Step 4:

Write one program to allow users to input different number of words and output each character's frequency.

Some of codes read as follows. Please print your results in the characters' ASCII descending order according to this format:

```
for item in sorted_total:
    print ('%s:%d' % (item, total[item]), end=' ')
```

Note that the following running example is in ONE line, and your output should be in ONE line.

Running example:

```
C:\INF1002\Lab3\CountingLetters>python CountLetters.py
Firefox,is,having,trouble,recovering,your,windows,and,tabs

y:1 x:1 w:2 v:2 u:2 t:2 s:3 r:5 o:5 n:4 l:1 i:5 h:1 g:2 f:1
e:4 d:2 c:1 b:2 a:3 F:1
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "CountLetters.py" from the "Python\Lab\Lab3" xSiTe folder.
2. Add your code to the function "*def CountLetters()*".
3. Do not change file name or function names.
4. Drag and drop your locally tested code to Gradescope. You can submit and auto graded any number of times. Last submission counts.
5. There are 5 test cases and maximum score for this task is 5.
6. Autograder will be timed out after 10 minutes.

Task 3: Pattern Searching

Task Description:

In this task you will write a program that reads two sequences of numbers. The first sequence of numbers is called *candidate,* and the second sequence of numbers is called *pattern*. Your program will determine if the pattern is found **entirely** in the candidate. To be considered found entirely, all elements of *pattern* must be in the *candidate* sequence at consecutive positions. You must output the number of found patterns, or 0 if the *pattern* is not found in the candidate.

Input: Allow the users to input **two sequences**, the first sequence is the candidate, and the second sequence is the pattern.

Output: The number of pattern sequence appearing in the candidate sequence (in ONE line).

Running Example:

```
C:\INF1002\Lab3\PatternSearching>python SearchPattern.py 1,2,3,1,2 1,2

Pattern appears 2 time!
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "SearchPattern.py" and from the "Python\Lab\Lab3" xSiTe folder.

2. Add your code to the function “*def SearchPattern()*”.
3. Do not change file name or function names.
4. Drag and drop your locally tested code to Gradescope. You can submit and auto graded any number of times. Last submission counts.
5. There are 5 test cases and maximum score for this task is 5.
6. Autograder will be timed out after 10 minutes.

Final submission and auto grading all three tasks together:

After testing all the above three tasks, submit (drag and drop) all four files *myMain.py*, *myMath.py*, *CountLetters.py* and *SearchPattern.py* together and auto graded for a maximum score of 15 for all the 3 tasks.

Note: It is not necessary to zip these files.