6.00 Problem Set 1

Handed out: Friday, September 7, 2018
Due: 5:00 PM, Friday, September 14, 2018

Objectives
- Introduction to control flow in Python
- Formulating a computational solution to a problem
- Exploring bisection search

Overview
- Save the **three** problems as follows: **ps1a.py**, **ps1b.py** and **ps1c.py**
-
- Include **comments** to help us understand your code— read the style guide for more specific instructions regarding this.
- There is a tester file for you to run and test your code with

Collaboration
- Students may work together but each student should write up and hand in their assignment separately
- You must include the names of your collaborators in a comment at the start of each file

Important Notes
- Read the style guide sections 1, 2, and 3.
- Declare variables in the order stated in the pset. Failure to do so will result in a 0 by the grader.
- **If files are named incorrectly, you will receive a 0 on this problem set because the grader will not run properly**

Part A: House Hunting

You have just graduated from MIT and have a job! You move to the Bay Area and decide that you want to start saving for a home. Houses are fairly expensive, so you start saving up for the down payment on your dream home. Your goal is to find the number of months this takes.

User Inputs
Ask the user to enter the following variables and cast them as *floats*. They must be initialized in the following order at the beginning of your program, before declaring other variables.
1. The starting annual salary (**annual_salary**)
2. The portion of salary to be saved (**portion_saved**). This variable should be in decimal form (i.e. 0.1 for 10%).
3. The cost of your dream home (**total_cost**)

Writing the Program

You will to determine how many months it will take given the following information:

1. **annual_salary**, as described above.
2. **portion_saved**, as described above.
3. **total_cost**, as described above.
4. The total cost portion needed for a down payment is **portion_down_payment**. For simplicity, assume portion_down_payment = 0.18 (18%).
5. The amount that you have saved thus far is **current_savings**, which starts at $0.
6. You get annual return of **r**. In other words, at the end of each month, you receive an additional *current_savings*r/12* funds for your savings (the 12 is because **r** is an annual rate). Assume your investments earn a return of r = 0.03 (3%).
7. At the end of each month, your savings increase by a percentage of your monthly salary and the monthly return on your investment. (*Note*: investment here is your savings before the end of this month)
8. Call the number of months required **months**.

Notes

- Initialize variables with the names given in bold in the order given. You can add additional variables as needed, but they must come after those listed in bold. Not doing so will result in the problem set being graded incorrectly.
- Be careful about values that represent annual amounts versus monthly amounts.
- Assume that users enter valid inputs (e.g. no string inputs when expecting an int)
- **Your program should print in the same format as the test cases below.**

Test Case 1

```
>>>
Enter your annual salary: 112000
Enter the portion of your salary to save, as a decimal: .17
Enter the cost of your dream home: 750000
Number of months: 78
>>>
```

Test Case 2

```
>>>
Enter your annual salary: 65000
Enter the portion of your salary to save, as a decimal: .20
Enter the cost of your dream home: 400000
Number of months: 62
>>>
```

Test Case 3

```
>>>
Enter your annual salary: 350000
Enter the portion of your salary to save, as a decimal: .3
Enter the cost of your dream home: 10000000
Number of months: 167
>>>
```

Testing

- Open the file **ps1_tester.py** and uncomment lines 5 and 6 by erasing the #
- Run this file in the same folder as **ps1a.py** and you should pass the first 3 test cases

Part B: Saving, with a raise

In Part A, we assumed that your salary didn't change over time. But you are an MIT graduate, and clearly you are going to be worth more to your company over time! In this part, we will build on your solution to Part A by factoring in a raise every six months. Begin **ps1b.py** by copying over your solution to Part A.

User Inputs
There is one additional user input in Part B. Again, remember to cast these inputs as *floats* and in the following order before declaring other 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**)
4. The semi-annual salary raise (**semi_annual_raise**), a decimal percentage

Writing the Program
Write a program to calculate how many months it will take you save up for a down payment. You can reuse much of the code from Part A. Like before, assume that your investments earn a return of **r** = 0.03 (or 3%) and the required down payment percentage is 0.18 (or 18%). In this version, increase your salary by **semi_annual_raise** after the 6^{th} month. Do the same after the 12^{th} month, the 18^{th} month, and so on.

Notes:

- Initialize variables with the names given in bold in the order given. You can add additional variables as needed, but they must come after those listed in bold.
- Be careful about values that represent annual amounts versus monthly amounts.
- Assume that users enter valid inputs (e.g. no string inputs when expecting an int)
- Raises should only happen **after** the 6^{th}, 12^{th}, 18^{th} month, and so on.
- **Your program should print in the same format as the test cases below.**

Test Case 1

```
>>>
Enter your starting annual salary: 110000
Enter the percent of your salary to save, as a decimal: .15
Enter the cost of your dream home: 750000
Enter the semi-annual raise, as a decimal: .03
Number of months: 76
>>>
```

Test Case 2

```
>>>
Enter your starting annual salary: 350000
Enter the percent of your salary to save, as a decimal: .3
Enter the cost of your dream home: 10000000
Enter the semi-annual raise, as a decimal: .05
Number of months: 114
>>>
```

Testing

- Open the file **ps1_tester.py** and uncomment lines 7 and 8
- Run this file in the same folder as **ps1b.py** and you should pass the first 5 test cases

Part C: Finding the interest rate you need to succeed

In Part B, you explored how both the percentage of your salary saved each month and a semi-annual raise affected how time it took to save for a down payment, given a fixed rate of return, **r**, on investment. Now, suppose you want to set a particular goal, e.g. how much should I have in my savings in order to be able to afford the down payment in three years? For an **initial_deposit**, what should **r** be in order to reach your goal of a sufficient down payment in 3 years?

