Lab/Homework 2

Deadline: 23:59 pm, Sunday, Nov 12.

What to submit:

A report with answers to each exercise and corresponding python program (.py file), packaged into a zip file. Named zip as “class_ name_HW2 ",for example, “AI1-刘晔-HW2”. Please submit TA (Jiashuo Zheng)

Requirements on Coding:

1. Adding header to each .py file.

```
"""
xxxx.py
author:
date:
description:
"""
```

2. Please add a space around the operator and after the comma.
3. Add a blank line between code of different functions
4. Indent your code blocks with 4 spaces. Never use tabs or mix tabs and spaces.

Exercise 2.0 Multiple factorials (15pts)

Create a new program called mulfactorials.py, and write code to print the first N factorials, in reverse order, N is the positive whole number. In other words,

if N = 10, then the program that prints 10!, then prints 9!, then prints 8!, ..., then prints 1!. Its literal output will be:

3628800
362880
40320
5040
720
120
24
6
2
1

Hint: Use two nested for loops.

The outer loop sets the value of k to the values N, N-1, N-2, ..., 1, in succession.

Then, the body of that loop is itself a loop — exactly your solution to factorial k!.

In the report, please show diagrams and results of two testing cases with necessary explanations.

Exercise 1.1 Math module (15pts)

In this exercise, we will play with some of the functions provided in the package math. Documentation for the math module is available at http://docs.python.org/release/2.6.6/library/math.html
Create a program called playmath.py, write code to compute the following values using package math and print out the result:

1. $angletest = sin(\frac{\pi}{4}) + \frac{cos(\frac{\pi}{4})}{2}$
2. $ceillingtest = \lceil \frac{276}{19} \rceil + 2log_7(12)$
3. Write a function named MathTest that has one argument and calculate the following

$$y = xe^{-x} + \sqrt{(1 - e^{-x})}$$

If you don't know how to define function, refer to the following:

```
def MathTest(x):
    #your code
    return y
```

Hint: If you are unfamiliar with the notation $\lceil * \rceil$, this represents the ceiling of a number. The ceiling of some float x means that we always "round up" x. For example, $\lceil 2.1 \rceil = \lceil 2.9 \rceil = 3.0$. Look at the math module documentation for a way to do this!

If everything is working correctly, show the results of some testing cases in your report. For example, the output is:

sin(pi/4) + cos(pi/4)/2 is: 1.06066017178
ceiling(276/19) + 2 log_7(12) is: 17.5539788165
MathTest(5) is 1.0303150673.

Exercise 1.2 Calculate float (30pts)

Create a program called calculatefloat.py, and write code to implement high-precision operations on float numbers.

1. (10pts) Use package decimal, given two float numbers num1and num2 represented as string, return a float number which is high-precision

summation and difference of two float numbers. The definition of function is defined as:

```
# return high-precision summation of two float numbers using decimal package as a float number
def DecimalAdd(num1, num2):
    #your code
    #sum = num1 + num2
    return sum

# return high-precision difference of two float numbers using decimal package as a float number
def DecimalSub(num1, num2):
    #your code
    #diff = num1 - num2
    return diff
```

Please write your code to implement high-precision operations on float numbers using package decimal. In the report, please show results of some testing cases and corresponding necessary explanations. For example:

1. print(DecimalAdd("0.1", "0.2") == 0.3)
2. print(DecimalSub("0.4", "0.3") == 0.1)
3. print(DecimalSub("0.3", "0.2"))
4. print(DecimalAdd("6.4" , "6.2"))

2. (10pts) Input two float numbers num1, num2, represented as string, return high-precision summation of two float numbers. Do not use package decimal. The definition of function is defined as:

```
# return high-precision summation of two float numbers without using package decimal as string
```

```
def FloatAdd(num1, num2):
    #your code
    #total = num1 + num2
    return total
```

Please write your code to replace the red code in the function FloatAdd. In the report, please show results of some testing cases and corresponding necessary explanations. For example:

1. print(FloatAdd("0.1","0.2") == "0.3")
2. print(FloatAdd("6.4", "6.2"))
3. print(FloatAdd("0.7", "4.4"))

3. (5pts) Input two float numbers num1, num2, return high-precision difference of two float numbers. Do not use package decimal. The definition of function is defined as:

```
# return high-precision difference of two float numbers without package decimal as a float number
def FloatSub(num1, num2):
    #your code
    #subtraction = num1 - num2
    return subtraction
```

Please write your code to replace the red code. In the report, please show results of some testing cases and corresponding necessary explanations. For example:

1. print(FloatSub(0.4, 0.3) ==0.1)
2. print(FloatSub(0.3, 0.2))
3. print(FloatSub(6.4 , 6.2))

Hint: The operation on two int numbers is precise.

Exercise 1.3 Find int roots of cubic equation(20pts)

Create a program called rootsofcubic.py, write the function FindIntRootsOfCubic(a,b,c,d) that takes the int or float coefficients a, b, c, d of a cubic equation of this form:

$$y = ax^3 + bx^2 + cx + d$$

You are guaranteed the function has 3 real roots, and in fact that the roots are all integers. Your function should return these 3 roots in increasing order. Define a function as:

```
def FindIntRootsOfCubic(a,b,c,d):
    #your code
    return root1, root2, root3
```

To get started, you'll want to read about Cardano's cubic formula here. Then, from that page, use this formula:

$$x = \{q + [q^2 + (r-p^2)^3]^{1/2}\}^{1/3} + \{q - [q^2 + (r-p^2)^3]^{1/2}\}^{1/3} + p$$

where:

$$p = -b/(3a), \quad q = p^3 + (bc-3ad)/(6a^2), \quad r = c/(3a)$$

This isn't quite as simple as it seems, because your solution for x will not only be approximate (and not exactly an int, so you'll have to do something about that), but it may not even be real! Though the solution is real, the intermediate steps may include some complex values, and in these cases the solution will include a (possibly-negligibly-small) imaginary value. So you'll have to convert from complex to real (try c.real if c is complex), and then convert from real to int.

