INF1002 Programming Fundamentals – Lab4

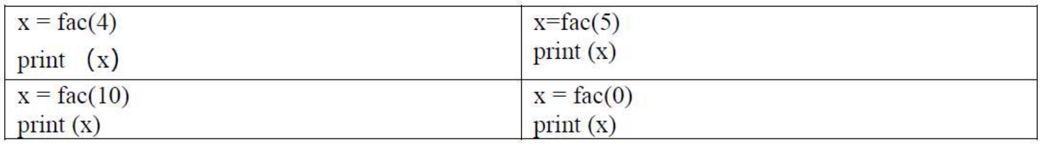
Topics:

1. Recursion
2. DocString

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. Implement the recursive function fac(n) to calculate the factorial number as shown in the lecture notes. And understand the program and test the following expressions:



x = fac(4) print (x)	x=fac(5) print (x)
x = fac(10) print (x)	x = fac(0) print (x)

2. Implement another function fac_iterative(n) to calculate the factorial number. But this time, you need to use the for loop to implement it, other than recursion.

3. In this exercise, we will try to practice how to write the DocString. Please use the myMath module that you created last week, in the lab 3 task 1, and write down the proper comment in each function. And then put module file into your Python library, which is the Lib folder of the python installation folder. In your Python IDLE, try the following commands and see whether you can get the DocString of your own myMath module.

```
>> import myMath
>>help(myMath)
```

4. If you need more exercises to understand the recursive function, you can implement the Fibonacci number and string reversal programs in the lecture notes and understand how they work.

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 Recursive and Iterative SUM Calculator

Task Description:

In this task, we need to design one recursive function to calculate the SUM value of one input number. The SUM of x is defined as the summation of all the positive numbers that are not bigger than X, such as 1+2+3…+X. So please design one recursive function sum_recursive(x) to return the SUM value of x. In addition, please implement another function sum_iterative(x) to return the SUM value of x with the iterative manner (e.g. for loop). Write the main program to allow users to input one number to your program and call these two functions to see whether they get the same output.

The example executions are shown as follows:
*Note: Your output should be in **ONE line***

Running example:

```
C:\INF1002\Lab4\SumCalculator> python SumCalculator.py 3

The SUM value calculated by recursive is 6 and by iterative is 6.
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file “SumCalculator.py” from the “Python\Lab\Lab4” xSiTe folder.
2. Add your code to the function “*def SumCalculator()*” 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: Developing Recursive and Iterative Digits Count

Task Description:

In this task, we need to design one recursive function digit_recursive(x) to calculate how many digits a positive number has. For instance, 10 has two digits, and 122 has three digits, and 5679 has four digits. HINT: The number of digits can be calculated by repeatedly dividing by 10 (without keeping the remainder) until the number is less than 10.

Please write another function digit_iterative(x) to achieve the same functionality to calculate the number of the digits of x but uses while loop. Write one main program to allow users to input one number and call these two functions to evaluate the output.

The example executions are shown as follows:
Note: *Your output should be in **ONE line***

Running example:

```
C:\INF1002\Lab4\CountDigits> python CountDigits.py 789

The number of digit(s) calculated by recursive is 3 and by iterative
is 3.
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file “CountDigits.py” from the “Python\Lab\Lab4” xSiTe folder.
2. Add your code to the function “*def CountDigits()*”.
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: Searching elfish

Task Description:

A word is considered elfish if it contains the letters: e, l and f in it, in any order. For instance, we would say that the following words are elfish: tasteful, whiteleaf, unfriendly and waffles, because they each contain those letters. Use the recursive function to implement this. Write one program to call your recursive function and tell the user whether the input word is one elfish or not.

HINT: You can recursively reduce both the elfish letters and input word.

The sample executions are provided as follows:
Note: *Your output should be in* ***ONE line***

Running example:

```
C:\INF1002\Lab4\elfish> python elfish.py waffles

waffles is one elfish word!


C:\INF1002\Lab4\elfish> python elfish.py instance

instance is not an elfish word!
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "elfish.py" and from the "Python\Lab\Lab4" xSiTe folder.
2. Add your code to the function "*def elfish()*".
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 three files *SumCalculator.py*, *CountDigits.py* and *elfish.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.