User Inputs
Cast the user inputs as a *float* in the beginning of your program.

1. The initial amount in your savings account (**initial_deposit**)

Writing the Program
Write a program to calculate what **r** should be in order to reach your goal of a sufficient down payment in 3 years, given an **initial_deposit**. You should be able to reuse some of your code from Part B. To simplify things, assume:

1. The cost of the house that you are saving for is $950K
2. The down payment is 0.32 (32%) of the cost of the house

You will use *bisection search* to try to determine the *lowest* return on savings, **r**, needed to achieve a down payment on a $950K house in 36 months. Since hitting this exactly is a challenge, we simply want your savings to be within $100 of the required down payment. You should keep track of the number of steps it takes your bisection search to finish.

Assume that **r** lies somewhere between 0% and 100%. Because we are searching for a value that is in principle a float, we will limit our answer to two decimals places of accuracy (i.e., for a return of 7.0414%, you should return 7.04%, or 0.0704 as a decimal). This means we can use bisection search to search for an integer between 0 and 10000 (using integer division), and then convert it to a decimal percentage (using float division) when we are calculating the **current_savings** after 3 years. By using this range, we have a finite number of numbers to search over as opposed to an infinite number of decimals between 0 and 1 (this also helps to prevent infinite loops!).

Use the following formula for compounded interest in order to calculate the predicted down payment for a given rate of return:

current_savings = **initial_deposit** * (1 + **r**/12)^(number_of_months)

Try different inputs for your **initial_deposit** to see how **r** changes in order to reach your desired down payment. Find the *lowest* return on investment you need in order to save up enough money for your down payment in three years. Save this value as **best_r**, a decimal (e.g. 0.0704 for 7.04%). Also return **steps**, the number of steps your bisection search took. In the case where it is not possible to save up a down payment in 3 years given a salary, your program should notify the user with a print statement, and your **best_r** should be equal to the string 'no r possible'.

Notes

- You can assume that the initial deposit will be always less than down payment + $100.
- There may be multiple rates of return on savings that yield a savings amount that is within $100 of the required down payment on a $950K house. In this case, you can

just return any of the possible values.

- Initialize variables with the names given in bold. The following variable must be initialized in the following order, at the beginning of your program, before declaring any other variables:
    - **initial_deposit**
- Retrieve user input.
- Initialize variables with the names given in bold. These variables must be initialized as the **first few lines of the program** otherwise the grading software will not work. You can add additional variables as needed.
- Depending on your stopping condition and how you compute a trial value for bisection search, your number of steps may vary slightly from the example test cases. The grader takes this into account.
- Note that the return value for **best_r** is different than what is printed to the user!
- **If a test is taking a long time, you might have an infinite loop! Check your stopping condition.**

Test Case 1

```
>>>
Enter the initial deposit: 65000
Best savings rate: 0.5253 [or very close to this number]
Steps in bisection search: 9 [or very close to this number]
>>>
```

Test Case 2

```
>>>
Enter the initial deposit: 303900
Best savings rate: 0.0002 [or very close to this number]
Steps in bisection search: [May vary based on how you implemented your bisection search]
>>>
```

Test Case 3

```
>>>
Enter the initial deposit: 15000
Best savings rate: It is not possible to pay the down payment in three years.
Steps in bisection search: [May vary based on how you implemented your bisection search]
>>>
```

Testing

- Open the file **ps1_tester.py** and uncomment lines 9 and 10
- Run this file in the same folder as **ps1c.py** and you should pass all eight test cases

Hand-in Procedure

1. Naming Files

Save your solution to part A as **ps1a.py**, to part B as **ps1b.py**, and to part C as **ps1c.py**.

2. Time and Collaboration Info

At the start of each file, in a comment, write down the number of hours (roughly) you spent on the problems in that part, and the names of your collaborators. For example:

```
# Problem Set 1A
# Name: Jane Lee
# Collaborators: John Doe
# Time Spent: 3:30
# Late Days Used: 1 (only if you are using any)

… your code goes here …
```

3. Submit

Before you upload your files, remove all debugging print statements and other commented out code that is not part of your solution. In addition, open a new console in Spyder and rerun your programs. Be sure to run the tester file as well and make sure the tests all pass. The tester file contains a subset of the tests that will be run to determine the problem set grade.

To submit a file, upload it to the **Problem Sets link on Stellar**. You may upload new versions of each file until the 5PM deadline, but anything uploaded after that time will be counted towards your late days, if you have any remaining. If you have no remaining late days, you will receive no credit for a late submission.

To submit a pset with multiple files, you may submit each file individually through the submission page. Be sure to title each submission with the corresponding problem number.

After you submit, please be sure to check the problem set page for the file we have in your submission folder and their contents. When you upload a new file with the same name, your old one will be overwritten.