Great, now you have one root. What about the others? Well, we can divide the one root out and that will leave us with a quadratic equation, which of course is easily solved. A brief, clear explanation of this step is provided here. Don't forget to convert these to int values, too!

So now you have all three int roots. Great job! All that's left is to sort them. Now, if this were later in the course, you could put them in a list and call a built-in function that will sort for you.; But it's not, so you can't. Instead, figure out how to sort these values using the limited built-in functions and arithmetic available this week. Then just return these 3 values and you're done.

In the report, please show results of some testing cases and corresponding necessary explanations. For example:

print(FindIntRootsOfCubic(2, 6, 12, 10))

Exercise 1.5 Saving plan (20pts)

PartA (10pts)

You have graduated from SCUT and now have a great job! You move to the Guangdong–Hong Kong–Macau Greater Bay Area and decide that you want to start saving to buy a house. As housing prices are very high in the Bay Area, you realize you are going to have to save for several years before you can afford to make the down payment on a house. We are going to determine how long it will take you to save enough money to make the down payment given the following assumptions:

1. Call the cost of your dream home **total_cost.**

2. Call the portion of the cost needed for a down payment **portion_down_payment**. For simplicity, assume that portion_down_payment = 0.25 (25%).
3. Call the amount that you have saved thus far **current_savings**. You start with a current savings of $0.
4. Assume that you invest your current savings wisely, with an annual return of r (in other words, at the end of each month, you receive an additional **current_savings*r/12** funds to put into your savings – the 12 is because r is an annual rate). Assume that your investments earn a return of r = 0.04 (4%).
5. Assume your annual salary is **annual_salary.**
6. Assume you are going to dedicate a certain amount of your salary each month to saving for the down payment. Call that **portion_saved**. This variable should be in decimal form (i.e. 0.1 for 10%).
7. At the end of each month, your savings will be increased by the return on your investment, plus a percentage of your **monthly salary** (annual salary / 12).

Write a program named savemoneyA.py to calculate how many months it will take you to save up enough money for a down payment. You will want your main variables to be floats, so you should cast user inputs to floats.

Your program should ask the user to enter the following variables:

1. The starting annual salary (annual_salary)
2. The portion of salary to be saved (portion_saved)
3. The cost of your dream home (total_cost)

Hints

To help you get started, here is a rough outline of the stages you should probably follow in writing your code:

● Retrieve user input. Look at input() if you need help with getting user input. For this problem set, you can assume that users will enter valid input (e.g. they won't enter a string when you expect an int)

- Be careful about values that represent annual amounts and those that represent monthly amounts.

Try different inputs and see how long it takes to save for a down payment.
Please make your program print results in the format shown in the test cases below.

Test Case 1

```
>>>
Enter your annual salary: 120000
Enter the percent of your salary to save, as a decimal: .10
Enter the cost of your dream home: 1000000
Number of months: 183
>>>
```

Test Case 2

```
>>>
Enter your annual salary: 80000
Enter the percent of your salary to save, as a decimal: .15
Enter the cost of your dream home: 500000
Number of months: 105
>>>
```

PartB(10pts)

In Part A, we unrealistically assumed that your salary didn't change. But you are an SCUT graduate, and clearly you are going to be worth more to your company over time! So we are going to build on your solution to Part A by factoring in a raise every six months.
your program to include the following

1. Have the user input a semi-annual salary raise **semi_annual_raise** (as a decimal percentage)
2. After the 6th month, increase your salary by that percentage. Do the same after the 12th month, the 18th month, and so on.

Write a program named savemoneyB.py to calculate how many months it will take you to save up enough money for a down payment. Like before, assume that your investments earn a return of **r** = 0.04 (or 4%) and the required down payment percentage is 0.25 (or 25%). Have the user enter the following variables:

1. The starting annual salary (annual_salary)
2. The percentage of salary to be saved (portion_saved)
3. The cost of your dream home (total_cost)
4. The semi-annual salary raise (semi_annual_raise)

Hints

To help you get started, here is a rough outline of the stages you should probably follow in writing your code:

- Be careful about when you increase your salary – this should only happen **after** the 6th, 12th, 18th month, and so on.
- Try different inputs and see how quickly or slowly you can save enough for a down payment.

Please make your program print results in the format shown in the test cases below.

Test Case 1

```
>>>
Enter your starting annual salary: 120000
Enter the percent of your salary to save, as a decimal: .05
Enter the cost of your dream home: 500000
Enter the semi-annual raise, as a decimal: .03
Number of months: 142
```

>>>

Test Case 2

>>>

Enter your starting annual salary: 80000
Enter the percent of your salary to save, as a decimal: .1
Enter the cost of your dream home: 800000
Enter the semi-annual raise, as a decimal: .03
Number of months: 159

>>>

Test Case 3

>>>

Enter your starting annual salary: 75000
Enter the percent of your salary to save, as a decimal: .05
Enter the cost of your dream home: 1500000
Enter the semi-annual raise, as a decimal: .05
Number of months: 261

>>>

In the report, show results of some testing cases for part A and part B.