INF1002 Programming Fundamentals – Lab5

Topics:

1. Lambda Expression

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

   a)

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

>>> func(20)
```

Ans:

```
>>> fun=lambda x:x+5
>>> fun(20)
```

Ans:

```
>>> map(lambda x:x*2, [10,20,30,40])
```

Ans:

```
>>> map(lambda x:x+30, [3, 4, 6, 7])
```

Ans:

```
>>> filter(lambda x: x<=20, [10,20,30,40])
```

Ans:

```
>>> reduce(lambda x,y: x+y, [10,20,30,40])
```

Ans:

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))
```

c) Similarly, rewrite the getBonus(Salary) function in (b) to replace the lambda *x:x+100(salary)* into another lambda expression to double the number of the salary like the double function in (b). And evaluate *print (getBonus(3000))* and *print (getBonus(6000))* then see whether you can get the same result as the *print (getBonus(double,3000))* and *print (getBonus(double,6000))* in (b).

d) If you still cannot understand these concepts well to this end, please refer to your lecture notes to try all the examples there.

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 two 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 Do-Twice Game

Task Description:

In this task, you need to design one program of performing a "Do Twice" game. The intuitive idea is that when user wants to do one operation, your program helps him to do twice. More detailed procedure is as follows:

a.) Design one function double(x) to calculate the double (2*x) of one number x.
b.) Design one function square(x) to calculate the square (x^2) of one number x.
c.) Design one function cube(x) to calculate the cube (x^3) of one number x.
d.) Design one program to ask user to input one number and another operation number (1 for Double, 2 for Square, 3 for Cube). Once user inputs these information, your program needs to perform the operation twice to user. For example, if user inputs option 1 to double the number x, your program should output double(double(x)) (e.g. 4x) for user. If the user inputs option 2

to square x, your program should output square(square(x)) (e.g. x^4)). If the user inputs 3 to cube the x, your program should output cube(cube(x)) (e.g. x^9).

Hint: You need to write one higher order function doTwice(func, x) to perform the function func twice for given number x.

Note:

a) Options:
   1 - double
   2 - square
   3 – cube
b) *Your output should be in ONE line*

Running example:

```
C:\INF002\Lab5\DoTwiceGame> python DoTwiceGame.py 4 1
16

C:\ INF002\Lab5\DoTwiceGame> python DoTwiceGame.py 4 2
256

C:\ INF002\Lab5\DoTwiceGame> python DoTwiceGame.py 4 4
It cannot be supported!
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "DoTwiceGame.py" from the "Python\Lab\Lab5" xSiTe folder.
2. Add your code to the function "*def DoTwiceGame()*" and other required functions such as *doTwice(func, x).*
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 4 test cases and maximum score for this task is 5.
6. Autograder will be timed out after 10 minutes.

Task 2: Developing simple Sales Analytics

Task Description:

As a programmer, you are given a list of sales numbers from different sales departments. Based on this list of numbers, you are requested to develop one program to assistant your sales manager to analyse and get different representations of the data. The task you must perform is listed below:

a.) Given a list of sales numbers, you need to write function scale(list1, x) to return another list of scaled numbers, where list1 is the list of sales numbers and x is the scale factor. The scaled number can be calculated by using the input number multiplied by the scale factor. Below are some sample executions of the function:

```
>>> scale([10,20,30,40],2)
[20, 40, 60, 80]
>>> scale([30,40,50],0.1)
[3.0, 4.0, 5.0]
```

b.) Given a list of sales number, you are required to write one function sort(list1) to return one list of sorted sales numbers. Different to the normal sorting, the sales manager requests you to sort the number based on their last digit. Below are some sample executions of the function:

```
>>> sort([55,70,61,34,72,59]
[70, 61, 72, 34, 55, 59]
```

c.) Given a list of sales numbers, you are required to write one function goodSales(list1) to output all the good sales. The sales number is good if it is above the average of the total sales numbers.

```
>>> goodSales([10,20,40,60,20])
[40, 60]
>>> goodSales([3,2,8,6,7])
[8, 6, 7]
```

d.) Develop one program to allow users to input the sequence of the sales number and the scale factor. Then call all the functions to show use the scaled data, sorting result and good sales.

Note:

1. For functions a, b and c, you can optimize the function body with at most 2 lines codes by using filter, map, reduce, sorted and lambda expression. See how short your code can be.
2. *(Your output should be in ONE line)*

Running example:

```
C:\INF1002\Lab5\Sales> python sales.py 10,20,30,40,50,60 2
```

```
The scaled number is:  [20, 40, 60, 80, 100, 120] The sorted sales
numbers are:  [10, 20, 30, 40, 50, 60] The good sales numbers are:
[40, 50, 60]
```

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "SalesAnalytics.py" from the "Python\Lab\Lab5" xSiTe folder.
2. Add your code to the function "*def SalesAnalytics()*" and other required functions.
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 the two tasks together:

After testing all the above two tasks, submit (drag and drop) all three files *DoTwiceGame.py*, and *SalesAnalytics.py* together and auto graded for a maximum score of 10 for the 2 tasks.

Note: It is not necessary to zip these files.